Л. А. РАСТРИГИН

ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ, СИСТЕМЫ, СЕТИ...

МОСКВА «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
1982

32.81
Р 24
УДК 62-50

Растригин Л. А.
Р 24 Вычислительные машины, системы, сети...— М.: Наука. Главная редакция физико-математической литературы, 1982.—224 с., илл.—40 коп.

Книга в доступной форме знакомит читателя с вычислительными машинами, вычислительными системами и сетями из этих машин. Это новые инструменты решения задач, выдвигаемых научно-технической, хозяйственной и просто индивидуальной практикой. Вычислительные машины, возникшие под давлением потребностей человечества, развиваются, изменяются и трансформируются под тем же давлением, образуя вычислительные системы, а в последнее время — вычислительные сети — самые удивительные машины нашего времени. Изложение ориентировано на «пользователя», то есть потребителя, который хочет использовать вычислительные средства для решения своих задач и в праве знать, что можно потребовать от компьютера, а что и нельзя. Автор не скрывает трудностей и иногда противоречий развития вычислительной техники и иллюстрирует их наглядными примерами.

Книга ориентирована на широкий круг читателей, желающих ознакомиться с современными компьютерами, системами компьютеров и сетями из них и, может быть, использовать их для решения своих насущных задач.

Р $\frac{1502000000-156}{053(02)-82}$ 169-82

ББК 32.81
6Ф0.1

Леонард Андреевич Растригин

ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ, СИСТЕМЫ, СЕТИ...

Редактор *Л. А. Чульский*
Технический редактор *В. Н. Кондакова*
Корректоры *Л. И. Назарова, М. Л. Медведская*

ИБ № 12027

Сдано в набор 21.04.82. Подписано к печати 23.11.82. Т-20937. Формат 84×108$^1/_{32}$. Бумага тип. № 2. Гарнитура литературная. Высокая печать. Усл. печ. л. 11,76. Уч.-изд. л. 12,54. Тираж 100 000 экз. Заказ 184. Цена 35 коп.

Издательство «Наука»
Главная редакция физико-математической литературы
117071, Москва, В-71, Ленинский проспект, 15

Ленинградская типография № 2 головное предприятие ордена Трудового Красного Знамени Ленинградского объединения «Техническая книга» им. Евгении Соколовой Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли, 198052, г. Ленинград, Л-52, Измайловский проспект, 29.

Р $\frac{1502000000-156}{053(02)-82}$ 169-82

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Писать книгу об ЭВМ — и легко и трудно. Легко потому, что об ЭВМ уже много писали и говорили, так что читатель уже подготовлен к восприятию этой тематики, тем более, что он все чаще сталкивается с ЭВМ и их деятельностью при оплате счетов, начислении зарплаты, заказе билетов и т. д. Трудно потому, что уже никого не удивишь огромным быстродействием ЭВМ, их малыми размерами и большой памятью.

Именно поэтому в книге ни слова не сказано о том, как устроен современный компьютер (хотя эта тема безусловно интересна и поучительна). Описание ведется по его функциям, которые могут быть реализованы, вообще говоря, разными схемами и способами. Изложение сопровождается примерами, которые автору представлялись наглядными.

В небольшой книжке нельзя описать все аспекты использования дискретной вычислительной техники. Да автор и не стремился к этому. Поэтому специалисты наверняка обнаружат «дыры» в изложении специфики ЭВМ, систем и сетей, особенно в той области, где работают они сами. Эти «дыры» неизбежны и не должны смущать читателя. У другого автора они будут на других местах, так как нельзя одной книгой исчерпать все затронутые вопросы.

Бессмертную мысль Козьмы Пруткова о том, что «нельзя объять необъятное» сейчас следовало бы расширить словами «...и нецелесообразно». Именно эта нецелесообразность и определила состав представленного в книге материала.

Автор по роду своей профессиональной деятельности является специалистом в области адаптации, что, естественно, не могло не оказать влияния на подбор материала книги. Некоторый перекос в сторону адаптации здесь связан еще и с тем, что эти идеи только входят в область вычислительной техники и поэтому несколько

подозрительно воспринимаются ее специалистами. Эхо этой полемики читатель может услышать в каждой главе.

Говорят, что на Диком Западе в салунах у стойки бара висело объявление «В бармена просят не стрелять!». Это предупреждение было особо своевременным, когда «полемика» между посетителями принимала очень личный характер.

В вычислительной технике, как и любой другой быстро развивающейся области знаний, часто возникают весьма острые дискуссии, в процессе которых часто достается тому, кто пытается излишне внятно изложить свои взгляды. Научно-популярная книга является именно таким поводом. А поэтому «В автора просят не стрелять!».

И последнее. Пафос этой книги не в восхищении компьютером, хотя он и достоин восхищения. Пафос в его замене другим, более совершенным, удобным, адаптивным.

Книгу в рукописи много читали и обсуждали до тех пор, пока она не приняла настоящий вид. Автор признателен всем, кто принял в этом участие. Особые усилия приложили рецензент книги Д. И. Поспелов, редактор Л. А. Чульский, а также Е. В. Гливенко, Я. А. Гельфандбейн, Н. А. Левин, А. Б. Розенблит, А. Н. Скляревич, Ф. П. Тарасенко, В. В. Шкварцев. Их замечания и советы были учтены в максимальной мере.

Глава 1

ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ

«Дважды два — четыре,
дважды два — четыре,
это знают дети в целом мире,
а не три и не пять,
это надо знать».

Детская песенка

Одним из самых забавных парадоксов нашего времени является то, что знаменитые вычислительные машины — самые сложные и самые нужные машины нашего времени — менее всего вычисляют. Время, затрачиваемое на вычисления современной ЭВМ, составляет едва ли не 10% от всего времени ее работы.

«Что же она делает в остальное время?» спросит возмущенный читатель. В остальное время она перерабатывает информацию и занята этим туманным занятием 90% своего «рабочего дня».

Вот и сказано слово, которое вот уже более тридцати лет по праву считают самым важным (и модным) словом в современной жизни — науке, технике, народном хозяйстве и... повседневных заботах. От информации не уйти, она повсюду настигает нас и больно наказывает за пренебрежение и неуважение, проявляемое к ней.

Что же такое «информация» и зачем понадобилось ее «перерабатывать» да еще применять для этого такие сложные машины — ЭВМ?

Стоит ли говорить, что на этот вопрос ответить коротко очень трудно. И чем короче будет ответ — тем он будет ошибочнее. Одним из таких «ответов» может служить следующий: информация это то, что каждый из нас интуитивно понимает под этим словом, а «переработка» — что-то из области технологии, где в качестве исходного материала выступает не металл, дерево или что-либо еще, а эта самая «информация».

Слов нет, функция вычислений и поныне остается одной из важнейших не только в науке, но и в хозяйстве (напомним хотя бы проблему народно-хозяйственного планирования, требующую ежегодно миллиардов вычислений), и тем не менее не вычислениями едиными

живо человечество. Его потребности в переработке информации значительно больше.

Дело в том, что вычисления являются лишь одним частным случаем переработки информации, но никак не наоборот. Вычислять — это значит перерабатывать информацию. Но перерабатывать информацию — это вовсе не только вычислять. К переработке информации относятся, например, ее передача от одного абонента к другому, преобразование из одной формы в другую, запоминание, поиск определенных данных в большом информационном массиве, моделирование процессов и многое многое другое. При выполнении каждой из этих функций могут использоваться вычисления — но это лишь их вспомогательная роль, и применяются эти вычисления потому, что машина вычислительная.

Для того чтобы разобраться в этом, давайте попробуем ответить на старый-престарый вопрос...

Что же такое информация?

Забегая вперед, следует откровенно сказать, что вполне внятного ответа не будет. Вина здесь лежит не только (или точнее: не столько) на авторе, сколько на довольно неопределенном состоянии самого этого понятия. Об информации до сих пор много говорят, пишут, спорят и... до сих пор не приходят к единой точке зрения, приемлемой для всех. Одни видят в информации одно, другие — совсем иное. Эту ситуацию не следует считать трагической. Наличие разных мнений по одному и тому же поводу чаще всего говорит о том, что мы имеем дело с плохо изученным явлением. Причин этому может быть много. Среди них основной является сложность этого явления. Именно таким сложным объектом является информация.

Различные точки зрения на информацию составляют основу для многочисленных теорий информации, выражающих отдельные аспекты этого сложного явления. Рассмотрим некоторые из них.

Начнем с простейшей ситуации — выясним, дома или нет наш приятель? Будет ли при этом получена информация и какая?

Естественно считать, что информация будет получена: раньше не знали, позвонили — узнали. Это и есть информация, точнее: информация есть прирост знания.

Следующий вопрос: сколько информации мы получим. Естественно связать объем получаемой информации со степенью неожиданности. Чем более неожиданно сообщение, тем больше оно содержит информации. Если приятеля не обнаружим дома в рабочее время, то информация, полученная при этом, не будет большой. А вот если ночью?

Таким образом, объем информации связывается с вероятностью события: чем менее вероятное событие произошло, тем больше информации содержит сообщение о том, что оно совершилось.

Но и этого мало. На объем информации влияет не только вероятность события, но и заинтересованность в нем человека, получившего это сообщение. Так узнав, что Ивана Ивановича, незнакомого вам, не оказалось ночью дома, вы останетесь равнодушны, так как при этом полученная вами информация не будет велика. (Малая заинтересованность здесь разве что заставит вежливо спросить, а кто такой этот Иван Иванович.) Если же им окажется ваш сотрудник, который должен быть в это время дома (ну, например, для того, чтобы получить телефонограмму о результатах испытаний разработанного вами прибора), то эта информация будет огромной.

Другой аспект измерения количества информации связан с ее последствиями для вашего поведения. Если эти последствия не требуют изменения поведения, то информации мало или вообще нет. Но если, получив сообщение, вам придется, сломя голову, бежать куда-то, чтобы что-то срочно сделать, то можно смело сказать, что полученная информация велика.

А теперь обратимся к вычислительным машинам, перерабатывающим информацию. Переработка информации машиной, как всякий регулярный процесс преобразования, позволяет получить на ее выходе нечто такое, что не подавалось на вход ЭВМ. Так, при решении задачи $2 \times 2 = ?$ на вход машины подаются две двойки и операция умножения. Они не содержат искомой четверки, которая является результатом работы машины по решению указанной задачи. Поэтому можно считать, что ЭВМ вырабатывает информацию, т. е. создает новую информацию.

В последней фразе не случайно использовано осторожное «можно». Дело в том, что есть и другая точка зрения на процесс переработки информации в

помощью ЭВМ. Она опирается на понятие тавтологии. При этом регулярные преобразования лишь меняют форму представления информации, не внося ничего принципиально нового, т. е. являются тавтологическими. «2 × 2» и «4» — пример двух тавтологических записей.

Если стоять последовательно на этой позиции, то машина производит новую информацию лишь в том случае, когда ошибается.

Столь большое различие во взглядах на переработку информации с помощью ЭВМ оказалось возможным только потому, что мы пока не имеем единого взгляда на феномен информации. Различные точки зрения поразному освещают это явление и соответственно дают различное толкование и оценку процесса переработки информации.

Хорошо это или плохо? Сказать трудно. Конечно, чисто по-человечески хочется иметь одну модель изучаемого явления.

Когда на вопрос «Что это такое?» получаешь много ответов, то невольно подозреваешь своего собеседника в некомпетентности. Пока науке удавалось давать однозначные ответы. Однако совсем недавно появились объекты, относительно которых не существует (возможно — пока!) единых представлений. Такова, например, природа света, двойственность которой постулируется в физике. Сосуществуют (хотя и не без трений) две теории происхождения нефти — органическая и неорганическая. Феномен информации относится именно к этому классу явлений. Разница лишь в том, что теорий информации не две и не три. Их число уже перевалило за десяток.

В дальнейшем мы не будем использовать какие-либо строгие определения информации. Будем подразумевать под этим очень емким понятием то, что мы подразумеваем в обыденной жизни — новые сведения, нужные кому-либо или чему-либо. (Заметим, что подтверждение старых сведений новыми данными тоже несет информацию, но лишь в том случае, если были сомнения в этих старых сведениях. Так, например, сообщение о том, что камень в эксперименте упал вниз, не содержит информации, так как в законе всемирного тяготения мы пока не сомневаемся. При появлении таких сомнений это сообщение будет восприниматься с

удовлетворением, как подтверждение того, что этот закон еще действует.)

А теперь рассмотрим вычислительную функцию ЭВМ. Сейчас она относительно незначительна, а раньше, лет 15—20 назад, была основной, что и определило название ЭВМ. Заметим, что указанная «незначительность» вовсе не означает неважность этой функции. Вычисления являются очень важной функцией в научно-технической и экономической практике общества. Поэтому давайте рассмотрим как потребность в вычислениях «довела» человечество до изобретения ЭВМ. Это короткая, но поучительная история.

Зачем нужно много и быстро считать?

Потребность в вычислениях возникла у человека давно. И так же давно он придумал много хитрых и ловких приемов ускорения счета (известный способ вычислений «столбиком» является одним из таких замечательных изобретений).

По мере роста потребностей человека росла и необходимость в вычислениях. Эта необходимость заставила искать пути механизации счета и привела к изобретению арифмометра.

Вращая ручку арифмометра можно очень быстро (по сравнению с вычислениями на бумаге) делать арифметические операции с многозначными числами. Но и это вскоре перестало удовлетворять. Тогда к ручке арифмометра подключили электромотор. Произошло это совсем недавно — лет пятьдесят назад.

Казалось, что быстрее считать нельзя! Действительно, десятизначные числа перемножались в секундное время, а на их деление (это самая сложная арифметическая операция) затрачивались две-три секунды! Такая ошеломляющая скорость вычислений давала повод для радужных надежд. Казалось, что эта скорость будет достаточной для всех и надолго.

Но суровая действительность очень быстро заставила искать новые пути убыстрения счета. Вторая мировая война! Она поставила человеческую жизнь (летчика, артиллериста и т. д.) в зависимость от быстроты вычислений. Кто быстрее и точнее принимал решения (а именно для этого нужно было вычислять), тот и побеждал!

Эту жесткую дилемму стоит рассмотреть специально, так как именно она заставила человечество изобрести ЭВМ и тем самым вступить в новый век — век вычислительных машин. Это вовсе не означает, что без войны век ЭВМ не наступил бы. Описанная тенденция привела бы к созданию ЭВМ в любом случае. Война лишь обострила потребность в быстром счете и тем самым интенсифицировала работы по созданию электронных машин для вычислений (так же, как освоение атомной энергии).

Ситуация, сложившаяся во второй мировой войне, сводилась к следующему. Огромные скорости движения боевых машин нападения (самолетов, танков, кораблей и т. д.) заставляли разрабатывать эффективные средства активной защиты. Таким средством всегда была артиллерия. Но в условиях повышенных скоростей боевых машин ей пришлось решать очень сложную проблему (да, да, именно проблему, а не задачу!) — куда целиться в момент выстрела? Читатели-охотники наверное будут сильно удивлены простотой этого вопроса. Им приходится решать очень похожую задачу при стрельбе «влет». И решают они ее просто — стреляют, упреждая движение дичи: на корпус или два вперед (в зависимости от скорости и расстояния). Эта простейшая стратегия линейной экстраполяции движения цели довольно естественна и вполне оправдана для «неразумных» целей, которые не учитывают поведение стрелка. Охотникам пришлось бы очень туго, если бы дичь догадывалась о последствиях выстрела и принимала бы контрмеры против линейного упреждения.

Именно в такой ситуации оказывается наводчик зенитного орудия. Ему еще хуже — ведь упреждать самолет приходится на десять-двадцать корпусов! А за это время летчик может сделать так называемый «противозенитный маневр», целью которого является изменение прямолинейного движения самолета, которое легко экстраполируется наводчиком зенитного орудия. А так как летчик не заинтересован в том, чтобы наводчик угадал, как он изменит направление полета, то этот маневр он делает наиболее замысловатым образом. В этом случае эффективность линейного упреждения падает до нуля.

Именно такая ситуация сложилась в ходе войны между Англией и фашистской Германией. Самолеты гитлеровского Люфт-ваффе почти безнаказанно бомбили города и военные объекты Великобритании, нанося

при этом огромный военный и материальный ущерб. Перед союзниками возникла острейшая проблема повышения эффективности зенитной стрельбы. Наводчик с этой задачей перестал справляться — слишком велики стали скорости самолетов и их маневренность. Надо было автоматизировать эту функцию.

За решение этой проблемы взялись крупнейшие умы науки того времени. Среди них был Норберт Винер — известный американский математик.

Он рассуждал следующим образом. Для эффективности стрельбы необходимо предвидеть будущее положение самолета, то есть экстраполировать его траекторию на некоторое время вперед, чтобы снаряд и самолет одновременно прилетели в «точку встречи». Однако точно угадать будущую траекторию самолета невозможно, так как летчик, выполняющий противозенитный маневр, может поступить достаточно произвольно. Поэтому траекторию самолета следует, по крайней мере для артиллериста, считать случайной (летчик может заранее предусмотреть свой маневр). Эта случайность появилась здесь из-за отсутствия информации о том, как поступит летчик.

Таким образом, автомат, ведущий стрельбу, должен предвидеть поведение случайной траектории. А можно ли экстраполировать случайные кривые? Именно такую задачу поставил Н. Винер как математик.

На первый взгляд он взялся за совершенно безнадежную задачу, так как угадать, как пройдет случайная траектория, попросту нельзя. На то она и случайная. Точное предсказание здесь невозможно. Но при стрельбе зенитным снарядом особой точности и не требуется — достаточно снаряду разорваться в нескольких десятках метров от самолета. Да и случайности в траектории самолета не так много, так как его маневр ограничен физическими законами движения самолета и его конструкцией. Оставшуюся неопределенность можно отнести на счет случайности, которую можно предсказывать лишь приближенно, статистически. Эту задачу и решил Н. Винер. Он указал, как по наблюдению траектории полета самолета до выстрела орудия можно определить положение его ствола в момент выстрела, при котором вероятность поражения будет максимальна. Такой ответ вполне устраивал, так как всем было ясно, что о поражении наверняка речи быть не может.

Способ, предложенный Н. Винером, требовал большого объема вычислений, которые необходимо было сделать за те мгновения, пока самолет приближается к цели — 2—3 секунды. Здесь, как легко заметить, никакой арифмометр не справится, даже если его снабдить самым быстрым электромотором. Здесь были нужны новые принципы вычислений, исключающие механику вращающихся колес арифмометра. Была нужна электроника! И электроника пришла на помощь.

Интенсивные исследования ученых и инженеров привели к появлению ЭВМ — электронной вычислительной машины. Ее создание связано с именем другого известного математика фон Неймана (венгра по происхождению). Он поставил задачу шире, как задачу автоматизации вычислений с помощью электронного автомата, работающего по задаваемой ему программе. Эта программа сначала вводилась в память автомата (ее тогда называли «запоминаемой программой»), после чего автомат выполнял вычисления по этой программе. Каждая программа как бы настраивала автомат на решение определенной вычислительной задачи. Такая машина обладает новым свойством — универсальностью. По одной программе она может управлять зенитным огнем, а по другой — составлять оптимальный план перевозок грузов на транспорте и т. д.

Принципы, разработанные фон Нейманом, используются и поныне при создании современных вычислительных систем (об этом будет рассказано ниже).

Первая электронная машина заработала в 1945 году и была названа ENIAC (аббревиатура слов Electronic Numerical Integrator And Calculator, что в дословном переводе значит: электронный численный интегратор и калькулятор). Она выполняла 5000 операций в секунду и имела память емкостью всего 20 десятизначных чисел. А дальше пошли поколения машин, которые быстро сменяли друг друга. Каждое новое поколение отличалось от предыдущего качественно новыми свойствами. На каждое поколение уходило в среднем 10 лет.

Первое поколение

Это поколение отличалось применением электронных ламп и сравнительно низким быстродействием — 10—20 тыс. арифметических операций (таких, как сложение, умножение и т. д.) в секунду. В Советском Союзе

к первому поколению относится первая отечественная вычислительная машина МЭСМ (Малая Электронная Счетная Машина), созданная в 1951 г. в г. Киеве под руководством академика С. А. Лебедева, серийные машины Минск-1, Стрела, БЭСМ (Большая Электронная Счетная Машина), Урал-1, Урал-4 и др. Любопытно, что одной из самых острых проблем эксплуатации этих машин был отвод тепла. Тысячи электронных ламп создавали такую температуру, с которой нелегко было справиться. Во всяком случае отопление зимой в машинных залах не включали.

Машины первого поколения использовались только для решения вычислительных задач научного характера. Машины ускоряли счет в рамках существовавших тогда методов, разработанных для «ручных» расчетов.

Процесс программирования на этих машинах требовал большого искусства и изворотливости — нужно было хорошо представлять, как устроена ЭВМ и каким образом она реагирует на ту или иную ситуацию.

Второе поколение

Это поколение отличалось тем, что вместо громоздких и горячих электронных ламп стали употребляться миниатюрные и «теплые» транзисторы.

Быстродействие машин второго поколения возросло до сотен тысяч операций в секунду. Стала использоваться библиотека стандартных программ. Появилась возможность программирования на так называемых алгоритмических языках, значительно облегчивших процесс программирования (позже мы расскажем о них). Первой полупроводниковой машиной была модель RCA-501, появившаяся в 1959 г. В Советском Союзе к этому поколению относятся машины Минск-2, Минск-22, Минск-32, БЭСМ-2, БЭСМ-4, БЭСМ-6 и др. (Заметим, что БЭСМ-6, делающая миллион операций в секунду, великолепно служит до сих пор и используется для решения научно-технических задач).

Третье поколение

Машины этого поколения создаются на так называемых интегральных схемах. Если раньше электронную схему собирали из транзисторов, емкостей и сопротивлений (резисторов), то теперь все эти элементы об-

разовывались на специальным образом сделанной кристаллической многослойной пленке. Делая лазером в этой пленке различной глубины разрезы, оказалось возможным создавать миниатюрные схемы с огромной плотностью элементов. Такие схемы названы интегральными. На одном квадратном миллиметре такой схемы оказалось возможным расположить свыше тысячи ее элементов. Основные узлы вычислительных машин третьего поколения могли разместиться, таким образом, в нескольких спичечных коробках.

Если вы заглянете внутрь корпуса ЭВМ третьего поколения (без ее внешних устройств), то обнаружите, что он далеко не полностью «забит» электроникой, как в ЭВМ первого и второго поколения. Дело в том, что его габариты определяются не электронной начинкой, а пультом, где располагается сигнализация и органы управления ЭВМ, которые делать миниатюрными нельзя, так как оператору станет трудно работать с ЭВМ.

Вычислительные машины третьего поколения, как правило, образуют серии (семейства) машин, совместимых программно. Такая серия состоит из ЭВМ, производительность и объем памяти которых возрастают от одной машины серии к другой. Но программа, отлаженная на одной из машин серии, может быть сразу запущена на другой машине этой серии (и во всяком случае на машинах большей мощности).

Первым таким семейством машин третьего поколения была выпущенная в 1965 году вычислительная система IBM/360. Она имеет свыше семи моделей.

В Советском Союзе такую серию машин третьего поколения образует семейство ЕС ЭВМ (Единая Система ЭВМ), в значительной части совместимая с IBM/360.

Четвертое поколение

Машины этого поколения уже не просто машины, а так называемые многопроцессорные вычислительные системы с быстродействием в десятки и сотни миллионов операций в секунду. Они предназначены в основном для коллективного использования вычислительных мощностей.

Такими машинами стали B-7700 фирмы Барроуз, Иллиак-IV, созданный в Иллинойском университете, Эльбрус, разработанный в Советском Союзе. Принци-

пы, используемые в этих машинах, мы рассмотрим во второй главе.

Этим счет поколений не заканчивается. Говорят о пятом, спорят о шестом, фантазируют о седьмом поколениях машин. Едва ли стоит говорить о них — это пока проекты и мечты. А реально существуют сейчас машины — немного — второго, в основном — третьего, и чуть-чуть — четвертого поколений.

А теперь давайте познакомимся с ЭВМ.— Нет, нет! Не с ее устройством — об этом мы говорить не будем, так как технические подробности часто отпугивают. Будем знакомиться с ЭВМ как с неким неодушевленным лицом, способным совершать вполне логические (но не логичные в житейском смысле) действия при общении с вами (здесь автор намеренно не использовал более точное понятие — автомат, каким и является всякая ЭВМ).

Итак — ЭВМ как неодушевленное лицо.

Этикет общения с ЭВМ

Как всякое сложное устройство любая ЭВМ требует определенных правил общения с ней, нарушать которые нельзя. На легкие нарушения она реагирует легкими упреками (например, указывает на допущенные ошибки), а на грубые — не отвечает вообще.

Поэтому давайте начнем знакомство с вычислительной машиной с правил общения.

Всякий процесс обработки информации вообще и вычислений в частности требует введения в ЭВМ исходной информации. Эту информацию можно условно, но довольно четко подразделить на два вида, отвечающие на вопросы: что обрабатывать (считать) и как это делать. Первый вид информации называют «данные», а второй — «программа». О программе стоит поговорить подробнее.

Программа

В программе пользователь (таким скучным словом называют человека, решающего свою задачу на ЭВМ) на специальном языке, доступном машине, описывает, каким образом ей следует обработать исходные данные, чтобы добиться результата, нужного пользователю.

С подобными программами мы очень часто встречаемся в жизни. Так, обычная кулинарная книга составлена из «программ», которые здесь названы рецептами.

Например, рецепт пражского салата записывается следующим образом. «Данные»: 150 г жареной телятины, 150 г жареной свинины, 150 г соленых огурцов, 150 г лука, 100 г яблок, майонез, лимонный сок или уксус. «Программа» — все твердые компоненты нарезают ломтиками, заливают лимонным соком или уксусом и смешивают с майонезом.

Иную «программу» сообщает мастер слесарного дела своему ученику: «Возьмешь ту штуковину и приставишь вот так к той, что побольше. Затем соединишь их этим болтом». Здесь «данными» являются две «штуковины»: гайка и болт, а «программа» указывает на то, как их соединять.

Или другой пример. Когда мы просим кого-либо из близких сходить за хлебом, лекарством и кефиром, то мы сообщаем ему лишь исходные данные. Программу же, то есть последовательность посещения булочной, аптеки и молочной, он составляет сам.

Заметим, что программы, предназначенные для ЭВМ, должны быть составлены куда более точно, чем программы, предназначенные для выполнения человеком. Составляя «программу» для человека, мы рассчитываем, что он сам домыслит то, что не было нами сказано, или переспросит, если что неясно. А при составлении программы для ЭВМ нужно учесть все до последней мелочи. Поэтому существенной особенностью любой программы является наличие в ней нескольких «запасных» вариантов, определяющих, как поступать при различных обстоятельствах. Чем тщательнее разработана программа, тем больше в ней предусмотрено вариантов — ветвей программы. Такие ветвления мы часто используем в жизни: «... а если нет кефира, купи простоквашу».

Читатель, наверное, уже догадался, что все, на что способна ЭВМ, зависит от того, какие программы в нее заложены, какую программу она в данный момент выполняет и какие данные эта программа использует в своей работе. Следовательно, более точно было бы говорить не об умной машине, а об умной программе, заложенной в эту машину. (А если уж быть совсем точ-

ным, то следует говорить об умном программисте, который составил эту программу. Но умного программиста можно встретить значительно чаще, чем «умную» программу. Именно поэтому можно, хотя и с оговоркой, сказать, что интеллект ЭВМ определяется ее программой).

Калькулятор — простейшая ЭВМ

Знакомство с ЭВМ начнем с простейшей вычислительной машины, называемой электронным калькулятором. В таком калькуляторе отражены почти все основные принципы современной ЭВМ. Рассмотрим их.

Электронные калькуляторы являются, по сути своей, электронными арифмометрами. Они сейчас в изобилии продаются в отделах канцтоваров наших магазинов. Это маленькие карманные электронные машинки, способные очень быстро выполнять несколько простейших арифметических действий (сложение — вычитание, умножение — деление, возведение в квадрат — извлечение квадратного корня и т. д.). Для этого достаточно с помощью определенных кнопок или клавиш ввести в калькулятор исходные числа (эти кнопки перенумерованы числами от 0 до 9) и указать (также соответствующей кнопкой) ту операцию, которую следует над ними произвести. Ответ почти мгновенно будет высвечен на миниатюрном цифровом дисплее (экране). Здесь вводимые числа являются исходными данными, а программа задается одной командой — «сложить», «вычесть», «умножить» и т. д. Как видно, все атрибуты ЭВМ здесь имеются, только в элементарном виде.

Четкую границу, разделяющую ЭВМ и калькуляторы, провести очень трудно. Простейшие калькуляторы могут выполнять только четыре арифметических действия. Более сложные имеют набор сменных программ, которые изменяют производимые калькулятором операции. Например, вычисление сложных процентов, перевод измерений из одной шкалы в другую (градусы Цельсия в градусы Фаренгейта и т. д.) и др. Эти программы (точнее: программки) записаны на маленькие кусочки магнитофонной ленты (примерно 1×5 см) и называются магнитными картами. Пропустив такую карту через калькулятор и тем самым введя в него записанную на карту программу, можно задавать исходные данные, которые будут переработаны в соответст-

вии с введенной программой. Например, программа вычисления сложных процентов требует исходных данных в виде трех чисел — годового процента, исходной суммы и срока ее хранения, а результатом вычислений будет конечная сумма.

И, наконец, есть калькуляторы, на которых можно составлять собственную программу. Пусть вам понадобилось часто вычислять результат по какой-то формуле при различных значениях ее параметров. (Примером такой задачи может служить задача о биоритмах. Задавая дату рождения и решая эту задачу, можно вычислить коэффициенты интеллекта, эмоциональности и здоровья на любое число. Автор не является сторонником такого рода предсказаний, но и не исключает возможных их позитивных применений, например, как фактора психологического воздействия.) Вы составляете на калькуляторе программу вычислений по этой формуле, куда входят параметры, которые могут изменяться (например, даты) и записываете ее на магнитную карту. Всякий раз, когда вам понадобится посчитать по этой формуле, введите эту карту в калькулятор и задайте с помощью клавиш исходные параметры. (Для программы биоритмов всего одно число — дату, когда вас интересует ваше самочувствие, дата рождения уже запрограммирована на карте). Результат будет немедленно обозначен на табло (дисплее).

Как видно, такой калькулятор по праву можно назвать маленькой ЭВМ. Он имеет изменяющуюся программу и память, отличающие «настоящие» ЭВМ.

Более того, калькулятором называют и очень сложные вычислители, имеющие большую скорость вычислений, огромную память, многосимвольный дисплей и т. д. По своим характеристикам он почти ничем не отличается от ЭВМ средней мощности и при этом имеет габариты портативной пишущей машинки.

Так что, пожалуй, единственным формальным отличием калькулятора от ЭВМ является его портативность и транспортабельность (обычно каждый калькулятор снабжается удобным чехлом для его транспортировки).

Удобство и возможности подобных калькуляторов были быстро оценены потребителями и... фирмами-производителями ЭВМ. Сейчас на международный рынок выпускается огромное количество электронных калькуляторов самых разнообразных типов. Многие

из них могут соединяться с обычным домашним телевизором и кассетным магнитофоном. Телевизор выполняет роль дисплея и на его экран калькулятор выводит всю информацию, необходимую пользователю (окончательный и промежуточный результаты, графики и т. д.), а также задает вопросы и отдает распоряжения пользователю (например, «ВВЕДИТЕ ДАННЫЕ» или « =?»). Кассетный магнитофон играет роль внешней памяти калькулятора. Здесь пользователь может хранить свои программы и информационные материалы (например, библиографию книг и статей по своей специальности).

Можно подключить калькулятор и к телефону. И тогда он сможет связываться с другими калькуляторами и большими ЭВМ, подключенными к телефонной сети (например, за получением информации, необходимой пользователю). Об этом будет рассказано во второй и третьей главах.

Так или иначе, но современный калькулятор по своим возможностям практически ничем не отличается от малых и средних ЭВМ, хотя значительно меньше и дешевле их. Это обстоятельство и обеспечивает ему успех.

А теперь рассмотрим трудности общения с ЭВМ. Именно они часто заставляют создавать новые типы ЭВМ.

Языковые трудности (общение с ЭВМ)

Язык, как известно, является прежде всего средством общения. Он изменяется и очень сильно в зависимости от специфики общающихся лиц, состояния среды, целей общения и т. д. Действительно, наш обычный разговорный язык является великолепным средством непосредственного общения людей в «нормальных» условиях. Достаточно ухудшить условия (например, ввести шум) и язык изменится — он станет более избыточным, усилится роль жестов и мимики, которые образуют тоже язык, эмоционально очень выразительный, хотя и мало информативный при передаче обычных (неэмоциональных) сообщений.

Если общение собеседников происходит по почте, они используют эпистолярный язык, сильно отличающийся от разговорного. Он менее избыточен, более «правилен» и может быть поэтому суховат. Вспомните, как трудно воспринимается на слух доклад, читаемый

по «шпаргалке». Здесь использование «чужого» языка явно затрудняет общение и понимание.

А теперь рассмотрим общение человека и ЭВМ. Оно так же происходит на языке. Каким должен быть этот язык? «Разумеется, разговорным!» — ответит человек. ЭВМ будет иного мнения. Но стоит ли считаться с этим «мнением»? — Стоит и очень даже! Для того чтобы ЭВМ понимала разговорный язык, ее следует снабдить соответствующей программой. А именно эта проблема создания программы понимания машиной разговорного языка является пока нерешенной проблемой. Мы (пока!?) не знаем, как должна быть устроена программа, с помощью которой машина будет способна понимать или научиться пониманию живого разговорного языка. (Имеется опыт создания программ, позволяющих ЭВМ понимать вопросы, заданные пока на очень узкую тему, например, о расписании движения самолетов, о наличии изделий на складе и т. д.).

Одной из причин трудностей является размытость и многозначность нашего языка. Это значит, что используемые нами в разговоре слова не могут быть строго определены, смысл их размыт и расплывчат, да при этом еще и многозначен. (Так, например, слово «большой» явно размыто и точный его метрический смысл зависит от многих факторов и обстоятельств. Например, большие глаза и большие ворота имеют явно разный размер.)

Другой не менее веской причиной является влияние жизненного опыта, определяющего специфику понимания того или иного слова. Мы, люди, сравнительно хорошо понимаем друг друга не столько потому, что говорим на одном языке, сколько потому, что живем в одном мире. Машина не имеет такого опыта и поэтому в принципе не может хорошо понимать нас, людей. Вот и получается, что разговора «по душам» с ЭВМ у нас, пожалуй, долго не получится. А пока...

...Послушаем другую сторону — ЭВМ. С ее «точки зрения» человек должен общаться с ней на ее машинном языке. Словами этого языка являются команды вида «сложить», «умножить», «записать» и т. д. А грамматикой (точнее синтаксисом) — правила, как следует использовать эти команды. Так, например, одним из таких грамматических правил является требование указать три числа после команды «сложить». Первые два числа указывают номера ячеек памяти, откуда следует

брать слагаемые, а третье — номер ячейки, куда нужно направить результат сложения.

Очевидно, что в этом языке отражена специфика устройства именно той ЭВМ, с которой предстоит «общаться», т. е. давать ей свое задание на обработку информации. Очевидно, что таких машинных языков существует ровно столько, сколько типов ЭВМ.

Как видно, для того чтобы общаться с ЭВМ, нужно либо знать ее машинный язык, либо... искать переводчика. Таким переводчиком раньше был программист — человек, специальностью которого было умение писать программы на машинном языке. Знание машинных языков для него, естественно, было обязательным.

Вот так и образовалась цепочка: пользователь — программист — машинная программа — ЭВМ, которая много лет тормозила широкое применение ЭВМ. Действительно, если Вам понадобилось решить задачу на ЭВМ, то нужно было найти программиста или стать им, чтобы составить машинную программу для решения этой задачи. Оба варианта требуют больших затрат энергии и настойчивости. Именно это обстоятельство — необходимость в программировании задач на машинном языке — и ограничивало использование ЭВМ.

Так возникла острая необходимость в создании такого промежуточного языка между человеком и ЭВМ, который бы достаточно легко воспринимался как человеком, так и машиной. И такие языки были созданы, их назвали алгоритмическими языками.

С чем связано многообразие этих языков (число их давно перевалило за сотню)? Здесь проявился эффект, замеченный еще М. В. Ломоносовым. Помните, как он отметил, что по разным поводам лучше использовать различные языки (разговорные, разумеется): с врагами лучше говорить на немецком, с женщинами — на итальянском и т. д. Конечно, и с женщинами можно успешно объясняться на немецком. Но итальянский — лучше. В нем есть необходимые для этого (разговора с женщинами) средства, которых нет в других языках. Вот и получается, что хотя все национальные языки универсальны, употреблять их лучше, ориентируясь на проблемную ситуацию.

Точно такое происходит и с алгоритмическими языками при их создании. Дело в том, что, строго говоря, можно было бы обойтись одним универсальным языком. Его универсальность гарантировала бы возмож-

ность написания любых программ. Но, как всякое универсальное средство, этот язык не был бы удобным для решения конкретных проблем. Всякая проблема обладает своей спецификой, которую можно учесть в языке программирования при решении задач этой проблемы. Такой язык будет очень удобен именно для той проблемы, на которую он ориентирован. Другие же проблемы он в принципе может обслуживать, но, естественно, хуже. Так и появились проблемно-ориентированные языки.

Это означает, что каждый из них позволяет эффективно решать проблемы своего класса. Так проблему формульных вычислений решают языки алгол, фортран, ПЛ/1 и др. Проблему обработки списков — язык ЛИСП, моделирования — GPSS и т. д. Такого рода языки называют *языками высокого уровня.* Они максимально приближены к человеческому языку, но, разумеется, не совпадают с ним. Овладение таким языком требует несколько дней или недель (а не месяцев, необходимых для того, чтобы овладеть машинным языком).

Итак, человек доволен! А как машина? Ведь она «понимает» лишь машинные программы, написанные на ее машинном языке!

Для того чтобы и ЭВМ понимала языки высокого уровня, для нее необходим переводчик с этого языка на ее машинный язык. Таким переводчиком является *транслятор*, т. е. программа, которая преобразует программу, написанную на языке высокого уровня, в машинную программу. Так образуется более удобная для пользователя цепочка: пользователь — программа на языке высокого уровня — транслятор — машинная программа — ЭВМ.

Эта цепочка буквально сделала революцию в применении ЭВМ, а следовательно, и в их развитии. Число активных пользователей стало быстро расти, так как процесс программирования упростился и, что тоже очень важно, не надо было идти «на поклон» к программисту.

А что же программисты? Остались без куска хлеба? Нет! Им дел даже прибавилось. На их плечи легли проблемы так называемого системного программирования, например такие, как сложнейшая задача создания транслятора. Современный программист — это прежде всего системный программист. Если теперь массовому

пользователю не надо знать специфику каждой конкретной машины (программы на языках высокого уровня никак не связаны со спецификой той машины, на которой будет решаться задача, об этом позаботится транслятор), то программист по-прежнему представляет «интересы» ЭВМ и следит, чтобы она с максимальной эффективностью и правильно была использована (транслятор и выполняет некоторые из этих функций).

Так произошло разделение функций пользователя и ЭВМ. Теперь пользователь ЭВМ не зависит от ее конкретной конструкции и может решать свою задачу на любой другой (лишь бы она имела транслятор с того языка, на котором написана программа пользователя). Важность этого события трудно переоценить. Ее можно проиллюстрировать на следующем простом примере. Представьте, что для того, чтобы воспользоваться телефоном нужно было бы знать его устройство (когда-то это было так). Можно с уверенностью сказать, что число пользователей телефоном было бы небольшим. И только независимость его функций от устройства стало залогом массовой телефонизации. Если бы не это, телефон до сих пор был бы прибором для «избранных», т. е. для специалистов в области связи.

Именно так ЭВМ из машины для «избранных», т. е. для программистов, превратилась в машину для всех. И произошло это в результате применения языков высокого уровня и трансляторов.

Правда, надо признаться честно, что овладение языком высокого уровня, сложнее, чем телефоном. Но ведь и задача при этом решается более сложная.

Внимательный читатель, наверно, удивится, а зачем все это здесь рассказывается? Ведь книга написана для пользователя, которому совершенно не интересна «внутримашинная кухня» с транслятором и машинной программой. Не все ли ему равно, работает машина по машинной программе или по программе высокого уровня! Результат ведь будет одинаковым!

Эти соображения верны до тех пор, пока мы не учитываем реального положения вещей. А оно требует, чтобы транслятор не был бы слишком велик. Ведь он должен располагаться в памяти конкретной ЭВМ. А на него кроме трансляции программы, написанной на языке высокого уровня, в машинную программу возлагаются функции выявления ошибок в программе, а иногда (для очень мощных трансляторов) и исправление

этих ошибок. Естественно, что создание такого транслятора является чрезвычайно трудной задачей, в процессе решения которой попросту нельзя предусмотреть все коллизии, которые могут сложиться в программе пользователя (составляют программу транслятора те же программисты, которые тоже могут ошибаться, точнее — что-то недоучесть).

Вот и получается, что программа, составленная пользователем, не работает не по вине пользователя, а по вине транслятора. Достаточно ее немного изменить и транслятор выдает правильную машинную программу. У пользователя создается впечатление, что машина «не любит» какие-то программы. А не любит их не машина, а транслятор.

Иногда для одного и того же языка высокого уровня имеются несколько трансляторов разной мощности. Самый мощный предусматривает многое, но поэтому и работает медленно. Его-то в связи с этим и не любят использовать (время машины всегда дорого). Вот и получается, что часто предпочтение отдается быстрому транслятору, который какие-то огрехи в программе допускает. Это нужно знать пользователю, которому приходится выбирать транслятор для своей программы, если, разумеется, он (транслятор) не один.

Но прежде чем выбирать транслятор, пользователь должен иметь представление о процессе создания программ, т. е. о программировании. Проще всего это сделать на конкретном языке (высокого уровня) и примерах.

Давайте ознакомимся с элементарными основами одного из самых распространенных языков высокого уровня — фортрана (известно, что 80% пользователей предпочитают этот язык другим).

Элементарный фортран

Свое название (FORTRAN) этот язык получил в результате сокращения двух английских слов FORmula TRANslator, т. е. «формульный переводчик». А по-русски: фортран есть ФОРмульный ТРАНслятор (напомним, что транслятор и есть переводчик). Он был создан в 1956 г. в США.

Причины появления фортрана, кроме тех, которые изложены выше (стремление к независимости программы от машины), были сугубо экономического, даже фи-

нансового, порядка. Нужно было обеспечить загрузку машинными программами мощных ЭВМ, которые появились к тому времени. Каждую такую машину должны были загружать работой сотни программистов, высокая квалификация которых требовала высокой оплаты. А дефицит специалистов в области программирования создавал неизбежный ажиотаж вокруг них и требовал дальнейшего увеличения их зарплаты (чтоб, грубо говоря, не перебегали в другую фирму, где больше платят).

Вот и получилось, что изобретение фортрана свалило бремя программирования на самих пользователей, что сразу решило проблему загрузки ЭВМ.

А теперь о фортране. Начнем с формул, для которых он и был предназначен. Программа вычислений здесь почти не отличается от привычной нам общепринятой записи. Лучше всего это пояснить на примере.

Пусть нужно вычислить выражение

$$z = (ax + y)^b - \frac{c}{w},$$

где величины a, b, c, x, y, w заданы. Фортран-программа в этом случае будет иметь вид

```
Z = (A * X + Y) ** B - C/W.
```

Легко заметить, что звездочкой обозначается произведение, двумя звездочками (**) возведение в степень, косая черта — деление. Все переменные в фортране обозначаются большими буквами или любым их сочетанием, включая цифры (например, X1, Y18, Z7A, IKS, FI40, PSI, DELTA9 обозначают соответственно привычные нам x_1, y_{18}, z_{7a}, x, φ_{40}, ψ, Δ_9).

Очень часто приходится использовать оператор условного перехода. Например, при $X \geqslant B$ следует делать одно (например, выполнять ту часть программы, которая отмечена цифрой 5), а в обратном случае другое. Фортран-программа такого условного перехода записывается так:

```
IF (X ⩾ B) GOTO5,
```

т. е. «Если (IF) $x \geqslant b$, то иди на (GOTO) метку 5 (эта метка поставлена перед соответствующей строкой программы). В противном случае ($x < b$) нужно переходить к следующему оператору в программе».

Используем этот оператор для решения квадратного уравнения $ax^2 + bx + c = 0$. Его решение $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ зависит от знака дискриминанта $\Delta = b^2 - 4ac$. При $\Delta < 0$ корни мнимые: $x_{1,2} = \frac{-b \pm i\sqrt{-\Delta}}{2a}$, где $i = \sqrt{-1}$. Различный знак дискриминанта Δ требует при вычислении корней применения оператора условного перехода, так как считать приходится по разным формулам.

Фортран-программа в этом случае принимает, например, такой вид (справа в комментарии приведена ее расшифровка в обычных обозначениях):

	Комментарий
DEL = B ** 2 — 4 * A * C	$\Delta = b^2 - 4ac$.
IF (DEL < 0) GOTO1	Если $\Delta < 0$, то иди на оператор, отмеченный меткой 1.
X1 = (— B + (DEL ** 0.5))/(2 * A)	$x_1 = (-b + \sqrt{\Delta})/2a$.
X2 = (— B — (DEL ** 0.5))/(2 * A)	$x_2 = (-a - \sqrt{\Delta})/2a$.
PRINT X1, X2	Печать x_1, x_2.
STOP	Прекращение счета.
1 Y1 = — B/(2 * A)	1 — метка оператора $y_1 = -b/2a$.
Y2 = (— DEL) ** 0.5/(2 * A)	$y_2 = \sqrt{-\Delta}/2a$.
Y3 = — Y2.	$y_3 = -y_2$
PRINT Y1, Y2, Y1, Y3	Печать y_1, y_2, y_1, y_3.
STOP	Прекращение счета.

Здесь введен оператор печати (PRINT), при котором ЭВМ печатает на бумаге значения указанных после этого оператора переменных. При $\Delta \geqslant 0$ будут напечатаны численные значения двух корней x_1 и x_2. А при $\Delta < 0$ — четыре числа. Первые два (y_1 и y_2) определяют первый корень: $x_1 = y_1 + iy_2$, а остальные — второй $x_2 = y_1 + iy_3$.

При этом использованы следующие обозначения:

DEL $= \Delta$,

XK $= x_k$ $(k = 1, 2)$,

YN $= y_N$ $(N = 1, 2, 3)$.

И, наконец, оператор цикла, который очень часто применяется в массовых вычислениях. Пусть, например, нужно определить среднее значение заданных чисел

$x_1, \ldots, x_N$. При вычислении суммы $s_N = x_1 + x_2 + \ldots + x_N$ естественно повторять операцию сложения

$$s_i = s_{i-1} + x_i.$$

Это делается с помощью оператора цикла, который позволяет вычисление s_N сделать с помощью такой простой программки:

```
S = 0
DO 1I = 1, N
1 S = S + X (I)
```

Здесь оператор цикла DO1 I = 1, N указывает, что надо выполнять все последующие операторы до и включая оператор с меткой 1 (до него может быть много операторов), после чего следует возвращаться к оператору цикла. И делать это N раз, изменяя переменную I от 1 до N последовательно. Оператор цикла DO1 I = 1, N таким образом означает: «Идти до метки 1 и при возвращении изменять переменную I на 1, повторяя такой цикл до I = N».

Оператор S = S + X(I) выполняет операцию вычисления по формуле $s_i = s_{i-1} + x_i$, где i = I и x_i = = X(I), т. е. добавляет к уже накопленной сумме очередное значение x. Для работы такой программы в ЭВМ необходимо предварительно ввести все числа x_i ($i = 1, \ldots, N$).

Читатель, наверное, заметил, что операторы программы пишутся на английском языке (DO, PRINT и т. д.). Причина здесь вовсе не в каких-то специальных удобствах английского языка (хотя применение одних латинских букв для записи формул и операторов языка высокого уровня создает некоторое техническое упрощение, просто меньше букв!). Причина заключается в традиции.

Дело в том, что первые ЭВМ, а следовательно и программисты, появились в США. Именно поэтому США стали «законодателем мод» в программировании. И хотя есть хороший русский вариант фортрана и других языков высокого уровня, программисты, а вслед за ними и пользователи, предпочитают английский. При этом программы понятны всему миру, с чем, разумеется, нельзя не считаться. Не говоря уже о том, что число трансляторов при этом минимально. А то пришлось бы для каждого нового «национального» алгоритмиче-

ского языка корректировать транслятор, что требует значительных интеллектуальных и экономических затрат, так как транслятор является сложнейшей программной системой.

Как вводят информацию в ЭВМ?

До сих пор мы говорили о работе ЭВМ. Но для этого прежде всего в нее необходимо ввести программу и исходные данные. Для простых задач введение информации не является проблемой. Проблема возникает при необходимости введения в ЭВМ больших массивов информации, например, отчета, книги, библиотеки и т. д. (Заметим в скобках, что основные трудности внедрения известных АСУ — автоматизированных систем управления — заключаются в недостаточной информированности ЭВМ, обслуживающей АСУ. Это связано прежде всего с неоперативным введением нужной информации в ЭВМ).

Поэтому выделим проблеме ввода информации в ЭВМ специальный параграф.

Процедура введения таких чисел и букв, образующих исходные данные и программы в ЭВМ, может быть самой разнообразной. Старым способом является использование перфокарт или перфоленты (дословно: «дырявые» карты и ленты). Здесь дырки, а точнее их расположение, кодируют определенным образом исходные данные и программу. Однако прежде всего программу и данные следует записать на специальном бланке, где для каждого знака (цифры или буквы) отводится отдельная клетка. Эти бланки отдаются в перфораторную, где молодые девушки — перфораторщицы (что в дословном переводе означает: «дыропробивальщицы») «пробьют» эту информацию в картонных карточках (размером 8×19 см) или бумажной ленте (шириной 2,5 см), получив в результате колоду перфокарт или рулончик перфоленты. Заложив их в устройство ввода (это делает оператор ЭВМ), можно очень быстро ввести данные и программу в память ЭВМ. Эту программу машина тут же оттранслирует в машинную программу, которую немедленно и выполнит, если, разумеется, транслятор не выявит ошибок, которые он не сможет исправить. Обнаружив ошибку, транслятор указывает на нее и комментирует. Все это займет считанные секунды.

В результате вы получите так называемый «листинг». Это широкая бумажная лента (шириной 42 см), на которой прежде всего будет напечатана ваша программа в том виде, как вы ее написали (точнее: в виде, в котором ЭВМ ее получила). В этой программе транслятор отметит ваши ошибки, т. е. все нарушения грамматики языка, на котором была написана ваша программа. Если ошибок не будет, то далее будут приведены результаты работы машины по этой программе. Но считать, что вы получили то, что хотели, было бы преждевременным. Ошибки могли быть в алгоритме — их-то транслятор заметить никак не может, так как он следит только за грамматикой (синтаксисом) программы, а не ее семантикой (смыслом). Бессмысленная программа может быть синтаксически правильной (как, например, фраза «овца есть волка») и виноват в этом будет не транслятор, а создатель программы.

Мы здесь рассмотрели старые и до сих пор применяемые способы введения информации в ЭВМ — перфокарты и перфоленту. Всем хорош этот способ, но требует процедуры перфорирования, которую должен выполнять человек. Процесс перфорирования чисто механический: смотри на знак (цифру или букву) в бланке программы и нажимай на клавишу с таким же знаком на перфораторе. Вот и все! Не то что интеллекта, но и элементарного соображения не нужно. При перфорировании трудно не делать ошибок, так как работа эта скучная, однообразная, нетворческая. Единственным «творчеством» здесь являются ошибки. И они неизбежны. (Заметим, что машинистка, печатающая на машинке, ошибается значительно меньше и печатает значительно быстрее, чем перфораторщица, хотя внешне они выполняют одинаковую работу. Дело здесь в том, что машинистка обычно понимает печатаемый текст, а перфораторщица — нет. Если машинистке дать печатать текст на неизвестном ей языке, то ее скорость и число ошибок станут такими же, как у перфораторщицы). Что же делать?

Как избавиться от ошибок?

Потребность избавиться от процедуры ручного перфорирования связана не только с ошибками. Надо избавиться от скучного, нудного, а поэтому непроизводи-

тельного труда перфораторщицы. Такие способы были найдены. Рассмотрим их.

Первый: создали читающий автомат, заменяющий перфораторщицу. Он имеет фотоэлектронный «глаз», считывающий каждый знак на бланке программы, написанной пользователем. В память этого автомата заложены образцы-эталоны всех знаков языка, на котором может быть написана программа и исходные данные, т. е. букв, цифр и специальных знаков (точки, звездочки, скобки и т. д.). Считанный знак автомат последовательно сравнивает с содержимым своей памяти и выбирает тот, который наиболее «похож». Именно этот эталон он объявляет значением считанного знака и отдает команду на пробивание соответствующих дырок в карте или ленте, или прямо вводит эту информацию в ЭВМ. Так он автоматизирует труд перфораторщицы.

Всем хорош такой читающий автомат: и работает быстро и ошибок не делает при условии, что бланки программ заполняются правильно, т. е. стандартными знаками. Это значит, что записывать программу на бланках нужно печатными знаками, т. е. применять для этого специальную пишущую машинку и, следовательно, обращаться надо к машинистке. А чем она отличается от перфораторщицы? Ведь перфоратор внешне очень похож на пишущую машинку, только клавиши побольше и работает помедленнее! Если же бланк программы заполняет пользователь и при этом очень старается правильно рисовать знаки, то автомат при считывании делает ошибок не больше, чем перфораторщица. При небрежном заполнении бланка число ошибок, естественно, значительно больше.

Таким образом, читающий автомат безусловно является шагом вперед, но... небольшим, так как требует большой тщательности при подготовке бланков. Наверно поэтому эти автоматы пока не получили большого распространения.

Другим способом является «проговаривание» программы перед микрофоном. Для этого изобрели автомат «слушающий», который должен «понимать» человека. Такой автомат обычно строится на базе специальной ЭВМ — слишком сложную задачу ему приходится решать. Справляется он с ней хорошо, но при условии, если он имеет дело с одним диктором.

Дело в том, что мы говорим по разному. Автомат можно настроить на определенный тембр голоса, темп и способ говорения, высоту звука. И он будет уверенно понимать говорящего. Если же диктор изменится, то нужно менять настройку, иначе неизбежны ошибки. Трудность пытаются преодолеть путем специальной процедуры подстройки автомата. Новому диктору дают начитать автомату определенный текст. Так как автомат знает текст то он соответственно корректирует себя, т. е. свои решения, так, чтобы они были правильными. Чем больше текст, тем точнее настроится автомат на данного диктора.

Все это, конечно, не очень удобно. Поэтому такие автоматы еще не применяются для введения программы и данных в ЭВМ. Проблема понимания автоматом слов, произнесенных человеком, еще научная, а не техническая проблема. Как только она будет решена, мы получим в наше распоряжение настоящего электронного собеседника, который будет нас понимать, выполнять наши распоряжения, произнесенные голосом, и отвечать на вопросы тоже голосом. Такие синтезаторы речи (так называют эти автоматы) уже производятся промышленностью. В синтезатор надо только ввести текст, который он проговаривает в заданном темпе своим бесстрастным голосом (эмоций автомат не имеет).

Делает он это следующим образом. Для простых текстов, с ограниченным числом слов, например, при произнесении цифр, в память ЭВМ вводятся записи всех нужных слов, произнесенные диктором. Эти слова выводятся ЭВМ на динамик в требуемом порядке. Примерно так работает телефонный автоответчик времени, где используется память на магнитной ленте.

Если необходимо произносить любой текст, то в память ЭВМ вводятся правила фонетики, следуя которым она произносит слова и фразы так же как ученик, начинающий изучать иностранный язык. Ведь для ЭВМ человеческая речь является иностранной. Любопытно, что подобные автоматы имеют ярко выраженный национальный характер произношения. И если его заставить произносить слова из другого языка, сделает он это с сильным акцентом. (Автору пришлось слышать автомат, сделанный в Финляндии. Русские слова он произносил с сильнейшим финским акцентом и ударением на первом слоге — такова специфика финского языка).

Так оказалось, что автомат «говорящий» уже есть, а автомата «хорошо понимающего» еще нет. (Не является ли это иллюстрацией к известной поговорке, что слушать всегда труднее, чем говорить).

Читатель уже заметил, наверное, что трудности при введении информации в ЭВМ всегда связаны с индивидуальностью пользователя. То он пишет знаки своей программы не так, как это требует машина, то не так говорит! Ей бы общаться с автоматом, «индивидуальность» которого всегда можно уложить в требуемые пределы, контролируемые ОТК цеха-изготовителя. (Известно, что индивидуальные свойства всякой машины образуются за счет разброса ее параметров).

Эта идея оказалась не такой уж бредовой. Действительно, нельзя ли потребовать от пользователя быть поаккуратней? Ну хотя бы в написании текста программы? Не посадить ли для этого пользователя за пульт пишущей машинки с тем, чтобы свою программу он не писал «шариком» или фломастером, а печатал и (уж это дело простой техники) вводил сразу свою программу в машину? Идея оказалась очень плодотворной!

Диалог с ЭВМ — зачем он?

Диалог пользователя и ЭВМ опирается на два фундамента:

— простота общения,

— польза от общения, т. е. заинтересованность пользователя в диалоге.

Первый фундамент (простота общения) требует организации простого и надежного обмена информацией между пользователем и ЭВМ. Он реализуется с помощью так называемых терминалов, или точнее терминальных (оконечных) устройств. Простейшим терминалом является электрическая пишущая машинка, с помощью которой можно просто и надежно реализовать взаимодействие пользователя и ЭВМ. Клавиатура машинки со всеми необходимыми символами позволяет пользователю вводить в ЭВМ *любые* исходные данные и *любую* программу на любом языке, включая машинный. Эта возможность гарантирует, что все, что захочет «высказать» пользователь, будет введено в ЭВМ.

На эту же машинку ЭВМ выводит ту информацию, которую от нее ждет (и требует) пользователь, или за-

дает ему вопросы, связанные с получением дополнительной информации, необходимой для решения поставленной пользователем задачи.

Возможность задавать эти вопросы вовсе не означает повышенной интеллектуальности машины. (Известно, что способность задавать вопросы в определеном смысле требует большей интеллектуальности, чем отвечать на них — вопреки известной поговорке о дураке, озадачившем сто мудрецов. Дурак не тот, кто задает много вопросов известным мудрецам, а тот, кто не может их задать). Эти вопросы обычно запрограммированы заранее и находятся в памяти ЭВМ. В определенной ситуации, сложившейся во время диалога с пользователем (эта ситуация должна быть также предусмотрена и запрограммирована), машина извлекает из своей памяти вопрос, связанный с этой ситуацией и печатает его на электрической пишущей машинке пользователя.

Опишем типичную ситуацию, складывающуюся в процессе общения ЭВМ с не слишком опытным пользователем. Он обычно забывает (или не знает как) ввести исходные данные для решения его задачи. Так, например, при решении квадратного уравнения необходимо задать его коэффициенты (см. пример на стр. 28) *a*, *b* и *c*. Получив программу, описанную на стр. 28, ЭВМ не сможет ее выполнить. Эта ситуация (отсутствие данных) предусмотрена заранее и из памяти ЭВМ будет извлечена первая реплика машины:

М: ВВЕДИТЕ ДАННЫЕ. (Здесь буквой М обозначена машина. Пользователя мы будем обозначать буквой П.).

Пусть пользователь не знает как это делать (если б знал, то немедленно ввел бы). В таком случае его реакция будет однозначной:

П:? (он нажал клавишу со знаком вопроса, признавая свою беспомощность).

Эта ситуация так же предусмотрена, и ЭВМ задает вопрос (точнее три):

М: A = ?
B = ?
C = ?

Этот вопрос не извлечен из памяти, а построен ЭВМ исходя из ситуации (в программе не хватает трех чисел: A, B и C). Но процедура построения вопроса также находилась в памяти машины.

Пусть наш пользователь совсем... малоинтеллектуален и не понял, что от него требуется и снова реагирует:

П: ?

Тогда ЭВМ дает конкретный пример введения данных:

М: ПРИМЕР ВВЕДЕНИЯ ДАННЫХ:
A = 0.346
B = 5.87
C = —76.3

Здесь значения чисел и знаков взяты случайно (у ЭВМ есть программа, генерирующая случайные числа).

Если и после этого пользователь задаст вопрос

П: ?

то машина повторит предыдущий ответ, но с другими случайными числами и знаками. Действительно, нет смысла дальше мучить пользователя разъяснениями — ему они уже не помогут (возможно, что он не знает русского языка и не понимает, что от него добивается машина). А примеры правильного поведения всегда найдут отклик у пользователя, даже самого малоквалифицированного. (Известно, что наглядные примеры часто оказываются значительно более эффективной мерой, чем глубокие и подробные разъяснения). Теперь на каждый «?» пользователя ЭВМ будет отвечать примерами — это предусмотрено в ее программе, как реакция на сложившуюся ситуацию (больше двух «?» пользователя).

Из этого диалога хорошо видно, что об интеллекте ЭВМ здесь говорить не стоит, хотя ведет себя она вполне разумно.

Пользователи бывают разные...

Заметим, что пользователи, то есть лица, обращающиеся к ЭВМ за решением своих проблем, отличаются не только именами. Одни (о них говорилось выше) составляют программы, с помощью которых ЭВМ дает ответы на вопросы этих пользователей. ЭВМ в этом случае выступает как автомат, на котором пользователь «проигрывает» свою программу. Это инструмент переработки информации — не более. Новой информации (говоря строго) этот пользователь не получит.

Но есть и другой тип пользователя, который обращается к ЭВМ только за новой информацией, которой располагает машина. Его иногда называют конечным поль-

зователем (почему, неизвестно). Будем его называть так и мы, хотя бы для того, чтобы отличить его от пользователя первого типа.

В отличие от него конечный пользователь не составляет программ и не хочет этого. Он лишь задает вопросы ЭВМ или отвечает на ее вопросы. Примером конечных пользователей являются лица, обращающиеся к ЭВМ за справками различного рода по различным поводам. Например, по поводу расписания движения транспорта, по поводу нужного адреса, своих недугов или потребностей и многого другого, что волнует нас в наш беспокойный век.

Очевидно, что число конечных пользователей значительно больше, чем людей, умеющих программировать, они более требовательны и менее терпеливы. С появлением конечных пользователей у создателей ЭВМ забот заметно прибавилось.

Действительно, нужно было обеспечить надежный диалог между ЭВМ и таким (скажем прямо) совсем неквалифицированным пользователем, который еще и капризен (в силу своей неосведомленности). Эта нелестная характеристика конечного пользователя вовсе не обидна для него, а определяет его специфику с точки зрения его партнера по диалогу — ЭВМ.

Именно для такого конечного пользователя и был придуман метод «меню», название и суть которого заимствованы из сферы обслуживания самых насущных потребностей и поэтому хорошо знакомы самому конечному пользователю. Познакомимся с ним.

Диалог способом «меню»

Конечный пользователь отличается еще и тем, что он обременен своими заботами, чужд преклонения перед вычислительными машинами, боится программирования и уж очень горькая нужда заставляет обращаться к ЭВМ.

Пусть эта нужда медицинская — у вас заболел, ну, скажем (извините), живот. Зная понаслышке о возможных последствиях, превратностях («острый живот» и проч.) и опасностях самолечения вы решили пойти проконсультироваться у своего врача, районного, разумеется. Усевшись в конце длинной очереди вы с тревогой оглядываетесь и видите нечто вроде телевизора. Экран телевизионный, но вместо ручек настройки

только три кнопки с надписями «Да», «Нет» и «?». На ваш вопрос вам ответят, что это ЭК (электронный консультант). Тоскливо оглядев очередь вы, наверняка, обратитесь к этому консультанту, хотя он и электронный. Лечить он вас вроде бы не будет (манипуляторов, как у робота, у него нет), а общаться с ним с помощью трех кнопок не сложно (это не программирование).

Вы решительно садитесь за ЭК и включаете его, как телевизор. Пока трубка прогревается — это обычная телевизионная трубка — поясним, что это такое перед вами.

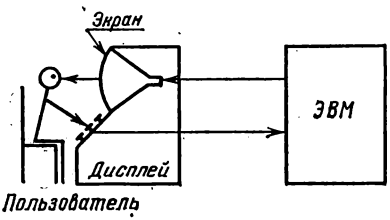

Рис. 1. Вот так происходит общение пользователя с ЭВМ через дисплей, который является посредником, обеспечивающим надежную связь человека с машиной.

Перед вами так называемый терминал вычислительной машины, которая стоит где-то поблизости (а может стоять и далеко!), т. е. «видеооконечное» устройство ЭВМ. На экран телевизионной трубки этого терминала ЭВМ может выводить тексты вопросов и пояснений, которые хранятся в ее памяти. Три кнопки также связаны с машиной.

Такой терминал обычно называется дисплеем. Всякий дисплей сочетает собой две основные функции связи человека с ЭВМ: передачу информации от ЭВМ к пользователю (обычно с помощью телевизионного изображения — текста, цифр, графиков, рисунков и т. д.) и передачу информации пользователя машине (с помощью обычной клавиатуры (см. рис. 1).

Только в данном случае консультант имеет клавиатуру лишь из трех указанных кнопок. (Обычный дисплей имеет клавиатуру из 50—70 таких кнопок).

Вот и все. Начинайте! На экране дисплея-консультанта засветился следующий текст (будем текст с экрана приводить курсивом).

1. *Я — консультант. Моя задача — консультировать всех желающих и, если нужно, собирать предварительную информацию для направления к врачу. Я понимаю только три простых ответа: ДА, НЕТ и ? (что означает «не понимаю» или «не знаю»). Поняли ли вы этот текст? Нажмите, пожалуйста, нужную кнопку.*

Вы человек любознательный и решили вступить в беседу с «консультантом» (вы-то знаете, что это ЭВМ). Ваш разговор с машиной мог быть, например, таким:

Вы: ДА (т. е. вы решили проявить свою любознательность и нажали кнопку ДА. Если бы вы нажали кнопку НЕТ, то на экране появился бы текст, разъясняющий сказанное в более простых и ясных выражениях, при ? — то же).

2. ЭВМ: *Спасибо. Позвольте задать вам несколько вопросов. Можно ли врачам узнать ваши ответы? Нажмите, пожалуйста, нужную кнопку.*

   Вы: Да. (Если бы вы возражали, то могли бы нажать кнопку НЕТ и тогда все ваши ответы были бы стерты из ее памяти после диалога. Это важное свойство по заказу полностью «забывать услышанное» очень ценно для ведения конфиденциальных разговоров и возможно только с таким собеседником, как ЭВМ. Именно это и дает вам возможность быть полностью откровенным, что чрезвычайно важно при обсуждении интимных вопросов вашего нездоровья. Если бы вы нажали «кнопку» «?», то ЭВМ объяснила бы, зачем врачам нужно знать содержание вашего диалога с ней.).
3. ЭВМ: *Скажите, является ли неприятное ощущение или боль в животе одной из причин, побудивших вас обратиться к врачу?*

   Вы: ДА (если вы ответите отрицательно, машина задаст аналогичный вопрос о голове, конечностях и т. д., а начала она с живота, так как, по свидетельству врачей, наибольшее число жалоб и болезней связано именно с этой частью тела).
4. ЭВМ: *эти боли или неприятное ощущение в животе беспокоят вас постоянно с момента их возникновения?*

   Вы: Нет.
5. ЭВМ: *Бывают ли промежутки времени — часы, дни или недели, когда боли вас совсем не беспокоят?*

   Вы: ДА.
6. ЭВМ: *связаны ли эти неприятные ощущения с приемом пищи?*

   Вы: ? (вы не поняли вопроса и поэтому затрудняетесь ответить определенно).
7. ЭВМ: *Влияют ли на ваши боли или неприятные ощущения время приема и состав пищи?*

   Вы: ДА.

8. ЭВМ: *Вы принимаете лекарства?*
   Вы: ДА. (при ответе «?» ЭВМ указала бы какие лекарства она имеет в виду).
9. ЭВМ: *Беспокоят ли вас боли или неприятные ощущения по утрам?*
   Вы: ДА.
10. ЭВМ: *Можете ли вы указать на область локализации боли?*
    Вы: ДА.
11. ЭВМ: *Укажите область живота, где вы чувствуете боль. Можете вы показать эту область одним-двумя пальцами или накрыть ладонью? Если вы можете указать на болевую область одним-двумя пальцами, нажмите кнопку ДА, если ладонью — нажмите кнопку НЕТ, если вопрос непонятен, нажмите кнопку «?».*
    Вы: «НЕТ (если вы ответите «?», машина пояснит, что значит указать пальцем или ладонью).
12. ЭВМ: *Расположена ли боль по средней линии живота?*
    Вы: ДА.
13. ЭВМ: *Вы питались нормально во время недомогания?*
    Вы: ?
14. ЭВМ: *У вас был хороший аппетит?*
    Вы: ДА.
15. ЭВМ: *Были у вас случаи рвоты?*

Не будем утруждать читателя медицинскими подробностями этой беседы — они интересуют только самого больного. Как видно, в таком диалоге машина может получить очень много сведений о специфических свойствах больного и симптомах заболевания (его возрасте, наследственности, особенностях психики и т. д.). Эти сведения дают возможность получить представление об имеющемся недомогании и по определенному решающему правилу рекомендовать что-либо: или обратиться к определенному врачу, или успокоиться по поводу напрасных опасений (известно, что значительный контингент посетителей поликлиник нуждается не в медицинской помощи, а в добром совете и успокоении, что опасения напрасны и беспочвенны). Кроме того, использование личных шифров пользователей исключит несанкционированный доступ к касающейся их информации и предохранит пациента от возможного злоумышленного введения ложных сведений о нем в память ЭВМ.

Такой диалог и реализует «принцип меню». На каждый вопрос здесь предусмотрено очень простое «меню» из трех «блюд» — возможные ответы ДА, НЕТ и ?. Иначе отвечать нельзя, так же как нельзя заказать в обычном ресторане блюдо, не включенное в меню. Составляя меню, ресторан или столовая защищают себя от непредвиденных заказов, которые возможно они не смогут выполнить. Предлагая варианты ответов на заданный вопрос, ЭВМ так же «защищает себя» от непонятных ей ответов. Как видно, меню является очень эффективным средством преодоления ЭВМ собственной слабости, которая (увы!) неизбежна.

Программа, по которой работала ЭВМ в приведенном диалоге, была составлена в Теддингтонской национальной физической лаборатории в Англии и использовалась для опроса пациентов, у которых предполагался гастрит или язва желудка. Интересно, что ни один из пациентов независимо от пола и возраста и образования, не отказался беседовать с ЭВМ.

Читатель, по-видимому, согласится, что ведение такой беседы не требует от человека никаких знаний о программировании. Достаточно, чтобы пациент понял прочитанный текст и дал правильный ответ путем нажатия одной из трех кнопок. Очевидно так же, что время, затраченное на беседу, было бы намного меньше, если бы пациенту дозволено было использовать более обширный запас слов, чем «да», «нет» и «не знаю». Однако это потребовало бы более длительного обучения пациента, и вряд ли оправдало бы себя. Кроме того, ЭВМ легче разобраться в простых ответах, ибо в них отсутствует двусмысленность, которой обычно изобилует речь человека. Существенным в примере является то, что ЭВМ сама выбирает очередной вопрос, задаваемый пациенту в зависимости от содержания его предыдущих ответов.

Итак мы установили, что в таком диалоге любой человек успешно может взаимодействовать («беседовать») с ЭВМ, ничего не зная об ее устройстве и совершенно не умея составлять программы на языках любого уровня. Это достигнуто ценой очень примитивного языка общения с ЭВМ. Легко представить, как бы реагировал человек, получая в ответ ваши «да», «нет» и «не понимаю». ЭВМ тем и отличается от человека, что волею программиста она не может раздражиться и довольствуется одним из трех ответов.

Всем хорош диалог методом «меню». Но такие разговоры возможно лишь на очень и очень узкие темы. Почему? Да потому, что машина должна реагировать на все варианты ответов собеседника. А это означает, что программист должен в нее заложить реакцию на каждый из ответов. Давайте подсчитаем, как велика должна быть память такой машины!

На каждый вопрос возможны три ответа. Каждый из ответов должен порождать новый вопрос. Сначала, после первого вопроса будет три варианта ответа, на каждый из которых необходимо иметь определенный вопрос. Таких вопросов на втором этапе будет три, затем девять, потом $3^3 = 27$ и т. д. На N-м этапе нашего разговора их будет 3^{N-1}. Много это или мало? Так, для приведенного выше простенького разговора $N = 14$, то есть в память ЭВМ необходимо заложить более миллиона вопросов! Много это или мало? Легко прикинуть. Если вопрос занимает всего две строки (иные, как показано, значительно длиннее), то миллион вопросов занимают треть Большой Советской Энциклопедии! Это допускает далеко не всякая современная ЭВМ, так как для записи всех этих вопросов требуется огромная память. А для более содержательного разговора с консультантом понадобится $N = 30$, т. е. вся Ленинская библиотека! Таких машин сейчас нет и не будет в ближайшем будущем. И это только по поводу живота!

Внимательный читатель, наверное, скажет, что не все варианты ответов необходимы в жизни, значит, не нужно и столько вопросов. Но в этом случае теория информации утверждает, что вопрос был поставлен неправильно. Действительно, вопрос, на который всегда следует один и тот же ответ, задавать не надо, поскольку ответ не несет никакой информации. Подобные риторические вопросы можно не задавать (чем, однако, часто принебрегают радио- и телекорреспонденты при интервьюировании). Чем больше неопределенности устраняет ответ, тем более правильно поставлен вопрос.

На самом деле в процессе разработки рассматриваемой программы, как правило, удается путем логического анализа немного сократить число вопросов. Например, если человек на любое разъяснение упорно «долбит» кнопку «?», требуя тем самым дополнительных пояснений, то по-видимому, он просто хулиганит или провоцирует машину. Естественно закончить этот поток знаков вопроса какой-нибудь энергичной фразой и прекраще-

нием диалога. Однако сокращение здесь будет невелико. Так или иначе, но описанный простой способ диалога с ЭВМ не решает проблемы общения человека с машиной. Он страдает двумя неизлечимыми пороками. Во-первых, для него необходимо заранее предусмотреть абсолютно все варианты всех будущих разговоров с ЭВМ, что для нетривиальной темы попросту невозможно. И даже если это удастся, т. е. будет составлено «дерево» всех вопросов на все возможные ответы (внутри данной темы, разумеется), то для размещения их в памяти современной ЭВМ понадобится слишком много средств, так как оно будет огромно. Это, однако, не исключает применения описанного способа для решения локальных задач типа «острый живот», которая была рассмотрена выше.

Метод меню может определять не ответы, а вопросы конечного пользователя, на которые ЭВМ может дать ответ. Такое меню «наоборот» принципиально ничем не отличается от описанного, так как для ЭВМ все равно: предлагать ли варианты ответов пользователю на заданный вопрос или принимать варианты его вопросов на полученный ответ. Но все же меню «наоборот» чаще используется. И причиной этому является активность конечного пользователя, который обычно предпочитает знать, какими средствами он располагает при общении с ЭВМ. Наиболее рельефно преимущество такого меню видно в информационных системах, с которыми мы сейчас и познакомимся.

Меню в информационных системах

Информационные системы, создаваемые для удовлетворения информационных потребностей пользователей (конечных, в первую голову), занимают основное время работы современного парка ЭВМ. Информация нужна всем, и ЭВМ является идеальным ее хранителем и «выдавателем», так как обращение к самой медленной памяти ЭВМ (это магнитные ленты) занимает во много раз меньше времени, чем обращение к самой быстрой неэлектронной памяти (например, на фотопленке). Именно это обстоятельство делает информационные системы незаменимым средством оперативного получения информации.

Так как абоненты информационной системы бывают самые разнообразные, то ориентируются эти системы на

самого-самого неискушенного пользователя (неискушенного в общении с ЭВМ, разумеется). Принцип меню для такого пользователя является самым подходящим.

Приведем пример использования меню в библиографической системе. Представьте, что вы пришли в библиотеку, снабженную такой системой. Здесь нет унылых рядов библиографических шкафов, заполненных тысячами библиографических карточек. Только видеотерминалы с перенумерованными кнопками. Кнопок немного больше, чем на «консультанте» в поликлинике. Их десять штук и все они имеют номера 1, 2, ..., 10.

Вы садитесь за видеотерминал, включаете его и диалог начинается с такого кадра на экране:

ЭВМ: *Библиографическая информационная система приветствует нас!*
В библиотеке имеются книги по следующим разделам:
1 — *наука*
2 — *литература*
3 — *история*
4 — *искусство*
5 — *спорт*
и т. д.
Наберите, пожалуйста, номер раздела, который вас интересует.

Обратите внимание, что вам предложены не ответы, а вопросы, а точнее: области, внутри которых вы можете найти ответы на интересующие вопросы.

Допустим, что вы интересуетесь наукой и в ответ на этот кадр набрали на клавиатуре номер 1. Тогда машина предлагает следующий кадр «меню»:

ЭВМ: *В разделе «Наука» имеются книги по следующим областям:*
1 — *физика*
2 — *химия*
3 — *вычислительная техника*
4 — *механика*
5 — *биология*
6 — *математика*
и т. д.
Наберите номер той области, которая вас интересует.

Пусть вы набрали номер 3, на что следует кадр:

ЭВМ: *В разделе «Вычислительная техника» имеются книги по следующим направлениям:*
1 — вычислительная математика
2 — программирование
3 — архитектура ЭВМ
4 — системотехника
5 — диалог «человек — ЭВМ».
Наберите номер направления, которое вас интересует.

Вас, разумеется, интересует диалог, и следовательно, кнопка номер 5. В ответ на это вы получите следующую страницу «меню» на экране дисплея:

ЭВМ: *В разделе Диалог «человек — ЭВМ» имеются следующие книги:*
1 — Глушков В. М. и др. Человек и вычислительная техника. Киев, 1971.
2 — Мартин Дж. Системный анализ передачи данных. Т. 1. М., 1975.
3 — Чачко А. Человек за пультом. М., 1974.
Наберите номер книги, которая вас интересует.

Нажав кнопку 3 вы получите на экране реферат книги А. Чачко (это необходимо, так как название книги редко правильно отражает ее содержание).

Ознакомившись с ним вы должны сделать очередной выбор, предлагаемый машиной:

ЭВМ: *Что желаете получить:*
1. *Содержание*
2. *Книгу с экрана*
3. *Собственно книгу*
Нажмите соответствующую кнопку.

Нажав первую кнопку, вы получите на экране содержание этой книги с повторением двух остальных вопросов. Вторая кнопка дает вам возможность читать книгу с экрана. Каждая ее страница будет сопровождаться вопросами:

ЭВМ: 1. *Следующую страницу?*
2. *Через одну страницу?*
и т. д.

Так вы можете «листать» и читать ее «по диагонали».

Нажав третью кнопку вы тем самым автоматически сделаете заказ в книгохранилище, а на экране получите указание когда и где в библиотеке вы можете получить книгу в ее натуральном виде.

Выключив экран, вы в любой момент можете выйти из системы.

Такой системы во всей ее мощи пока нет. Есть отдельные фрагменты. Но реализация ее вполне возможна на современном уровне развития вычислительной техники (основная трудность состоит во введении текста книг в память ЭВМ — пока это делается вручную). Автоматизация этого процесса сделает такую систему повседневной реальностью.

Как видно, способ меню не требует от пользователя никаких знаний о программировании, устройстве и специфике ЭВМ. Знай читай текст и нажимай нужную кнопку — благо, что их немного. Но таким способом удается решать крайне узкий класс задач. Чуть ваша задача отклонится от предусмотренных в программе и в меню не окажется нужного вам «блюда».

Что же делать? — Надо программировать — никуда тут не денешься! А уж если приходится программировать, то хотелось бы иметь в своем распоряжении очень простой язык. Таким языком является БЭЙСИК.

Язык для туземцев (бэйсик)

В прошлом веке один английский миссионер, желая приобщить туземцев к цивилизации, выделил из английского языка самую распространенную и самую простую его часть и стал учить туземцев такому упрощенному английскому языку. Этот язык содержал всего 300 слов и почти не имел грамматики. Назвали его «Basic English», то есть «основной английский» (в русской транскрипции: бэйсик инглиш). Язык привлекал своей простотой и вскоре завоевал популярность не только среди туземцев, но и эмигрантов.

В 1965 г. был разработан новый бэйсик, но уже для других «туземцев» — людей, не владеющих языками общения с ЭВМ. Назвали его тоже BASIC, что является аббревиатурой английской фразы: Beginner's All — purpose Symbolic Instruction Code, то есть многоцелевой язык символических инструкций для начинающих. Название несколько надуманное для того, чтобы получить сокращение бэйсик, известное всему миру.

Как и первый бэйсик, второй быстро завоевал популярность среди «туземцев» — пользователей ввиду своей простоты и доступности. Другой его важной чертой является то, что он позволяет решать задачу в режиме диалога с ЭВМ, что очень важно для начинающих.

Бэйсик (одна из его версий) содержит следующие символы:

1. 26 заглавных английских букв от A до Z.
2. 10 цифр от 0 до 9.
3. 4 знака препинания: . (точка), , (запятая), ; (точка с запятой), ' (апостроф).
4. 5 знаков операций: + (плюс), — (минус), × (умножение), / (деление), ↑ (возведение в степень).
5. 6 знаков отношений: $=$, $\neq$, $>$, $\geqslant$, $<$, $\leqslant$.
6. ( ) — круглые скобки.

Вот и все! В русском варианте добавлены заглавные русские буквы, отличающиеся от английских, например, Б, Ю и т. д.

Основные операторы бэйсика:

NEW (новый) — открывает новую программу,

RUN (прогон) — запускает программу,

PRINT (печатать) — команда на печать того, что следует за этим оператором.

Все операторы в программе нумеруются обычно через 10, кроме NEW и RUN

Пример простейшей программы:

```
  NEW
10 PRINT 'Я УЧУ БЭЙСИК'
  RUN
```

ЭВМ на эту программу выдаст:

Я УЧУ БЭЙСИК,

то есть напечатает то, что окаймлено апострофами ('),

Еще операторы бэйсика:

LET (пусть) — оператор присвоения. Он определяет значение переменных, следующих за ним.

IF... THEN... (Если..., то...) — оператор условного перехода. Отличается от фортрановского только тем, что отсылает не к метке, а к номеру оператора, который играет в бэйсике роль метки.

Другой пример — вычисление факториала $N!$, заданного числа N:

	Комментарий
NEW	
10 LET X = 1	Задается начальное значение переменной x.
20 LET I = 0	Задается начальное значение переменной i.
30 LET I = I + 1	Увеличение i на единицу.
40 LET X = X * I	Новое значение x равно старому, умноженному на i.

50 IF I<N THEN 30 Если $I < N$, то следует идти на оператор с номером 30 (при $I = N$ программа перейдет на следующий оператор с номером 60, в этот момент $X = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot N = N!$.

60 PRINT X Печать результата $N!$.

Как легко заметить, бэйсик очень похож на фортран, только значительно проще. Именно поэтому этим языком сейчас снабжены почти все ЭВМ, чаще всего используемые в диалоговом режиме.

Опыт показывает, что тот, кто хоть раз посидел за пультом дисплея и поработал с ЭВМ на бэйсике никогда не забывает захватывающего ощущения огромных возможностей, которые открывает ЭВМ перед пользователем: ее мгновенная реакция на любой фрагмент составленной программы (достаточно нажать клавишу «RUN», а если ее нет, то набрать слово RUN на клавиатуре дисплея) и неограниченные возможности по исправлению и совершенствованию программы. Так, если надо в старую программу вставить новый кусок, то достаточно после какого-то, например, 80-го, оператора вставить 85 GOTO 500 и направить вычисления в нужную сторону (в данном случае к оператору под номером 500) с тем, чтобы, аналогично, вернуться в нужном месте к старой программе при помощи того же GOTO например, так: 590 GOTO 90 (оператор GOTO в дословном переводе на русский означает «иди к...» оператору, номер которого указан далее). Работая по такой программе ЭВМ после 80-го оператора обратится к 85-му, выполнит далее кусок от 500 до 590-го оператора и возвратится к 90-му. (Именно для реализации таких возможностей и предусмотрен порядковый счет операторов в бэйсике через 10).

Можно смело сказать, что именно с бэйсика часто начинается влюбленность в ЭВМ и «туземец» превращается в маститого «аборигена», бесстрашно решающего сложнейшие задачи с помощью ЭВМ.

А для того чтобы решать более или менее сложные задачи на ЭВМ, нужно не только уметь программировать, но и знать, какими готовыми программами можно воспользоваться (они расположены обычно во внешней памяти ЭВМ, т.е. на магнитных дисках или лентах). Все это образует программное обеспечение.

Давайте разберемся в этом.

Что такое программное обеспечение ЭВМ?

Читатель уже заметил в описанных примерах, что привлекательность ЭВМ для пользователя определяется двумя ее фундаментальными свойствами: уметь работать по задаваемой извне программе и хранить в своей памяти информацию с тем, чтобы выдавать ее пользователю по мере необходимости. К этим двум свойствам надо добавить и третье: способность выполнять и другие программы, хранимые в памяти ЭВМ. Эти другие программы очень разнообразны. Они-то и образуют так называемое программное обеспечение современной ЭВМ. Иногда его называют математическим обеспечением, но к математике, как к науке, оно отношения почти не имеет. Программное обеспечение современной ЭВМ представляет собой набор программ, с помощью которых реализуются ее различные и многочисленные функции и режимы работы. Объем современного программного обеспечения хорошо иллюстрируется его стоимостью — она сейчас в десять раз превышает стоимость той машины, для которой разработано это программное обеспечение. Так что ЭВМ в ее «металлическом» исполнении представляет собой только возможности, которые реализуются с помощью и при непосредственном участии программного обеспечения. Без него ЭВМ, грубо говоря, лишь груда красивого металла.

Учитывая важность программного обеспечения, давайте рассмотрим его структуру. Оно подразделяется на два класса программ: прикладных и системных.

Прикладными программами называют те программы, ради которых и существует ЭВМ. Они решают конкретные задачи пользователей. Например, если у вас возникла необходимость решить систему линейных уравнений, а вы располагаете лишь исходными данными — таблицей коэффициентов этой системы уравнений, то составлять программу ее решения не нужно — она обычно входит в программное обеспечение и хранится где-то в памяти машины. Нужно найти ее и вызвать определенной командой. Теперь осталось лишь ввести исходные данные и запустить найденную программу.

Таких прикладных программ существует очень много и все они решают какую-то типовую задачу. Если вам нужно решить нетипичную задачу — придется самому составлять программу или идти на поклон к программисту.

Если прикладные программы нужны пользователю, то системные — машине. Именно они позволяют ей работать хорошо, т. е. эффективно обслуживать пользователя. Для этого ЭВМ нужно уметь многое. Именно это многое программируется и образует системное программное обеспечение ЭВМ. О системных программах будет рассказано чуть позже, а теперь вернемся к прикладным.

Пакеты прикладных программ

Прикладные программы объединяются в пакеты, которые так и называются «пакеты прикладных программ» (сокращенно ППП). Это объединение производится по принципу специализации, т. е. в один пакет объединяют прикладные программы, решающие задачи из одного и того же раздела науки, техники и т. д. Например, упомянутая выше программа решения систем линейных алгебраических уравнений будет извлечена из пакета «Алгебра», где она объединена вместе с программами вычисления определителя, обращения матрицы и т. д.

Пакет «Статистика» объединяет программы, необходимые для обработки результатов наблюдений. А в пакете «Анализ» собраны программы численного решения дифференциальных уравнений, вычисления определенных интегралов и другие вычислительные программы математического анализа. И т. д. Очевидно, что число таких пакетов определяется числом областей, требующих широкого применения ЭВМ (для редких случаев заранее составлять прикладную программу нецелесообразно).

Следует различать пакеты и библиотеки прикладных программ. Библиотека отличается тем, что устроена она так же, как книжная библиотека: каждая книга не зависима от других. Например, ее можно вынуть из библиотеки и это никак не повлияет на содержание других книг. Библиотека программ, таким образом, является механическим объединением программ по принципу их специализации. Однако такую организацию нельзя считать удачной для ЭВМ. Библиотека программ слишком избыточна, так как содержание ее программ всегда пересекается. Действительно, даже в обычной книжной библиотеке (особенно в научно-технической) содержание отдельных разделов книг очень часто повторяется (если не текстуально, то по смыслу). Специали-

сты утверждают, что оригинальная информация занимает всего процентов двадцать от объема любой научно-технической библиотеки. Точно так и с прикладными программами.

Так, например, программа решения системы линейных алгебраических уравнений используется (уже в качестве подпрограммы) другими прикладными программами, например, программой регрессионного анализа, которая позволяет экспериментальные данные представить в виде полинома. Именно таким образом реализуется так называемый модульный принцип для пакетов.

Этот принцип требует, чтобы пакет состоял из отдельных непересекающихся «кусочков» (модулей), из которых собираются прикладные программы пакета. Если теперь вынуть из него какой-то один модуль, то многие прикладные программы (а может быть и все) не смогут «собраться».

Процедуру сборки прикладной программы из имеющихся в ППП модулей производит так называемый монитор (точнее: программа монитора). Он, по заказу пользователя, собирает нужную ему программу. Таких программ монитор может собрать много. И если их сложить вместе, то они займут значительно больше места в памяти ЭВМ, чем исходные модули и монитор. Это обстоятельство и делает ППП более удобным для ЭВМ, чем библиотека прикладных программ.

Как видно, пакет прикладных программ является довольно сложной, но очень компактной программной системой.

Итак ППП экономит память ЭВМ за счет своего усложнения. Для создания ППП нужно владеть довольно тонкими приемами программирования. Если память ЭВМ велика, то лучше делать библиотеки, а не ППП. Так и поступают пользователи, когда создают собственные пакеты прикладных программ, решающие их специфические задачи.

Следует, однако, заметить, что грань, отделяющая библиотеку от пакета, определена не четко. Действительно, программы библиотеки могут использовать в качестве вызываемых подпрограмм стандартные программы ЭВМ. Эти вызываемые программы можно рассматривать, как программный модуль, что делает библиотеку похожей на пакет. Именно поэтому иногда пакет называют библиотекой, а библиотеку — пакетом.

Системные программы

Эти программы обеспечивают работу прикладных программ. Дело в том, что современная ЭВМ является очень сложной электронной машиной с большим числом узлов, которые сложно взаимодействуют в процессе решения каждой конкретной задачи пользователя.

Поясним это. Всякая ЭВМ состоит из 3-х основных частей: процессора, выполняющего функцию вычислений ЭВМ, памяти и аппаратуры ввода и вывода информации. Если ЭВМ решает одну задачу, то обеспечение координации работы всех ее устройств возлагается на управляющее устройство (оно входит в состав процессора), которое в соответствии с программой (она хранится в памяти) загружает последовательно все узлы машины. В этом случае никаких особых системных программ не нужно. Действительно, программа составлена так, что в каждый момент должна выполняться какая-то одна конкретная операция. Машина устроена так, что эту операцию выполняет одно из ее устройств (процессор, печатающее устройство и т. д.). Координировать, по сути говоря, нечего, так как эти устройства не могут войти в конфликт при последовательном выполнении программы, когда работает лишь одно устройство, а остальные ждут, т. е. простаивают.

Совсем другая ситуация складывается при желании убыстрить работу ЭВМ. Это можно сделать по разному. Во-первых, можно запараллелить выполнение одной программы, то есть одновременно выполнять ее разные куски (если это возможно). Например, во время печати промежуточных результатов продолжать выполнение другой части программы, если эта часть не требует печати. За этим «если» должна следить системная программа и устранять конфликт, координируя работу процессора и устройства вывода (в данном случае — печать). А, во-вторых, можно на одной ЭВМ одновременно решать сразу несколько задач.

ЭВМ в роли Цезаря

Очевидно, режим работы ЭВМ должен быть таким, чтобы быстрее решать задачи. А от чего зависит скорость решения? Прежде всего от скорости выполнения операторов программы ЭВМ, от скорости ввода данных и вывода результатов. Если скорость работы машины

невелика, то рабочие операции в основном и «съедают» все время. Так было в недавнем прошлом. Однако, как всем известно, скорость счета современных ЭВМ очень быстро растет (так, за каждые 10 лет развития вычислительной техники эта скорость увеличивается более чем в 10 раз).

Ввод и вывод информации осуществляется значительно медленнее, причем (и это главное) в это время процессор простаивает, что приводит к противоречию между операциями вычислений и ввода/вывода. Из-за этого получалось так: несколько минут вводились программа и исходные данные, затем за 2—3 секунды решалась задача, далее выводились результаты, на что также затрачивалось много времени. Складывалось нелепое положение, когда самая дорогая и быстродействующая часть ЭВМ — процессор — долго простаивал в ожидании выполнения вспомогательных операций ввода/вывода.

Напрашивалось решение — обеспечить одновременную работу всех устройств, но над разными задачами. Когда N-я задача вычисляется, $(N+1)$-я вводится, а $(N-1)$-я (уже решенная) — выводится. Так появился пакетный режим работы ЭВМ, суть которого заключается в следующем. Из подготовленных программ и данных составляется так называемый «пакет задач», которые поступают непрерывно в ЭВМ. Машина решает задачи одну за другой без перерыва, а результаты выводятся по мере их получения.

Реализация такого режима потребовала новых усовершенствований программного обеспечения. Так, нужно было добиться, чтобы машина могла одновременно считать, вводить и выводить информацию «бесконфликтно». Для этого понадобились системные программы.

Говорят, что известный Гай Юлий Цезарь мог одновременно слушать, писать и говорить. Такое умение совмещать различные «режимы» работы произвело настолько большое впечатление на современников, что до многих из нас, их потомков, дошли сведения только об этом его достоинстве (если не верите, спросите у знакомых, чем был знаменит Цезарь). Современная ЭВМ, работающая в пакетном режиме, подобна Цезарю: она может одновременно выводить результат решения одной (уже решенной) задачи, считать другую и вводить программу третьей, которую ей предстоит решать.

Если продолжить аналогию с Цезарем, то можно сказать, что пакетный режим позволяет ЭВМ одновременно «слушать» (вводить программу и данные следующей задачи), «думать» (решать задачу, программа и данные которой уже введены в ее память) и «говорить» (т. е. выводить результаты решения только что просчитанной задачи).

Выигрыш налицо! И обеспечивают этот выигрыш системные программы, координирующие работу всех устройств ЭВМ.

Самой сложной и самой важной системной программой является следующая.

Операционная система

Как сказано, системные программы, а, следовательно, и операционная система (сокращенно ОС) возникли для расширения возможностей вычислительных машин (в основном для увеличения их производительности). Потребность в повышении производительности ЭВМ связана не только с экономическими факторами. Дело в том, что закупка какой-то организацией ЭВМ вызывает лавинообразный процесс интереса к ней и, в конечном счете, оказывается, что определенная категория работников этой организации без нее просто не может существовать. Совершенно непредвиденно появляются потребности, о которых раньше нельзя было даже и подозревать. Такого рода перегрузку ЭВМ нельзя планировать, т. к. она является свидетельством каких-то сложных процессов в сознании людей, заставляющих их обращаться к услугам ЭВМ. Поэтому так остро встал вопрос о расширении возможностей и повышении производительности ЭВМ. Именно этому служит операционная система. Перечислим ее основные функции.

Одной из основных функций операционной системы является обеспечение полного использования ресурсов ЭВМ, т. е. процессора (арифметического и управляющего устройств), памяти и устройств ввода-вывода. Основным ресурсом обычно является процессорное время, которое и определяет быстродействие ЭВМ.

Распределение ресурсов ЭВМ операционной системой производится в соответствии с потребностями пользователей (точнее: их программ) и при учете требований эффективности и надежности работы ЭВМ.

Задачу, которая стоит перед операционной системой можно образно представить, как задачу хозяина, к которому непрерывно приходят гости (это программы пользователей). Каждый из гостей нуждается в определенной доле удовольствий (коктейль, музыка, танцы, журналы и т. д.), причем в определенной индивидуальной последовательности. Однако каждое из обслуживающих мест (бар, электрофон, танцплощадка, библиотека и т. д.) имеют естественно ограниченный ресурс. Хозяину необходимо так спланировать досуг каждого гостя, чтобы тот получил свою долю развлечений и побыстрее покинул его дом. Не будем осуждать хозяина за последнее желание — просто он хочет, чтобы другим гостям, которые пришли позже, тоже было хорошо.

Если гостей мало, то у хозяина проблем нет. Он может смело пустить все на самотек: каждый гость сам будет развлекаться, так как народу мало и поэтому всего (коктейлей, музыки, танцев, журналов и т. д.) можно легко добиться, не рискуя конфликтовать с кем-либо из-за какого-то ресурса ввиду его ограниченности.

Но совсем иная ситуация складывается при большом стечении гостей — в баре не хватает рюмок (хозяйка не успевает мыть), на танцплощадке толчея, у электрофона каждый хочет послушать свой шлягер, а в библиотеке нет свободных мест. Хозяину приходится изрядно попотеть, чтобы соблюсти приличия: надо поддерживать очереди (они неизбежно появятся перед каждым местом обслуживания). Для убыстрения обслуживания одних гостей надо попридержать других. Очевидным приоритетом пользуются у хозяина гости-торопыги — их нужно быстро обслужить и ... проводить. Медлительным гостям можно подождать — ведь они все равно будут «засиживаться» в гостях.

Пусть читатель не думает, что автор не любит гостей. Просто при их большом числе следует об их развлечении специально позаботиться так, как это делает операционная система с пользовательскими программами.

Основными функциями операционной системы являются планирование работ по выполнению пользовательских программ и выполнение этого плана с учетом сложившейся ситуации. Первую функцию выполняет планировщик (точнее: его программа), а вторую — супервизор

(тоже его программа). Заметим, что последние замечания в скобках можно было бы не приводить, так как ЭВМ является программируемым автоматом и все ее возможности связаны с программами, которые выполняет ЭВМ. Такой программой, расширяющей возможности ЭВМ по обработке пользовательских программ, и является программа операционной системы, основой которой являются программы планировщика и супервизора.

Начнем с планировщика. Он, располагая данными о программах, поступивших в ЭВМ от пользователей, составляет такой план распределения ресурсов ЭВМ, чтобы быстрее решить задачи пользователей. При этом планировщик руководствуется простым и надежным правилом — максимально загрузить все ресурсы ЭВМ и прежде всего ее процессор. Действительно, простой какого-либо ресурса ЭВМ всегда снижает ее быстродействие, а следовательно задерживает решение задач пользователей.

Но план, составленный планировщиком, не может быть совершенным хотя бы потому, что планировщик располагает крайне скудной информацией о потребностях пользовательских программ (исключая память). Действительно, как узнать планировщику, сколько процессорного времени займет какая-то программа, если она имеет условные переходы (помните оператор IF... THEN?), которые реализуются при выполнении определенных условий, момент наступления которых не определен? Так что планировщику приходится планировать очень приближенно.

Составленный планировщиком план осуществляет супервизор, что в дословном переводе с английского означает «сверхнаблюдатель» — название не очень удачное. Его бы назвать суперюзером, т. е. «сверхисполнителем», так как он является прежде всего исполнителем составленного планировщиком плана и кроме того его корректировщиком на основе информации, получаемой в процессе реализации плана. А информации супервизору поступает очень много!

За это, по-видимому, его и назвали сверхнаблюдателем. Эта информация о действительной загрузке ресурсов ЭВМ (процессора, памяти и устройств ввода-вывода) дает супервизору большие возможности по обеспечению загрузки этих ресурсов. Так, например, видя, что процессор не загружен, супервизор может вопреки пла-

ну загрузить его новой программой, очередь которой по плану еще не подошла.

Взаимоотношения между планировщиком и супервизором можно сравнить с взаимоотношениями между Госпланом и министерством. Госплан планирует, а министерство реализует этот план с учетом сложившейся ситуации, которая иногда заставляет министерство действовать вопреки плану, но обеспечивая при этом максимальную эффективность своей отрасли.

Монте-Карло и ЭВМ

А теперь давайте рассмотрим методы решения задач на ЭВМ. Эти методы обычно заимствуются из вычислительной математики — раздела математики, рассматривающего вычислительные методы. Вычислительная математика возникла едва ли не раньше всех других математик. Действительно, правила счета на пальцах являются первыми методами решения «математических» задач того времени.

За всю свою многовековую историю вычислительная математика накопила большое количество эффективных вычислительных приемов. Формализация этих приемов привела к тому, что сейчас называют алгоритмами. Стоит ли говорить, что эти алгоритмы были разработаны для «ручного» счета, т. е. для человека вооруженного карандашом и бумагой.

Когда появилась ЭВМ, то, естественно, что ее заставили работать по этим «ручным» алгоритмам. Это было не лучшее решение, но другого выхода не было — машинных алгоритмов решения задач еще не существовало.

Но вот, в 1949 г. появился первый метод, ориентированный на вычислительные машины. Это был метод Монте-Карло. Вручную считать этим методом бессмысленно. Это метод быстродействующих вычислительных машин! Состоит он в генерировании с помощью ЭВМ случайных процессов с различными свойствами. Эти процессы используются для решения конкретных вычислительных задач.

Например, для определения площади сложной фигуры методом Монте-Карло (или, как его иногда называют, методом статистических испытаний) достаточно генерировать парами случайные числа x и y, которые

определяют точку A внутри описывающего эту фигуру прямоугольника (см. рис. 2). Некоторые из этих точек будут попадать на интересующую нас фигуру (такие как A_1), а остальные — нет (A_2). С помощью ЭВМ определить это очень просто. Если теперь вычислить относительное число точек, попавших на фигуру, то ее площадь примерно составит именно эту долю от площади описывающего прямоугольника, то есть будет равна:

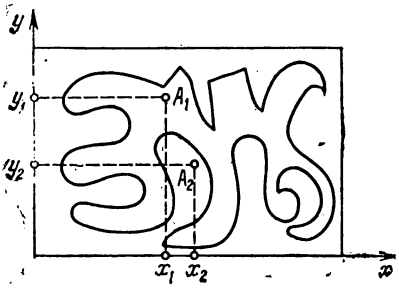

Рис. 2. Вычислить площадь такой фигуры — довольно сложное и кропотливое занятие, если это делать «старым» способом. Метод Монте-Карло решает его значительно проще. Надо «бросать» на прямоугольник случайные точки и считать, сколько из них попадет на фигуру. Доля этих точек из всех брошенных и будет долей площади прямоугольника, занимаемой интересующей нас фигурой.

$$S_{\text{ф}} \approx S_{\text{п}} \frac{N_1}{N_1 + N_2},$$

где $S_{\text{ф}}$ — площадь фигуры, $S_{\text{п}}$ — площадь прямоугольника, N_1 — число случайных точек, попавших на фигуру (типа A_1), N_2 — число случайных точек вне ее (типа A_2). Самым замечательным свойством этой формулы является то, что при $N = N_1 + N_2 \to \infty$ она становится абсолютно точной. А так как точные значения нигде и никому не нужны, то ею можно пользоваться при конечных значениях N (обычно достаточно больших, для чего и используется ЭВМ).

Нетрудно заметить, что это свойство получено за счет того, что числа x и y случайны, их достаточно много и они равномерно распределены по сторонам прямоугольника.

Как видно, интригующее название метода Монте-Карло никакого отношения к знаменитому казино не имеет. Разве что игровая рулетка может быть использована как датчик равновероятных случайных чисел. Но это слишком медленный датчик. ЭВМ вырабатывает случайные числа практически мгновенно.

Метод Монте-Карло лежит в основе одного популярного сейчас метода — метода имитационного моделирования. Рассмотрим его.

Как нынче грабят банки? (имитационное моделирование)

Грабят нынешние гангстеры по-старому, т. е. в масках и грозя пистолетами. А вот подготавливают ограбление по-новому — с помощью ЭВМ. Дело в том, что процесс ограбления (как и всякий другой реальный процесс) зависит от состояния множества внешних факторов, точное определение которых невозможно. Об каждом из таких факторов можно высказать лишь приближенное суждение в виде интервалов, в котором находится значение этого фактора. Так, например, сколько человек будет в операционном зале, сколько денег и золота окажется в сейфе, какое время понадобится для вскрытия сейфа и т. д. Относительно каждого из этих факторов ничего точного заранее сказать нельзя. Но кроме точного описания существует еще стохастическое описание, которое дает возможность представить характер поведения интересующего нас параметра. Так, например, при слове «очередь», естественно знать, какая она. Точное ее значение указать заранее нельзя, а стохастическое описание ее дать легко, например, в виде 5 ± 2 или 30 ± 10. В первом случае очередь может быть в пределах от 3 до 7 человек, а во втором — от 20 до 40. Получить такую информацию очень важно — каждый, стоявший в очереди, это знает.

Но вернемся к планированию ограбления банка (автор здесь рассматривает не шутливую задачу, а вполне реальную, которую в действительности решает с помощью ЭВМ не только современная мафия, но и служба охраны банков). Всякий план нуждается в проверке на возможный провал. Этот провал может произойти по разным причинам, а причины зависят от состояния факторов, точное значение которых определить нельзя. Одним из важнейших показателей является длительность операции до того, как она становится явной для окружающих. С этого момента следует ожидать и вызова полиции и столкновения с возможным полицейским агентом в банке и т. д. Именно поэтому из всех вариантов плана ограбления следует выбрать тот, который занимает минимальное время. Но как это время определить, если оно зависит от большого числа факторов, значение которых точно не определить?

Для решения задач такого типа и был разработан метод, который называют методом имитационного моделирования.

Идея этого метода заключается в пошаговом моделировании поведения исследуемого процесса. Дело в том, что о поведении любого процесса на коротком интервале времени мы обычно знаем больше, чем о поведении в целом. Это обстоятельство и является основным при использовании методов моделирования на ЭВМ.

А что делать с неопределенными факторами? Тоже моделировать, но статистически, то есть используя метод Монте-Карло. Так, например, если известно, что фактор x находится в заданных пределах: $a \leqslant x \leqslant b$, то выбирается случайное число ξ в этих же пределах: $a \leqslant \xi \leqslant b$, а фактору x приписывается это случайное значение $x = \xi$ в данной реализации при моделировании процесса. В другой реализации это будет другое случайное значение в тех же пределах.

Поэтому каждая реализация моделируемого процесса будет отличаться от другой именно значениями неопределенных факторов. Как видно, результат моделирования каждый раз будет различный и при этом случайный, так как в его образовании участвовали случайные числа ξ. Как же воспользоваться этим случайным результатом? Да так же, как и при наблюдениях за реальным случайным процессом! Если есть несколько наблюдений какого-то реального процесса, например, времени проезда в городском транспорте по определенному маршруту, то уже многое можно сказать о свойствах этого процесса — его среднее значение, максимальные и минимальные затраты времени и т. д. Это уже не мало.

Вот так, располагая весьма скудными сведениями о локальном поведении интересующего нас процесса, можно изучить его свойства в целом. Для этого необходимо иметь достаточное число реализаций процесса. Именно здесь и нужна ЭВМ, так как с ее помощью можно быстро «разыгрывать» различные реализации такого процесса. А так как наибольший интерес представляют собой сложные процессы с большим числом факторов, то быстродействие ЭВМ является решающим для эффективности такого подхода. Без ЭВМ метод имитационного моделирования не имеет смысла — огромные затраты на «ручное» моделирование никогда не покроют полученного эффекта, даже немалого.

Извольте бриться ... (язык моделирования GPSS)

Системы массового обслуживания являются, пожалуй, самыми многострадальными (в смысле приносящими страдания) системами, с которыми нам приходится иметь дело повседневно. А «страдания» они приносят обслуживаемому клиенту, когда ему приходится ждать очереди на это самое обслуживание. Именно поэтому так важно знать характеристики и свойства проектируемой системы массового обслуживания или как эти свойства изменяются при введении определенных усовершенствований в действующую систему.

Существующая теория массового обслуживания, к сожалению, дает ответы только в простейших случаях, чаще всего не интересных для практики. Жизнь заставляет создавать значительно более сложные системы массового обслуживания, свойства которых необходимо уметь предвидеть, т. е. отвечать на вопросы о длине очередей в будущей системе, среднем времени ожидания обслуживания и т. д.

Именно для этого применяется метод имитационного моделирования. Только он позволяет получить ответы на вопросы, задаваемые разработчиками систем массового обслуживания.

Приведем пример такой простейшей задачи. Имеется один обслуживающий прибор (это могут быть парикмахер, кассир, продавец и т. д.) и поток клиентов на обслуживание (желающих побриться, получить деньги, купить и т. д.). Свойства прибора и потока заданы. Прибор обслуживает клиента за время в интервале $T \pm \Delta$, где T — среднее время обслуживания, $T - \Delta$ — минимальное время обслуживания, а $T + \Delta$ — максимальное (величины T и Δ заданы). Поток клиентов так же описывается интервалом времени между клиентами: $\tau \pm \delta$, где τ — среднее время между появлением клиентов, а $[\tau - \delta, \tau + \delta]$ — интервал возможного изменения этого времени (значения τ и δ должны быть заданы). Вот и вся исходная информация. А вопросов много: какая будет очередь клиентов, ее среднее значение и максимальное (минимальное, очевидно, равно нулю), какой процент времени обслуживающий прибор будет простаивать? И т. д.

Ответы на эти и многие другие вопросы дает моделирование. Процесс моделирования обладает своей спецификой, которая отличает его, и очень существенно,

например, от вычислительных процессов. Конечно, и процесс моделирования можно было бы свести к формульным вычислениям, но такое представление было бы не свойственно процессу моделирования и поэтому было бы очень громоздко. Именно поэтому для моделирования разработаны специальные языки высокого уровня. Среди них наибольшей популярностью пользуется язык GPSS (по-русски его называют «джи — пи — эс — эс»), название которого является аббревиатурой слов General Purpose Simulating System, т. е. общецелевая моделирующая система.

Всякий процесс моделирования связан с воспроизведением поведения моделируемого объекта. А это поведение характеризуется моментами его изменения. Поэтому надо моделировать те моменты времени, когда в объекте что-то изменяется (приход клиента, начало его обслуживания, конец обслуживания и т. д.).

Операторы языка GPSS описывают именно эти моменты. Рассмотрим основные из них.

Оператор GENERATE (генерировать) определяет моменты появления клиентов в системе обслуживания. Он характеризуется пятью числами A, B, C, D, E:

GENERATE A, B, C, D, E.

Это означает, что клиенты поступают в среднем через A единиц времени в интервале $A \pm B$, причем первый клиент придет в случайный момент в интервале $C \pm B$; число таких клиентов равно D, а E — это приоритет такого рода клиентов.

Например,

GENERATE 5

обозначает, что клиенты поступают строго через 5 единиц времени, то есть в моменты 5, 10, 15, ..., причем отсутствие (умалчивание) остальных данных означает, что $B = 0$; $C = A$; $D = \infty$; $E = 0$. Как видно, таким способом можно задавать весьма разнообразные потоки клиентов.

Оператор QUEUE A (стать в очередь) присоединяет клиента к очереди A, где A — имя очереди (название или номер). Таких очередей может быть много. Вообще говоря, не для всякого клиента потребуется этот оператор. Так, нетерпеливые клиенты не становятся в очередь, если она велика, т. е. больше какой-то определенной величины.

Оператор DEPART B (покинуть очередь) выводит клиента из очереди B при обращении к этому оператору.

Оператор SEIZE *C* (занять) реализует переход клиента на обслуживающий прибор *C* (это название прибора или его номер), если он свободен.

Оператор RELEASE *D* (освободить) освобождает обслуживающий прибор *D*.

Оператор ADVANCE *E, D* (задержать) реализует задержку клиента во время его обслуживания на рабочем месте, указанном выше, и характеризуется средним временем обслуживания *E* и полуинтервалом *G*, таким, что время обслуживания является случайной величиной в интервале $E \pm G$

Например,

ADVANCE 12, 8

означает, что прибор обслуживает клиента случайное время в интервале 12 ± 8, то есть от 4 до 20 минут.

Оператор TERMINATE (завершить) удаляет клиентов из модели. При отсутствии этого оператора они могли бы встать в другую очередь к другому прибору.

Оператор SIMULATE (моделировать) отдает команду на моделирование ЭВМ процесса, программа которого написана ниже. При отсутствии этого оператора ЭВМ лишь просмотрит программу на предмет выявления ошибок, но работать по ней не будет.

А теперь напишем простейшую программу на GPSS, моделирующую работу парикмахерской с одним рабочим местом. Пусть 25 клиентов с одинаковым приоритетом поступают через случайные промежутки времени в интервале 18 ± 6 мин, то есть от 12 до 24 мин, а парикмахер обслуживает клиента случайное время в интервале 16 ± 4. Интерес к моделированию процесса обслуживания вполне реальный — например, сколько ставить стульев для ожидающих клиентов? А число стульев должно быть равно максимальной длине очереди. Именно эту величину следует прежде всего определить в результате моделирования.

Итак, программа моделирования процесса обслуживания в такой маленькой парикмахерской имеет вид:

	Комментарий
SIMULATE	То есть надо моделировать программу, а не просто просмотреть.
GENERATE 18, 6, 10, 25	Приход 25 клиентов через случайные интервалы 18 ± 6 мин с одинаковым (нулевым) приоритетом, причем первый клиент приходит в интервале 10 ± 6 мин.

QUEUE1	Присоединение клиента к очереди номер 1 (другой здесь не будет).
SEIZE *A*	Переход в кресло парикмахера по имени *A* (другого здесь нет).
DEPART1	Уход из очереди 1. То, что этот оператор стоит позже SEIZE не ошибка, так как кресло могло быть занято, и уход из очереди логично фиксировать после занятия кресла. Этим модель отличается от жизни.
ADVANCE16,4	Обслуживание клиента у парикмахера занимает случайное время в интервале 16 ± 4 мин.
RELEASE *A*	Освобождение парикмахера *A*.
TERMINATE	Уход клиента из парикмахерской.

Вот и все. ЭВМ, проделав процедуру моделирования, выдает его результаты в виде характеристик этой системы массового обслуживания. Какие именно характеристики и как их вычислять, указывать не надо. Все это заранее заложено в язык GPSS (он специально ориентирован на такие задачи).

Так, прогон указанной выше программы дает следующее:

1. Общее время работы парикмахерской 472 мин, то есть 7 часов 52 мин.

2. Парикмахер был загружен 86% своего рабочего времени. Остальные 14% он ждал клиентов и, следовательно, мог делать какую-то другую работу (уборку помещения и т. д.).

3. Среднее время обслуживания каждого из 25-ти клиентов было 15,88 мин. (а по плану 16 мин.).

4. Очередь не превышала одного человека. Следовательно, если судить по модели одного дня работы парикмахерской, достаточно поставить один стул.

5. Средняя длина очереди 0,16 клиента.

6. Среди 25-ти клиентов 12 не ожидали в очереди, а прошли сразу на свободное кресло.

7. Среднее время ожидания в очереди одного клиента, т. е. включая и тех, кто не ожидал, равно 2,85 мин.

8. Среднее время ожидания в очереди одного ожидающего клиента (их было 13 человек) равно 5,13 мин. И т. д.

Эту программу не трудно было бы расширить на определенное время работы парикмахерской (например, 480 мин = 8 час) и тогда определилось бы число обслу-

женных клиентов (их будет 26 человек). Можно учесть и приоритеты клиентов, т. е. моделировать известное правило, что инвалиды Отечественной войны обслуживаются вне очереди, и т. д.

Легко представить, как применить этот язык при планировании очередного ограбления банка. Программа, аналогичная описанной, могла бы служить источником таких важных сведений, как среднее число клиентов в банке, максимальное и минимальное число клиентов и т. д. Значения же параметров A, B, C, ..., необходимые для работы программы, следовало бы определить из наблюдений за работой банка.

Отметим, что описание подобного криминального использования ЭВМ является не более, чем авторским приемом, опирающимся на интерес читателей к детективам. Не гангстерские потребности породили имитационное моделирование. Оно появилось в ответ на очень насущные потребности проектировщиков систем массового обслуживания. Им очень важно знать, какие ситуации будут складываться в проектируемой системе. Имитационное моделирование вообще и язык GPSS предоставляют им эту возможность.

Многоэтажная память ЭВМ

Мы уже говорили, что способность хранить информацию в своей памяти является одной из примечательных черт вычислительных машин. Эта память устроена довольно сложно — она многоэтажна, т. е. как и всякая память, имеет многоуровневый характер. Самая быстрая память невелика, но зато позволяет «вспоминать» с большой скоростью. Ее называют оперативной памятью. Там хранится программа, по которой работает машина. Процессор на каждом шаге обращается к оперативной памяти с тем, чтобы узнать, что же делать на следующем шаге (сам он почти ничего не знает). Скорость обращения к памяти и определяет поэтому быстродействие машины. Действительно, скорость обработки информации не может быть быстрее скорости «чтения» программы процессором. А считывает он ее из оперативной памяти со скоростью примерно миллион символов (чисел, букв, значков) в секунду. Объем же оперативной памяти приблизительно равен миллиону таких символов. Так что всю оперативную

память процессор может «просмотреть» за одну секунду, т. е. очень быстро.

Много ли это — миллион знаков? Не очень. Есть такие программы, которые не «влезают» в оперативную память ЭВМ и их заводят туда по частям по мере необходимости.

Именно поэтому современная ЭВМ снабжена более емкой, но и не такой быстрой памятью. Ее называют внешней памятью. Самой быстрой из внешних памятей ЭВМ является память на магнитных дисках. Ее объем во много раз превышает оперативную память (в 10—30 раз). Эти диски размером в большую грампластинку собраны на одной оси на некотором расстоянии друг от друга, по несколько дисков в пакете (обычно — по шести). Их можно быстро снять и поставить в дисковод, который вращает их с большой скоростью (2400 об/мин). Считывающие магнитные головки (их обычно 10 штук — внешние поверхности пакета не используются) расположены гребенкой и вводятся в пакет гидравлическим устройством, имеющим 203 фиксированных положения, соответствующих дорожкам на дисках. По команде головки быстро могут быть подведены к любой из 203-х дорожек (три из них запасные). Время движения головок от первой до последней дорожки достаточно мало — около одной десятой секунды. Однако для того, чтобы просмотреть все дорожки, времени потребуется значительно больше, так как придется фиксировать головки на каждой дорожке.

Но самой большой (и, соответственно, самой медленной) является память на магнитных лентах. Ее объем раз в десять превышает память на магнитных дисках (и, следовательно, по крайней мере раз в сто больше объема оперативной памяти ЭВМ).

Конструктивно память на магнитных лентах оформляется в виде магнитофона, способного с большой скоростью считывать информацию с ленты.

Как видно, три уровня памяти ЭВМ (оперативная, на дисках и на лентах) позволяет быстро манипулировать очень большой информацией. Нужная для работы ЭВМ информация по команде «сбрасывается» в ее оперативную память с диска или ленты.

Программы пользователя, его таблицы и тексты, т. е. все его массивы информации (их называют *данными*), удобно хранить на дисках или лентах (а часто для надежности и там и там). В программе пользова-

тель указывает номер диска (или ленты), где хранятся его данные. Оператор машины перед выходом пользователя на ЭВМ обязан поставить требуемый им носитель памяти в дисковод (или магнитофон).

Такое трехэтажное устройство памяти ЭВМ дает ей возможность запоминать теоретически неограниченный объем информации, что очень важно при использовании ЭВМ в качестве банка данных (чуть позже о нем будет рассказано подробнее). Но мало много запоминать — надо уметь хорошо воспользоваться этими знаниями. Кто-то умно сказал, что предпочитает хорошо наполненному мозгу мозг хорошо устроенный. Таким хорошим «устройством» является гибкость памяти, ее приспособляемость к новым условиям. Здесь особую роль играет адаптация...

Адаптация ЭВМ

Адаптация как способ приспособления к новым условиям существования является, пожалуй, наиболее интересным свойством живых существ. Именно благодаря адаптации появились и развились удивительно приспособленные биологические формы, что даже породило легенду об их изначальной гармонии в природе. Но все дело в адаптации!

На каждом шагу мы сталкиваемся с адаптацией. В человеческом обществе она приняла прямо-таки изощренные формы. Так обучение является одной из форм адаптации. Действительно, обученный человек более приспособлен, чем необученный. Осваивая языки программирования мы тем самым адаптируемся к ЭВМ, которая реализует программы, написанные на этих языках. Знание языка облегчает нам «сосуществование» с ЭВМ и делает нас тем самым более приспособленными.

Но это все адаптация человека к ЭВМ. А нельзя ли адаптировать ЭВМ к человеку? Такая адаптация машины к условиям ее «существования» значительно облегчила бы жизнь человеку, ведь критерии адаптации машине задает человек. Естественно, что он должен позаботиться о таком критерии, чтобы адаптированная по этому критерию ЭВМ была бы лучше для человека, чем неадаптированная.

Приведем пример такой адаптации. Пусть мы часто обращаемся к ЭВМ за какой-то информацией, хранимой

в ее памяти, т. е. используем ЭВМ в качестве банка данных. Эти данные хранятся в ее долговременной памяти, т. е. на магнитных лентах. Известно, что поиск информации на ленте требует, вообще говоря, ее полного просмотра, так как искомая информация может оказаться в конце ленты. На это понадобится несколько минут ожидания. Столько придется ждать, если запрашивать редко используемую информацию. И изменить здесь что-либо трудно, если поиск ведется в больших массивах, которые могут разместиться только на магнитной ленте.

Но информация информации — рознь. Одна требуется редко (или вообще никогда — часто случается и такое: известно, что в ленинской библиотеке 98% фонда никогда не запрашивалось), а другая информация нужна часто. Очевидно, что именно эту информацию следует расположить на носителях памяти с быстрым считыванием, т. е. на магнитных дисках. Объем такой информации, если судить по опыту ленинской библиотеки, составляет лишь 2% от общего объема. Поэтому на пять магнитных лент ЕС ЭВМ достаточно выделить лишь один комплект магнитных дисков (напомним, что в комплект входит шесть дисков с десятью рабочими сторонами и на ленту можно записать примерно содержимое десяти комплектов дисков).

Но какую именно информацию надо записать на дисках? Решить заранее это трудно, а иногда попросту невозможно. Действительно, создавая банк данных, мы почти всегда представляем, как он будет использоваться на данный момент. Но банк существует не один год и предсказать, что именно чаще понадобится через пять — десять лет, практически невозможно. Как же быть?

Именно здесь необходима адаптация. Адаптивный банк данных имеет специальную программу адаптации. Работает она просто. На чистый диск записывается каждый затребованный пользователями блок информации, если, разумеется, его не было на диске раньше. При каждом новом обращении к банку ЭВМ сначала выяснит, нет ли искомой информации на диске (это легко и быстро проверяется). Если есть, то пользователь сразу получает то, что ему нужно. В противном случае ему нужно подождать несколько минут, когда закончится поиск на ленте. При этом найденная ин-

формация одновременно с выдачей ее пользователю записывается на свободное место в диске.

Легко себе представить, что довольно быстро диск будет исчерпан, т. е. полностью «исписан». Вот здесь включается программа адаптации. Очевидно, что перед тем, как записать очередную порцию информации на диск, следует освободить место для нее, т. е. что-то надо «стереть» (Здесь, как и в обычном магнитофоне, специально стирать не надо — можно прямо наложить новую запись на старую).

Программа адаптации решает, что можно стереть с диска. Логика здесь простая и стратегий много. Например, можно стирать запись, к которой последний раз обращались давно, то есть запись с самым старым последним обращением к ней.

Для этого все время следует вести список дат последних обращений к записям на диске. ЭВМ отыскивает самую старую дату и стирает запись с этой датой. Этим будет поддерживаться режим самых «свежих» записей, т. е. банк будет быстро реагировать на те запросы, интервал между которыми не велик.

Такой режим адаптации хорош в системах оперативного управления, где запросы обладают достаточной стабильностью — частой или редкой. Частые запросы при этом будут удовлетворяться быстро, так как ответы на них расположатся на магнитном диске.

Однако такой способ адаптации едва ли одобрят пользователи, которые обращаются к банку данных редко, но «основательно», то есть задают много вопросов. Им каждый раз придется долго ждать ответа — ведь они выходят на машину редко.

Для такого рода пользователей лучше алгоритм адаптации, который учитывает не только давность последнего обращения, но и число обращений. Чем больше было обращений (хотя и старых) к какой-то записи в банке, тем дольше ее нужно хранить!

В любом случае, т. е. при любой дисциплине стирания записей на дисках, ЭВМ будет адаптироваться к потребностям пользователей и будет быстро отвечать на те запросы, которые более часто поступают, чем на редкие. В этом и состоит адаптация банка данных к пользователям. Банк как бы приспосабливается к их потребностям так, чтобы в среднем уменьшить время ожидания. При изменении потребностей перестроится

и «начинка» магнитных дисков. Во время такой перестройки ждать ответа, разумеется, придется дольше.

Здесь описан простейший вид адаптации, но он приводит к ощутимым результатам для большого банка данных и при большом числе «клиентов» этого банка.

Рассмотрим другой способ адаптации со странным названием...

«Двурукий бандит»

Такое претенциозное название имеет весьма мирный алгоритм адаптации, позволяющий выбирать лучшую из двух альтернатив в обстановке, осложненной действием посторонних факторов.

Простейшим примером такого двуальтернативного выбора может служить выбор транспорта, например, трамвая или троллейбуса в часы пик, когда вы добираетесь на свою работу. Действительно, троллейбусом быстрее, но ввиду малого числа посадочных мест есть риск не попасть в первый подошедший. Трамвай всегда заберет всех — мест достаточно, но едет медленнее. Здесь трамвай является одной альтернативой, а троллейбус — другой. Очевидно, что одним из этих видов транспорта добираться в среднем лучше, т. е. быстрее. Но каким? Решить это двумя экспериментами, съездив один раз на трамвае, а другой — на троллейбусе, нельзя, так как результат каждого эксперимента очень «зашумлен» всякого рода посторонними факторами (несинхронность движения транспорта, возможные «пробки» на улице и т. д.). Надо построить процедуру, которая выделила бы лучшую альтернативу при минимальном числе опозданий на работу. Это и будет алгоритм двуальтернативной адаптации (или «двурукий бандит»).

Почему бандит? («Двурукий» — ясно почему — это две альтернативы). Это криминальное название придумали американцы — большие любители броских названий. Имеются два объяснения.

Первое (игровое) связано с игральным автоматом, очень распространенным на Западе. Этот автомат имеет рукоятку, дернув за которую (но предварительно бросив в щель монетки) можно выиграть что-то, а можно и ничего не выиграть. Решение о выигрыше и его величине принимается случайным механизмом, который и запускается рукояткой.

Американцы назвали эти автоматы «одноруким бандитом» (естественно, что случайный механизм этого автомата настроен так, чтобы игрок чаще проигрывал, иначе «бандит» не приносил бы доходы своим хозяевам).

А теперь представьте автомат с двумя ручками (таких пока нет), причем известно, что вероятность выигрыша одной ручкой немного больше, чем другой. Но какая это ручка неизвестно (например, решается неким случайным механизмом автомата для каждого игрока отдельно и сохраняется на всю партию).

Легко видеть, что перед вами та же задача о выборе лучшей альтернативы, где результат каждого эксперимента намеренно «зашумлен» случайным механизмом. Алгоритм адаптации должен указать в каком порядке дергать за ручки автомата в зависимости от получаемого результата с тем, чтобы выявить «счастливую» ручку и заплатить при этом минимальную сумму.

Второе (криминальное) объяснение названия «двурукого бандита» навеяно аналогией: представьте себе настоящего бандита, отстреливающегося от полицейских или своих коллег, которым он чем-то не понравился. У него два пистолета разных марок, но вот беда... он не знает свойств этих пистолетов. Они попались ему в руки перед началом перестрелки. Зная по опыту, что пистолеты довольно сильно отличаются точностью стрельбы, бандит, естественно, хочет из двух альтернативных пистолетов выбрать пистолет с лучшим боем. Времени для пристрелки у него нет и приходится адаптироваться, т. е. приспосабливаться к обстановке. Бандит достаточно квалифицирован, чтобы понимать, что промах из какого-то пистолета вовсе не означает, что этот пистолет плохой — с одной стороны могла дрогнуть рука, с другой — промах мог быть следствием нормального разброса, который имеет любое оружие. Именно эти факторы являются «зашумляющими» при выборе лучшего пистолета.

Легко видеть, что перед бандитом со всей неумолимостью встала все та же проблема альтернативной адаптации: по результатам стрельбы (попал — не попал) определять очередную альтернативу (пистолет) с тем, чтобы в конечном счете определить лучший и при этом поразить максимальное число противников (предполагаем, что патронов ему хватает).

Здесь намеренно приведены три обыденных примера задач двуальтернативной адаптации (выбор транспорта, рукоятки и пистолета) с тем, чтобы читатель мог ощутить специфику задачи. Легко видеть, что «двурукость» здесь не принципиальна. Число альтернатив может быть и больше двух. Соответственно алгоритм адаптации, решающий эту задачу, следует назвать «многоруким бандитом».

А теперь рассмотрим задачи альтернативной адаптации ЭВМ. Они возникают благодаря тому, что нельзя заранее определить режим, в котором будет работать ЭВМ. Этот режим зависит от потребностей пользователей, которые предвидеть точно невозможно, а приближенный прогноз всегда бывает настолько приблизительным, что не дает возможности сделать что-либо достаточно надежно (может быть именно поэтому увяла на корню «прогностика» — наука о долгосрочных прогнозах, возникшая лет пятнадцать назад).

Рассмотрим одну из самых распространенных задач, решаемых на ЭВМ, задачу о сортировке массива информации. Формулируется она следующим образом. Пусть в памяти ЭВМ имеется большой массив записей (анкет, таблиц и т. д.) и эти записи нужно упорядочить по какому-то определенному признаку, например, по дате записи или ее номеру, или ее объему, или по какому-либо иному атрибуту. Такие задачи сплошь и рядом возникают при обработке экономической, технологической, коммерческой и многой другой информации. Часто, например, необходимо расположить массив анкет сотрудников предприятия по датам их поступления в организацию, по величине заработка или другим образом.

Задачи такого рода называются задачами сортировки. Для их решения разработаны многочисленные алгоритмы сортировки. (В семитомной монографии Д. Кнута «Искусство программирования» алгоритмам сортировки посвящен один толстенный том — третий). Пользователю, которому предстоит сортировать массив, прежде всего следует выбрать программу сортировки. А таких программ несколько десятков. Какую из них выбрать? Каждая из них решает поставленную задачу, но по-разному. И лишь одна из них решает в минимальное время (это зависит от специфики массива, типа ЭВМ, на которой решается задача, от алгоритма сортировки и других факторов). Действительно, каждая

программа будет сортировать один и тот же массив свое время. Ту, которая сделает это за минимальное время, следует считать наилучшей. Но выяснить это удается лишь после того, как задача сортировки будет решена. А для другого массива оптимальным будет другой алгоритм сортировки.

Дело в том, что каждый алгоритм (а, следовательно, и его программа) ориентирован на определенную специфику массива, который надо упорядочить. Такой спецификой могут быть, например, его частичная упорядоченность в одной половине массива и частичная обратная упорядоченность в другой. Зная это обстоятельство, можно выбрать программу, которая очень быстро упорядочит этот массив. И чем ближе специфика вашего массива к специфике эталонного, для которого разработан алгоритм, тем лучше работаст программа, т. е. быстрее сортируется массив! Но если взять «чужую» программу, т. е. программу, ориентированную на другую специфику, то расплата почти наверняка будет тяжелой: время сортировки увеличится в несколько раз и, естественно, придется в несколько раз больше платить за решение этой задачи (ведь поговорка «время — деньги» особенно справедлива для ЭВМ, каждая минута работы которой обходится обычно в довольно «круглую» копеечку), не говоря уж о том, что при длительной работе ЭВМ увеличивается вероятность сбоя.

Но как узнать специфику массива? Сделать это можно, но проще будет его упорядочить, ведь разбираясь в специфике массива мы, по сути дела, его упорядочиваем! Получился порочный круг: для выбора алгоритма сортировки массива надо знать его специфику, а для ее определения надо упорядочить массив, т. е. решить задачу.

Как быть? Если массивы, поступающие на сортировку, могут иметь любую специфику, то делать нечего, и следует терпеть убыток, рассматривая его как плату за наше незнание специфики массивов, которая может быть какой угодно. Но если, как это и бывает в жизни, какая-то определенная специфика встречается чаще других, то можно использовать это обстоятельство для адаптации процесса выбора программы сортировки.

Пусть, для простоты, имеются лишь две таких программы, а поток поступающих на сортировку массивов

такой, что использование одной из этих программ чаще бывает более выгодно, чем другой, т. е. при этом в среднем времени на сортировку тратится меньше. Как найти эту программу? Ведь каждый конкретный массив отличается именно своей конкретностью, и время, затраченное на его сортировку, вообще говоря, случайно. Вот и возникает задача: каким образом, манипулируя двумя программами сортировки и фиксируя лишь затраченное время, определить, какая из программ сортирует в среднем быстрее и, при этом, не затратить лишнего времени.

Легко заметить, что это задача «двурукого бандита», у которого вместо пистолетов — программы, выстрел — сортировка поступившего массива, а результат — попадание, если время сортировки мало, и промах, если велико.

Аналогично можно сформулировать задачи выбора наилучших программ оптимизации, решения систем алгебраических и дифференциальных уравнений и многих других. Все эти задачи будут решаться алгоритмом «двурукого», а точнее «многорукого бандита».

Теперь рассмотрим этот алгоритм. Попробуем построить разумные стратегии «стрельбы».

Первая (классическая) стратегия опирается на очевидные соображения: нужно попробовать оба пистолета, а потом принять решение. Сделать это можно, например, так. Произведя N выстрелов из одного пистолета и N — из другого, выбрать тот, которым удалось попасть большее число раз. Такое решение вполне оправдано, но получено оно слишком дорогой ценой. Во время экспериментов с пистолетами поведение бандита нельзя считать наилучшим, так как он будет использовать каждую альтернативу N раз вне зависимости от ее эффективности. Это неразумно. Значительно лучше другая стратегия — линейная.

Эту стратегию иногда также называют «игрой победителя». Смысл ее прост и естественен: использовать ту альтернативу, которая на предыдущем шаге принесла успех, и обращаться к другой альтернативе, если успеха не было. В «бандитской» интерпретации это означает, что надо продолжать стрелять из того пистолета, которым попал, и переходить на другой при промахе первым.

Легко заметить, что действуя таким образом бандит убьет больше своих «коллег», чем по первой стратегии.

Это легко показать для одного крайнего случая, когда один пистолет вообще никуда не годится, а второй бьет без промаха. В этом случае первая стратегия дает возможность стрелку свести счеты лишь с N бандитами, а вторая в самом худшем случае, когда он начинает стрелять из плохого пистолета, обеспечивает $2N - 1$ попаданий: один промах и переход на другой пистолет, который обеспечивает остальные $2N - 1$ попаданий. (Здесь для правильного сопоставления стратегий второй стратегии было выделено $2N$ выстрелов, как и для первой.)

Рассмотрим линейную тактику подробнее — она этого заслуживает хотя бы потому, что эта тактика широко используется в природе. Она лежит в основе организации рационального поведения живых существ. Действительно, повторять то действие, которое приводит к успеху, и менять его на другое альтернативное при неудаче, не в этом ли состоит простейшая, но успешная формула адаптивного поведения?

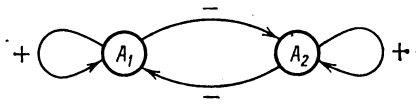

Рис. 3. Так представляется линейная тактика — спасительная тактика биологических систем. Она гарантирует, что неприятности (−) будут избегаться с тем, чтобы поддерживать режим успеха (+). Не так ли, дорогой читатель, мы с вами поступаем в наших жизненных коллизиях? И поступаем правильно!

Алгоритм линейной тактики удобно представить в виде графа, изображенного на рис. 3. Здесь кружками обозначены две альтернативы: первая (A_1) и вторая (A_2). Стрелки указывают переходы, которые реализуют работу алгоритма, а значки рядом со стрелками — условия, при которых следует пользоваться данной стрелкой. Здесь плюс (+) обозначает успех применения альтернативы, а минус (—) — неудачу.

Хорошо видно, что в соответствии с графом при удаче (+) выбирается та же альтернатива, а при неудаче (—) она изменяется на другую. В этом и состоит линейная тактика.

Однако действуя таким образом, мы никогда не выбираем окончательно одну из альтернатив — лучшую. Действительно, достаточно одной неудачи, чтобы в соответствии с линейной тактикой сменить альтернативу. Это линейная тактика без обучения, которое позволило бы продолжать эксплуатировать лучшую альтернативу

даже при неудачном использовании ее. Введем такое обучение.

Психологи, изучающие процессы обучения животных, давно заметили, что обучение является вероятностным процессом. Это означает, что в процессе обучения мы обычно ничего не заучиваем наверняка, а лишь с определенной вероятностью. Если эта вероятность большая, то говорят о хорошем обучении, а если малая, то обучение плохое.

Введем такую вероятность в алгоритм линейной тактики. Будем при неудаче менять альтернативу не всегда, а лишь с определенной вероятностью. Это означает, что всегда существует возможность сохранить альтернативу при ее неудачном использовании.

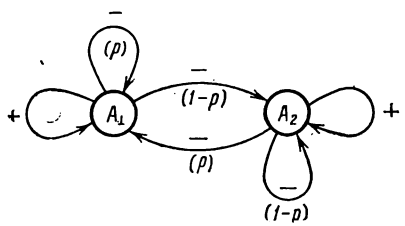

Рис. 4. А это алгоритм линейной тактики с обучением. Здесь обучение производится на базе опыта, полученного линейной тактикой (см. рис. 3). Но это обучение эффективно лишь в том случае, если среда ведет себя достаточно стабильно (стационарно). В очень нестабильной среде лучше использовать алгоритм линейной тактики (рис. 3).

Граф такого алгоритма линейной тактики с обучением приведен на рис. 4. Здесь в скобках обозначены вероятности реализации соответствующих переходов. Так при неудаче на первой альтернативе (A_1) с вероятностью p она сохраняется и с вероятностью $1 - p$ изменяется на A_2. При неудачной работе второй альтернативы она сохраняется с вероятностью $1 - p$ и изменяется на A_1 с вероятностью p. При заданном p этот алгоритм определен полностью и его нетрудно реализовать. Для этого понадобится лишь «разыгрывать» события с заданной вероятностью. Это осуществляется методом Монте-Карло.

Как видно, процесс обучения сводится к определению величины p. Теория утверждает, что эта величина равна вероятности того, что первая альтернатива лучше второй. Оценивать эту вероятность можно на базе предыдущего опыта работы обеих альтернатив, то есть на основе наблюдений за работой алгоритма.

При отсутствии такого опыта $p = 1/2$ и алгоритм работает почти так же, как алгоритм с линейной так-

тикой (чуть-чуть хуже за счет того, что при отрицательном результате он не сразу меняет альтернативу, а лишь с вероятностью $p = 1/2$). Если первая альтернатива лучше второй, то в результате обучения вероятность p стремится к единице ($p \to 1$) и чаще всего реализуется именно первая альтернатива. Если лучше вторая, то в процессе обучения вероятность p стремится к нулю ($p \to 0$) и почти каждый раз, независимо от результатов, реализуется вторая альтернатива (это хорошо видно на рис. 4) *).

Это и есть третья обучаемая стретегия альтернативной адаптации, которая сейчас начинает применяться при адаптации ЭВМ к потребностям пользователя.

До сих пор мы рассматривали общие функции, выполняемые ЭВМ в процессе ее функционирования. Это программируемость ее работы, адаптируемость, специфика взаимодействия с пользователем и другие важнейшие функции, без которых ЭВМ не получила бы столь большой популярности. Теперь же обратимся к решению некоторых конкретных задач. На примере этих задач будут проиллюстрированы некоторые новые функциональные возможности ЭВМ, которые проявляются в конкретных ситуациях.

Начнем с банков данных.

ЭВМ как записная книжка (банк данных)

Мы уже говорили, что одной из наиболее популярных функций современной ЭВМ является хранение информации и выдача ее в требуемой форме. Рассмотрим эту функцию ЭВМ на примере ведения обычной записной книжки. Если по специфике вашей профессии (или в результате вашей общительности) вам необходимо контактировать с большим числом людей, то ЭВМ может оказать много ценных услуг, если вы доверите ей свою записную книжку. Доверяете? Тогда обратитесь к ближайшей ЭВМ и попросите у системного программиста выделить для вас место на магнитном диске и программу банка данных. Это все, что нужно.

Банк данных — это не способ «захоронения» данных, а скорее способ их «оживления». Всякий банк пре-

*) Идея подобных алгоритмов, называемых автоматами с переменной структурой, предложена и разработана советской школой М. Л. Цетлина и В. И. Варшавского, которые заложили ее основы еще в 1960 г.

доставляет пользователю широкие возможности по манипулированию хранящимися в нем данными.

Итак рассмотрим банк под названием «моя записная книжка». Он состоит из записей, каждую из которых можно представить в виде определенной последовательности атрибутов:

$a_1, a_2, \ldots, a_N$,

где a_1 — дата записи (например, 10.3.80), она нужна для контроля.

a_2 — фамилия адресата (например, ИВАНОВ),

a_3 — его имя (например, НИКОЛАЙ),

a_4 — отчество (например, ПАФНУТЬЕВИЧ),

a_5 — город, где живет (например, ЕЛАБУГА),

a_6 — улица (например, КОСОЙ ПЕР., т. е. это не улица),

a_7 — номера дома, корпуса, квартиры (например, 35, —, 17, т. е. нет корпуса),

a_8 — телефон домашний (например, 243),

a_9 — название учреждения, где работает ваш адресат (например, НИИЧАВО),

a_{10} — город, где оно расположено (этот атрибут может не совпадать с a_5),

a_{11} — улица,

a_{12} — номер дома,

a_{13} — служебный телефон адресата, и т. д.

Здесь, как легко заметить, домашний адрес адресата занимает атрибуты $a_5 — a_7$, а служебный: $a_9 — a_{12}$.

Если вы не поленитесь и заполните все атрибуты конкретными значениями слов и цифр всех ваших адресатов и введете их в машину, то получите свой файл в ЭВМ. Файлом называют набор записей, обладающих определенными структурными свойствами. В данном случае файл образован записями адресатов и адресов по определенной структурной схеме $a_1 — a_{13}$, где каждый атрибут имеет вполне определенный и заранее оговоренный смысл (это и есть та самая структура, которая должна быть свойственна всем записям файла, нарушение ее для каких-то записей, например, при ошибках заполнения файла, чувствительно снижает его ценность).

Заметим, что прочерк (—) свидетельствует об отсутствии информации о значении того или иного атрибута в какой-то записи, что также может интересовать владельца файла, например, для того, чтобы устранить тот или иной пропуск.

Приведем примеры работы с таким банком данных.

Каждый вопрос к банку начинается с описания атрибутов, определяющих те записи, которые интересуют пользователя. Например, при необходимости получить телефон определенного лица (пусть Петрова) запрос будет следующим:

А2 = ПЕТРОВ, А8 = ?

Если в банке Петров один, то ЭВМ ответит номером его телефона:

А8 = 1250031

или прочерком (А8 = —), если телефона нет. Если же Петровых много (например, трое), ЭВМ ответит:

ИМЕЕТСЯ ТРИ ЗАПИСИ

и вам придется уточнять вопрос, добавляя значения известных вам атрибутов нужного Петрова, например, название улицы, где он живет:

А6 = КАЛАНЧЕВСКАЯ

или другим способом.

Таким образом можно не только «вспомнить» адреса и телефоны, но и фамилии, по почему-то застрявшим в вашей памяти фрагментам атрибутов; например:

А4 =ПАФНУТЬЕВИЧ, А2=?, А3=?

Если вы расширите записи новыми атрибутами:

a_{14} — пол адресата (М или Ж),

a_{15} — дата рождения (например, 23.7.29).

a_{16} — отношение с адресатом (близкое, хорошее, служебное, шапочное, незнаком),

то банк данных может вам предоставить другие услуги. Например, желая разослать поздравления к 8-му марта, следует сделать запрос:

А14 = Ж; А16 = БЛИЗКОЕ, ХОРОШЕЕ;
А2 — А7 = ?

Ответ машины:

ИМЕЕТСЯ ТРИНАДЦАТЬ ЗАПИСЕЙ.

Следует запросить их адреса:

ВЫВЕСТИ ВСЕ

и вы получите все адреса ваших близко и хорошо знакомых женщин с их фамилиями, именами и отчествами (если они есть в банке).

Не желая пропускать пятидесятилетнего юбилея Ваших близких знакомых (приятелей) сделайте запрос:

А14 = М; А15 = —, —, 33; А16 = БЛИЗКОЕ; А2 = ?

и вы получите фамилию того, кому из них в 1983 году исполнится 50 лет (прочерки в последнем запросе нужны для того, чтобы поиск велся только по году рождения).

А желая поздравить своих близких с днем рождения в ноябре запрашивайте так:

А16 = Б; А15 = —, 11, —; А2 = ?

Ответ:

ЗАПИСЕЙ НЕТ

избавит вас от необходимости писать поздравления.

В предыдущем вопросе видно, что нет необходимости писать полностью значения атрибутов. Достаточно взять столько букв, чтобы не возникло неопределенности. Так для атрибута a_{16} (отношение с адресатом) достаточно указать одну букву из Б, Х, С, Ш и Н, каждая из которых однозначно определит значение атрибута (Близкое, Хорошее и т. д.). Для фамилий и имен следует брать больше букв. Так, например, значение:

А3 = АЛЕКС

может быть и Алексом, и Алексеем, и Александрой, и Александриной.

Иногда приходится задавать вопросы, включающие знаки $>$ (больше), $\geqslant$ (не меньше), $<$ (меньше) и $\leqslant$ (не больше). Например, желая собрать своих молодых друзей на вечеринку, делайте запрос:

А5 = РИГА; А15 = —, —, $\geqslant$53; А16 = Б; А2 = ?; А3 = ?; А8 = ?; А13 = ?

по которому ЭВМ вам выдает фамилии, имена и телефоны (домашний и служебный) всех ваших друзей в Риге в возрасте не старше 30 лет (в 1983 году).

Можно пользоваться также процедурой определения минимального или максимального значения атрибута. Например, желая определить дату самой старой записи (например, для ее коррекции), следует задать вопрос:

А1 = МИН; А1 = ?,

а фамилию самого молодого из ваших корреспондентов ЭВМ выдает на запрос:

А15 = —, —, МАКС; А2 = ?

Очень удобно ввести атрибут

a_{17} — прозвище.

Так как прозвище очень хорошо ассоциируется с адресатом, по нему очень удобно вести его поиск. Например, на запрос:

А17 = СВ; А13 = ?

ЭВМ укажет вам служебный телефон Сверчка.

Несколько слов о заполнении банка. Каждую новую запись следует начинать командой ВПИСАТЬ ЗАПИСЬ (или ВП ЗАП), после чего ЭВМ сообщаются значения атрибутов новых записей. Например:

ВП ЗАП А13 = 2873416; А3 = КОСТЯ; А17 = УСАТЫЙ.

(Порядок заполнения атрибутов не играет роли.) Эта короткая запись определит вам служебный телефон усатого Кости. Дату заполнения этой записи банк проставит сам — он знает календарь. Так что, если раньше был записан другой усатый Костя, то их записи будут различаться датами заполнения. И того, другого, будет легко выделить значением первого атрибута:

А1 < (дата),

где указывается дата (например, —. —. 80), позже которой первый Костя не поступал в банк наверняка (это было не позже конца 1979 г.).

Можно добавлять в банк новую информацию относительно старых знакомых. Так на вопрос знакомого: «А есть ли у вас мой домашний телефон?» вы обращаетесь к ЭВМ с вопросом:

А2=(его фамилия), А3 = (имя), А8 = ?

Если будет ответ:

А8 = —

то телефона нет и его надо ввести в файл. Делается это просто — вы отдаете команду ВПИСАТЬ:

ВП А8 = (номер домашнего телефона).

Так можно сделать лишь после предыдущего запроса, который определяет ту запись, где надо делать дополнения. Если же этого не было, надо предварительно ввести определяющие атрибуты (обычно это a_2 и a_3):

А2 = (фамилия); А3 = (имя); ВПИСАТЬ А8 = = (телефон).

При изменении значения атрибута следует делать то же, что и при добавлении. Новое значение атрибута при этом сотрет старое. Например, запись:

А2 = ИВАНОВА; ВП А2 = ПЕТРОВА

означает, что ваша знакомая Иванова вышла замуж за Петрова. Если же вы хотите запомнить ее девичью фамилию, то надо вписать:

ВП А2 = ПЕТРОВА (ДЕВ. ИВАНОВА).

Эту запись ЭВМ будет выдавать вам на запрос А2 = ? Но поиск она будет вести по новому значению А2 = = ПЕТРОВА.

Изменения в своем файле вы можете делать самостоятельно. А вот введение новых атрибутов необходимо согласовывать с администратором банка — так торжественно называют системного программиста, ведущего программу банка и необходимую для него память (ведь эта программа используется не только вами). Введение новых атрибутов (например, a_{18} — чем интересен) изменяет структуру вашего файла, для чего необходимо выделить дополнительную память на диске (или ленте). Это делают хозяева программного обеспечения ЭВМ — системные программисты. Один из них и является администратором банка. Именно к нему вам следует обратиться при изменении структуры своего файла.

Эту структуру следует помнить. А если вы ее забыли, то не печальтесь и запросите ее командой:

СТРУКТУРА ФАЙЛА = ?

Ответом машины будет описание структуры, сделанное вами под руководством администратора банка:

А1 — (ДАТА ЗАПОЛНЕНИЯ ЗАПИСИ) = X, Y, Z
X — ЧИСЛО ДАТЫ
Y — МЕСЯЦ ДАТЫ (ЦИФРА)
Z — ГОД ДАТЫ (ДВЕ ПОСЛЕДНИХ ЦИФРЫ).

(Если вы запрос сделаете буквами, например, ИЮНЬ, то машина вас не поймет. Для этого нужно было ввести в структуру:

Y — МЕСЯЦ ДАТЫ (ЦИФРА ИЛИ СЛОВО)

и снабдить ЭВМ правилом перевода: 1 = 01 = ЯНВАРЬ, 2 = 02 = ФЕВРАЛЬ и т. д.).

A2 — (ФАМИЛИЯ)
СЛОВА

. . . . . .

A7 = (X, Y, Z)
X — номер дома (можно дробь, например: 45/47),
Y — номер корпуса (прочерк при отсутствии),
Z — номер квартиры (прочерк при отсутствии).
A8 = (ТЕЛЕФОН)
ЛЮБОЕ ЧИСЛО
A9 = (УЧРЕЖДЕНИЕ)
СЛОВА И ЦИФРЫ

. . . . . .

A14 = (ПОЛ)
БУКВЫ: М, Ж
A15 = (ДАТА РОЖДЕНИЯ) = X, Y, Z
см. A1.

(т. е. этот атрибут заполняется так же, как атрибут даты записи)

A16 = (отношение с АДРЕСАТОМ)
Слова: БЛИЗКОЕ, ХОРОШЕЕ, СЛУЖЕБНОЕ, ШАПОЧНОЕ, НЕЗНАКОМ.
A17 (ПРОЗВИЩЕ)
Слово без кавычек

. . . . . .

Эту структуру ЭВМ по вашему требованию может напечатать. Как видно, при описании структуры файла необходимо не только указать на содержательный смысл атрибутов — они описаны в круглых скобках, но и иногда привести словарь и форму заполнения файла или запроса. При изменении словаря изменяется структура файла, что необходимо согласовывать с администратором банка.

Банк может вам оказать и другие услуги при ведении своего файла. Так, например, не надо каждый раз записывать шестизначный почтовый индекс. Достаточно один раз ввести в память ЭВМ индексный справочник и указать, чтобы при выведении значений атрибутов $a_5 - a_7$ и $a_{10} - a_{12}$ (это адреса) банк обращался к этому справочнику и сопровождал бы адрес его индексом по справочнику. Этим справочником, очевидно, смогут пользоваться и другие «клиенты» банка (только не следует забывать, что в большом справочнике одно и то же название имеют несколько городов. Так,

например, если две Риги: в Прибалтике и в Сибири. ЭВМ выдаст вам соответственно два почтовых индекса).

Записная книжка — дело сугубо личное. Поэтому и ваш файл вы вправе скрывать от других, т. е. засекретить. Делается это просто. Для того, чтобы считывать содержимое вашего файла, нужно иметь пароль — ключевое слово, которое открывает доступ к файлу, например, «сезам» (один из банков данных так и называется: «сезам»). Это слово можно менять время от времени. Делает это администратор банка.

Каждый сеанс вашего общения с банком в этом случае должен начинаться словами:

ФАЙЛ N, ПАРОЛЬ СЕЗАМ,

где N — номер вашего файла. Получив такое сообщение, ЭВМ сверит этот пароль со словом, хранящимся в памяти, и при их совпадении ответит:

ФАЙЛ N ОТКРЫТ,

после чего можно начинать работать, т. е. задавать вопросы или вводить новые записи в свой файл. Только не забудьте закрыть файл после окончания работы с ним сообщением

КОНЕЦ СЕАНСА,

которое закрывает свободный доступ к вашему файлу.

Если же пароль не верен (т. е. не совпадает со словом, хранящимся в ЭВМ), то поступит ответ:

ФАЙЛ N ЗАКРЫТ.

Пароль знаете вы и администратор банка и ему, естественно, тоже доступен ваш файл. Если же вы захотите засекретить содержание файла и от него, банк предоставит вам и такую услугу. Для этого содержимое файла следует записывать в каком-то коде, ключ (точнее: шифр) которого знаете только вы — хозяин файла. Вы в начале сеанса вводите в ЭВМ сообщение:

ФАЙЛ N КЛЮЧ НИБУМБУМ.

ЭВМ, получив этот ключ («нибумбум»), которым может быть любое слово, число или фраза, сама программирует шифратор и дешифратор и записывает вводимую в файл информацию только в зашифрованном виде. А при выведении содержимого файла пользователю — предварительно расшифровывает ее.

Разница между паролем и ключом заключается в том, что в памяти ЭВМ нет ключа и она каждый раз в начале сеанса синтезирует программы шифрации и дешифрации по вашему «нибумбуму», а в конце сеанса стирает эти программы в своей памяти (об этом позаботится банк). Теперь никто не сможет узнать содержимого вашего файла! Разве что подберут «отмычку», но это уже уголовное дело. А если вы забыли и кодовое слово, то вам не поможет даже администратор банка. Файл в этом случае следует считать потерянным, как бы ценен он не был. Правда, если обратиться к специалистам по расшифровке тайных кодов (криптологам), то они смогут вернуть вам «нибумбума». Но эту услугу банк вам не окажет, хотя в принципе и мог бы — структура программы шифровки ему известна, а для восстановления кодового слова достаточно сравнить несколько слов с их шифром и... вычислить его. Однако программы таких вычислений в банке нет — иначе было бы бессмысленно вводить шифрование.

(Заметим, что принципиальная возможность расшифровки закодированного файла иногда беспокоит клиентов банка. Однако средства шифровки могут быть настолько сильными, что информация в зашифрованном файле практически абсолютно секретна.)

Компьютер-репетитор (ЭВМ в обучении)

Едва ли кого удивишь тем, что «ученье — свет, а...». Но поговорка ничего не сообщает о том, как надо учиться. А эта проблема с каждым годом становится все острее и острее. Причина — колоссальные темпы развития современной науки, с одной стороны, и старые-престарые методы обучения — с другой. Сегодня студентов и школьников учат так же, как в Древней Греции или в Средневековье: собирают в группы и классы и читают им лекции, время от времени проверяя усвоение предмета зачетами и экзаменами. В результате дидактика не поспевает за наукой и процесс обучения и подготовки специалистов все больше и больше удлиняется: к десяти годам обучения в школе и пяти в институте — три года аспирантуры. А докторантура? Многочисленные курсы повышения квалификации и курсы переподготовки? Если пойдет так и

далее, то в будущем человеку, закончившему курс обучения, придется уходить на пенсию, так как срок обучения возрастает, а пенсионный возраст неизменно снижается. Так вот и придется просиживать за партой всю жизнь, если не придумают более эффективных методов обучения.

Что же такое эффективные методы обучения? Это прежде всего методы индивидуального обучения. Ими широко пользовались состоятельные дворяне в дореволюционной России (нанимали гувернеров), ими и сейчас пользуются отстающие ученики, когда ходят к репетитору. Репетитор хорош прежде всего тем, что учитывает индивидуальные особенности ученика — запас знаний, специфику памяти, психики, темперамента и т. д.

Нельзя ли эту функцию поручить ЭВМ?

Для ускорения темпов обучения, для повышения его эффективности и качества ныне разрабатываются автоматизированные обучающие системы (АОС), использующие большие ЭВМ. Процесс обучения в АОС осуществляется следующим образом. Человеку выдается порция информации, которую он должен изучить или заучить. Затем задается один или несколько вопросов, по которым определяется степень усвоения данной порции. После этого в системе осуществляется проверка правильности ответов и определяется следующая порция информации, которая и сообщается обучаемому (определяется эта порция согласно заложенному в ЭВМ алгоритму обучения, то есть правилу, по которому ЭВМ учит ученика).

Рассмотрим работу АОС в многострадальной задаче обучения иностранному языку. Большинство людей, его изучающих, нуждается лишь в умении понимать специальную литературу на иностранном языке. И только. Все остальное — грамматика, произношение и прочее нужны лишь при специальном, глубоком обучении и немногим: дипломатам, преподавателям, лингвистам и т. д. Подавляющее же большинство «потребителей» иностранного языка лишь жаждут понимать тексты по своей специальности.

Это последнее обстоятельство очень важно, так как всякая область науки и техники пользуется достаточно ограниченным запасом слов и весьма простой грамматикой. Научить этим словам и грамматике (ввиду их простоты) уже может ЭВМ.

Но при обучении обязательно надо, как сказано, учитывать индивидуальные способности каждого. Здесь опять на помощь приходит ЭВМ. Она без труда определит, каким словарным запасом обладает тот или иной человек, что он знает лучше и что — хуже, как быстро забывает различные слова, т. е. каковы свойства его памяти и т. п. В зависимости от этого «диагноза» машина подберет наилучший для него алгоритм (правило) обучения, что поможет ученику достигнуть заданной цели за минимальное время.

Представьте себе, что рядом с вашим телевизором стоит небольшой пульт, снабженный клавиатурой с русским и латинским шрифтами. Рядом — печатающее устройство, например, в виде электрической пишущей машинки. Пульт с телевизором и печатающее устройство образуют минимальный комплект для работы с ЭВМ. Пульт с телевизором заменяют уже известный нам дисплей — это идеальное средство диалога с ЭВМ, а для получения «домашнего задания» от ЭВМ служит пишущая машинка. Но где ЭВМ? Ее здесь нет. Она расположена в одном из ближайших вычислительных центров вашего города, и обслуживает много таких учеников как вы. Набирайте номер этого центра и просите оператора соединить вас с программой обучения иностранному языку (например, английскому). Оператор ответит согласием или попросит немного подождать, чтобы поставить в магнитофон ЭВМ магнитную ленту, на которой записан курс обучения английскому языку.

Вы кладете телефонную трубку на пульт, который специально для этого имеет углубление для трубки, и обучение с ЭВМ начинается. На экране вашего телевизора возникают слова:

СИСТЕМА АСОЛИЯ ПРИВЕТСТВУЕТ ВАС! ЗАРЕГИСТРИРОВАНЫ ЛИ ВЫ В СИСТЕМЕ (ДА ИЛИ НЕТ)?

АСОЛИЯ — это Автоматизированная Система Обучения Лексике Иностранного Языка. Дело в том, что для обучения пониманию специальных текстов нет необходимости специально изучать грамматику. Главное — это лексика, т. е. прежде всего надо знать те иностранные слова, которые используются в интересующих вас текстах. Зная слова и представляя о чем идет речь (вот здесь и используется «специальность» текста) легко понять смысл этого текста.

Вы набираете на клавиатуре «нет» и на экране читаете:

ИЗУЧАЛИ ЛИ ВЫ АНГЛИЙСКИЙ ЯЗЫК:
1 — НЕТ
2 — УЧИЛ В ШКОЛЕ
3 — УЧИЛ В ИНСТИТУТЕ
4 — УЧИЛ И В ШКОЛЕ. И В ИНСТИТУТЕ.

(Это уже известное вам меню.)

Вы сообщаете все запрашиваемые ЭВМ сведения о себе и читаете:

КАКОЙ КУРС ВЫ ЖЕЛАЕТЕ ВЫБРАТЬ ДЛЯ ОБУЧЕНИЯ?
1 — РАЗГОВОРНАЯ ЛЕКСИКА:
1.1 — ДЛЯ ОТЕЛЯ
1.2 — ДЛЯ РЕСТОРАНА
1.3 — ДЛЯ МАГАЗИНА
и т. д.
2 — МАТЕМАТИЧЕСКИЕ ТЕКСТЫ:
2.1 — ЭЛЕМЕНТАРНАЯ МАТЕМАТИКА
2.2 — МАТЕМАТИЧЕСКИЙ АНАЛИЗ
2.3 — ТЕОРИЯ ВЕРОЯТНОСТЕЙ И МАТЕМАТИЧЕСКАЯ СТАТИСТИКА
2.4 — ТЕОРИЯ ИНФОРМАЦИИ
и т. д.
3 — ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА:
3.1 — ДИАГНОСТИЧЕСКИЕ СООБЩЕНИЯ ОПЕРАЦИОННОЙ СИСТЕМЫ ЕС ЭВМ ОПЕРАТОРУ
3.2 — ДИАГНОСТИЧЕСКИЕ СООБЩЕНИЯ ОПЕРАЦИОННОЙ СИСТЕМЫ ЕС ЭВМ ПРОГРАММИСТУ
3.3 — ЛЕКСИКА ПО ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКЕ ДЛЯ СИСТЕМНЫХ ПРОГРАММИСТОВ
и т. д.

(Заметим, что последний, третий раздел уже реализован и с его помощью обучаются программисты, не знающие английского языка.)

Вам предлагаются курсы, которые составлены по оригинальным текстам на английском языке, в той или иной предметной области. В основе каждого курса лежит словарь слов, словосочетаний и грамматических форм, полученный в результате обработки огромного числа таких текстов (ее также производила ЭВМ).

Набирайте на пульте шифр выбранного вами курса. Система регистрирует вас, запоминает шифр и присваивает вам номер, под которым вы будете в ней фигурировать. Этот номер следует запомнить, в начале каждого сеанса обучения надо будет его набирать.

Первое занятие начинается с того, что на экране телевизора высвечиваются иностранные слова и словосочетания с переводом. Приводятся примеры использования их в тексте. Затем дается упражнение на использование этих слов. Если, выполняя упражнение, вы ошиблись, ЭВМ поможет исправить ошибку. В конце занятия можете получить слова и словосочетания, которые высвечивались на экране. Печатающее устройство (оно управляется ЭВМ) напечатает домашнее задание: слова и словосочетания с транскрипцией и переводом.

На другой день вызывая программу обучения «Иностранный язык» на вопрос, зарегистрированы ли вы в системе, отвечаете «ДА» и набираете свой номер. Новый сеанс начинается с проверки того, как вы запомнили слова и словосочетания заданного урока. Даете перевод, и ЭВМ сообщает, правилен ли он. Теперь она о вас уже кое-что знает. Ей известно, сколько слов вы забыли, какие это слова. Очевидно, именно их нужно будет когда-то повторить. Так, после каждого сеанса обучения ЭВМ будет знать о вас все больше и больше. Это поможет ей лучше исполнять роль учителя. Она будет «натаскивать» вас по тому материалу, который вы забыли или нетвердо усвоили. Кроме того, она знает, что одни слова встречаются в текстах чаще, а другие — реже, и ей известна частота появления того или иного слова в интересующих вас текстах. (Частоты получаются в результате машинной обработки текстов.) Очевидно, что чаще других ей следует выдавать те слова, частота встречаемости которых выше.

И последнее, ЭВМ должна иметь возможность оценивать числом уровень обученности каждого ученика, т. е. следует иметь формальный критерий степени обученности. Таким критерием является, например, вероятность того, что ученик знает слово, наугад выбранное из текста, пониманию которого он учится.

Эту характеристику можно определить экспериментально, но на это уйдет много времени. Поэтому нужно иметь возможность вычислять значение критерия обученности, минуя эксперимент. Именно этому служит модель ученика.

Эта модель представляет собой набор вероятностей знания учеником тех слов, которым его учат. Оценку этих вероятностей (точнее: коррекцию оценки) ЭВМ делает при каждой «встрече» с учеником. Именно в этом состоит адаптация модели.

В модели отражается специфика памяти ученика. Всем известно, что мы запоминаем по-разному. Одни слова запоминаются сразу, а с другими приходится мучиться долго. Это зависит от индивидуальности ученика, его ассоциаций, жизненного опыта и т. д. Все это и отражается в его модели.

Располагая моделью, ЭВМ легко строит процесс обучения так, чтобы быстрее научить данного ученика. Для этого достаточно на модели выбрать такую очередную порцию обучающей информации, которая максимально повысит его обученность.

Так шаг за шагом ЭВМ по ответам будет строить модель обучаемого и учить его наилучшим образом, что делает время его обучения минимальным.

В процессе обучения ученик сам определяет темп обучения, указывая ЭВМ, какое время он выделяет на заучивание слов. ЭВМ соответственно выдает ему различные по объему порции обучающей информации. Это очень важно, например, тогда, когда ученик торопится в заграничную командировку и ему «позарез» и быстро нужно освоить лексику разговора на таможне, в отеле и ресторане, а также в магазине. АСОЛИЯ в этих условиях является незаменимым инструментом быстрого обучения.

Ответы ученика используются не только для коррекции параметров модели. Модель, которую мы выбрали для данного обучаемого, может быть не адекватна ему. Как правило, имеется несколько альтернативных моделей, из которых в процессе обучения необходимо выбирать модель, адекватную данному обучаемому. Такой выбор модели — тоже адаптация, но уже адаптация структуры модели. Осуществляется она все тем же алгоритмом «многорукого бандита». Чем вернее будет модель, т. е. чем больше она будет соответствовать поведению обучаемого, тем правильнее определит программа, чему учить, какую информацию выдать обучаемому, и тем быстрее выучится ученик.

При этом алгоритм обучения — правило, по которому определяются слова и словосочетания, необходимые для обучения в данный момент,— может меняться в за-

висимости от индивидуальных способностей обучаемого. Значит, в процессе обучения осуществляется не только выбор и коррекция модели самого обучаемого, но и выбирается наиболее подходящий для него учитель! Все это позволяет построить оптимальное управление процессом обучения, а именно, достигнуть желаемого эффекта за минимальное время.

Описанная автоматизированная система обучения — не фантазия автора, а реальность, в создании которой он принимал участие.

Абонент системы АСОЛИЯ так быстро запоминает иностранные слова и словосочетания, как ни при одном другом методе обучения. На заучивание одной лексической единицы, т. е. проще — на одно слово, ученику в среднем приходится тратить одну минуту. Это в три раза лучше известной Лозановской системы обучения. Правда, при этом ученик не научается говорить — АСОЛИЯ этому не учит, но ведь знание слов является основой знания языка. Специалист, знающий слова, без труда переводит (точнее: понимает) текст из своей области (опыты показали, что для этого достаточно знать 70—75% встречающихся слов). А обучение с помощью системы АСОЛИЯ студентов показало, что они с меньшими затратами времени обучились вдвое лучше, чем «старым» способом.

Таким образом, ЭВМ может помочь ученику накопить определенный запас иностранной лексики в интересующей его области, научить применять определенные речевые модели, научить понимать иностранные тексты. Заметим, что это нисколько не умаляет роли преподавателя, а лишь облегчает его задачу — обучать специфике языка, его грамматике, речевому общению и т. д.

Тандему «преподаватель — ЭВМ» принадлежит будущее!

ЭВМ в автомобиле (микропроцессоры)

На первый взгляд ЭВМ нечего делать в автомобиле. Действительно, с пятью органами управления всякой автомашины (руль, газ, тормоз, сцепление и рукоятка переключения передач) легко осваивает любой новичок за два-три часа езды с инструктором. И что тут можно, да и главное, нужно, ли вычислять? — Оказывается, очень нужно!

Начнем с управления. Если сравнить расход топлива у новичков и опытного водителя, то разница будет очень большая. А иные шоферы — любители так на всю жизнь и остаются новичками. ЭВМ поможет им, да и не только им, но и опытным водителям, экономично, а следовательно — эффективно расходовать топливо.

Не спешите с выводами о пустяшности такой экономии (один-два литра бензина на 100 км пробега) и вспомните о том, что число автомашин быстро растет, а нефть является (увы!) не восстанавливаемым ресурсом. ЭВМ поможет не только сэкономить бензин, но и тем самым поможет человечеству отодвинуть момент исчерпания этого дорогого ресурса. При этом не следует забывать, что полное сгорание топлива обеспечивает не только экономичность, но и гарантирует минимальную токсичность выхлопных газов. Этот фактор очень важен в наше «автомобилизированное» время и особенно для жителей больших городов. (Вспомним печальную историю англичан. Им удалось справиться со своим знаменитым лондонским смогом после запрещения пользоваться углем для каминов, не обеспечивающих полного сгорания. Но взамен они получили автомобильный смог, вызванный неполным сгоранием топлива в автомобильных двигателях.) ЭВМ может помочь и здесь и это лишь одна из ее функций в автомобиле (как она реализуется — будет сказано чуть позже).

Другая не менее важная автомобильная функция ЭВМ заключается в выполнении роли штурмана, т. е. прибора, прокладывающего маршрут движения автомашины. Задача прокладки маршрута далеко не так проста, как это может показаться сначала, особенно в большом городе. Представьте, что перед вами карта города. Она может быть выполнена на специальном планшете или высвечена на экране автомобильного телевизора. Кроме этого есть три кнопки: «СТАРТ», «ФИНИШ» и «ОБЪЕЗД».

Перед поездкой вы нажимаете кнопку «СТАРТ» и после этого прикасаетесь пальцем к точке старта на карте. Этим вы сообщаете ЭВМ исходную точку вашего маршрута. Конечная цель маршрута вводится в ЭВМ аналогично прикосновением пальца к карте с предварительным нажатием кнопки «ФИНИШ».

Теперь ЭВМ знает все, что необходимо для прокладки маршрута (карта дорожной сети города нахо-

дится в ее памяти). ЭВМ вычисляет кратчайший маршрут и сообщает вам на табло (или на экран телевизора) примерное время, которое придется затратить на прохождение маршрута. Точно сделать прогноз временны́х затрат нельзя, так как это зависит от ситуации, сложившейся в данный момент в городе (загрузка улиц транспортом, состояние дорожного покрытия, атмосферные условия и т. д.). Если ЭВМ снабдить этой информацией, то прогноз был бы точнее. А пока на экране вашего дисплея появляется оценка

ВРЕМЯ ПРОХОЖДЕНИЯ МАРШРУТА 43 ± 12 МИН.

Самое худшее время (55 мин.) рассчитано на неудачное стечение дорожных обстоятельств, а самое лучшее (31 мин.) — при идеальных условиях движения, но с учетом задержек на повороты и светофоры. Если ЭВМ снабдить календарем и часами, то прогноз будет еще точнее (она будет учитывать повышение потока машин в часы «пик», снижение его в субботу и воскресенье и т. д.).

Если предложенная ЭВМ оценка времени вас устраивает (и вы безнадежно не опоздали), то нажимайте снова кнопку «СТАРТ», что значит ваше согласие с предложенным маршрутом.

Теперь на экране дисплея появится название улицы, по которой надо начинать движение. Например,

ДВИГАТЬСЯ ПО УЛ. ЛЕНИНА ДО ПЛ. СВОБОДЫ.

Вы включаете двигатель и начинаете движение по ул. Ленина в сторону пл. Свободы. На табло ЭВМ сообщает вам расстояние до первого поворота. Например,

ДО ПОВОРОТА НАПРАВО 850 М,

причем число будет все время уменьшаться — ведь ЭВМ легко вычисляет расстояние вашей машины до нужного поворота. Для этого ей все время автоматически сообщается скорость движения машины.

Пройдя поворот (об этом ЭВМ узнает по соответствующему повороту рулевого колеса и снижению скорости), вы получите очередную информацию вашего штурмана:

ДО ПОВОРОТА НАЛЕВО 3270 М.

Причем о возможности сделать такой поворот, т. е. что там нет запрета на него, ЭВМ знает заранее — это обозначено на карте, хранимой в ее памяти.

Но дорога всегда полна неожиданностей: где-то произошла авария и ждут милицию, где-то перерыли улицу и поставили «кирпич», где-то недавно сделали одностороннее движение и т. д. Обо всем этом ЭВМ, естественно, не знает. Столкнувшись с препятствием такого рода вы должны сделать его объезд. В этом тоже поможет ЭВМ. Нажав кнопку «ОБЪЕЗД» вы сообщаете о своих затруднениях ЭВМ и она прокладывает маршрут объезда с учетом сложившихся обстоятельств.

Как видите, ЭВМ не руководит вами, а лишь советует. А вы вправе поступать как угодно. Такое распределение обязанностей следует считать правильным, так как водитель всегда располагает большей информацией о ситуации на дороге, чем ЭВМ, даже снабженная телевизионным глазом. Именно поэтому проекты, в которых ЭВМ доверяют «баранку», следует считать проектами очень далекого будущего и пока они обсуждаются лишь в научной фантастике.

Совсем другое дело при управлении двигателем. Здесь ЭВМ вполне можно «доверять», так как она будет действовать на основе информации недоступной человеку. Такой информацией являются

x_1 — обороты двигателя (об/мин.),

x_2 — нагрузка (крутящий момент) (кг/см),

x_3 — подача топлива в двигатель («газ») (г/с.),

x_4 — температура двигателя (град. С),

x_5 — температура горючей смеси, которая зависит в основном от температуры среды (набегающего воздуха) (град. С).

Из этих пяти параметров только четвертый (температура двигателя) контролируется на всех автомобилях, а обороты (x_1) контролируются из отечественных автомобилей) только на «Жигулях» высших моделей. Остальные недоступны водителю. Скажем прямо, что это не вина автоконструкторов. Они смогли бы без особого труда поставить датчики крутящего момента (x_2), расхода топлива (x_3) и температуры топливно-воздушной смеси (x_5). Но зачем они водителю? Даже, если он знает, что все параметры однозначно определяют оптимальные значения y_1 — обогащенности смеси (весовое соотношение топливо-воздух) и y_2 — угол опережения зажигания, при которых топливо сгорает пол-

ностью, то определить эти значения он не сможет, так как для этого нужно произвести довольно сложные вычисления. Единственно, что каждый водитель твердо знает, что холодный двигатель (при $x_4 \leqslant 50°$) следует заводить на богатой (точнее обогащенной) смеси, т. е. при прикрытой воздушной заслонке. Проще говоря, надо вытянуть ручку «подсоса», которая закрывает доступ воздуха в карбюратор, что приводит к резкому обогащению смеси. Но нельзя и «пересосать», так как при малом поступлении воздуха смесь не будет гореть и двигатель заглохнет, на чем новичок обычно и «спотыкается», когда заводит холодный двигатель.

Очевидно, что обогащение смеси должно быть оптимальным (y_1) и зависеть каким-то определенным образом от всех указанных параметров $x_1, \ldots, x_5$. Так же определенным образом зависит от всех указанных параметров и оптимальный угол опережения зажигания y_2.

Что значит «зависит каким-то определенным образом»? Для того, чтобы определить эту зависимость надо обратиться или к теории двигателей внутреннего сгорания или к эксперименту, который проводят на специальных стендах, что дает возможность установить, как конкретно зависят искомые оптимальные значения параметров смеси (y_1) и угла опережения зажигания (y_2) от параметров работающего двигателя x_1 и x_2. Пусть эти зависимости имеют вид:

$$y_1 = f_1(x_1, x_2, x_3, x_4, x_5),$$
$$y_2 = f_2(x_1, x_2, x_3, x_4, x_5),$$

где f_1 и f_2 — определенные функции, полученные теоретически или экспериментально, и указывающие каким образом можно вычислить y_1 и y_2 по измерениям $x_1, \ldots, x_5$. Очевидно, что эти функции для каждого типа двигателя свои.

Чтобы их использовать, необходимо иметь возможность:

1. Быстро измерять параметры $x_1, \ldots, x_5$ работающего двигателя.

2. Быстро вычислять значения функций f_1 и f_2, т. е. определять оптимальные значения y_1 и y_2.

3. Быстро реализовать эти значения, т. е. сделать так, чтобы обогащенность поступающей в двигатель смеси стала равной только что вычисленному

значению y_1, а угол опережения зажигания — полученной величине y_2.

Первую функцию (измерения параметров) выполняют датчики — специальные приборы, преобразующие измеряемый параметр (обороты, момент, температуру и т. д.) в определенное электрическое напряжение или ток.

Третью функцию преобразования вычисленных значений y_1 и y_2 (они также обычно представляются в виде определенного электрического напряжения или тока) в определенную обогащенность смеси и угол опережения зажигания выполняют так называемые исполнительные механизмы (или сервоприводы). Один из них поворачивает воздушную заслонку, а другой изменяет угол опережения так, чтобы они были равны только что вычисленным оптимальным значениям y_1 и y_2.

ЭВМ нужна для выполнения второй (вычислительной) функции. Она должна быть запрограммирована на вычисление значений y_1 и y_2 по значениям $x_1, \ldots, x_5$, поступающим от датчиков. Для этого вполне достаточно маленькой вычислительной машины. Она должна быть проще универсальной ЭВМ, так как выполняет лишь одну программу — вычисления двух заданных функций f_1 и f_2. Причем эта программа должна сохраняться при выключении питания ЭВМ.

Необходимость в малогабаритных и экономичных ЭВМ с указанными специфическими свойствами для управления приборами привела к созданию специальных ЭВМ, называемых микропроцессорами. Использование такого микропроцессора для управления работой двигателя позволяет поддерживать оптимальный режим его работы при любых обстоятельствах, т. е. полностью и качественно использовать топлива, а, следовательно, максимально экономить его и не отравлять среду продуктами неполного сгорания. Запуск двигателя с микропроцессором не представляет проблем, так как водителю не нужно работать с «подсосом» и бояться «пересосать». Быстродействие такого управления зависит не от микропроцессора — он выполняет свою функцию — вычисление состава y_1 оптимальной смеси и оптимального угла опережения зажигания y_2 — очень быстро, как всякая электронная вычислительная машина, и надежно. Оперативность такой системы управления определяется исполнительными механизмами, кото-

рые «отрабатывают» команды y_1 и y_2, выдаваемые микрокомпьютером.

Скажем прямо, что такой системы управления режимом работы двигателя в массовом автомобилестроении пока нет. И виной тому не ЭВМ, а исполнительные механизмы, которые пока слишком громоздки, ненадежны и дороги, а ЭВМ в виде микропроцессора есть: дешевые, надежные и в большом ассортименте.

Что же такое микропроцессор?

Функционально это процессор, выполненный на одном кристалле. Внешне он представляет собой «паучка» размером $20 \times 10 \times 5$ мм с $20 \div 30$ ножками — выводами. Отличается микропроцессор не только своими малыми размерами, но и малой разрядностью чисел. Дело в том, что большая разрядность, отличающая большие ЭВМ, минипроцессору не нужна — он справляется со своими обязанностями малым числом разрядов $(4 \div 16)$.

Для работы микропроцессора, как и любого процессора, нужна память, где располагается программа, которую он выполняет. Эта память образуется на том же кристалле, а функционально является постоянным запоминающим устройством (его сокращенно называют ПЗУ). Отличие этого вида памяти в том, что она не стирается при выключении питания.

Как видно, функция, выполняемая микропроцессором, определяется программой, хранимой в постоянной памяти. Такое сочетание универсального микропроцессора с постоянной памятью, специализирующей его на выполнение строго определенной функции, оказалось очень удачным и быстро завоевало признание в мире.

Появились микропроцессоры в начале 70-х годов и сразу нашли широкое применение в различных областях науки и техники. Стоимость их невелика и все время снижается, а производство неуклонно растет.

Причина популярности микропроцессоров заключается в том, что с их появлением отпала необходимость в разработке специализированных схем обработки информации, которые в большом количестве нужны во всякого рода системах управления и переработки информации. Вместо проектирования и изготовления нужной схемы достаточно ее функцию запрограммировать и ввести в ПЗУ микропроцессора, который этим и

образует необходимую систему обработки информации. Это значительно упрощает процесс проектирования и создания систем управления, в том числе и очень сложных, за счет различных комбинаций наборов микропроцессоров.

Если микропроцессор выполняет функции управления, то его называют контроллером (от английского controller, т. е. буквально «управитель». В русском языке слово «контроль» имеет иной смысл по сравнению с английским control — управление. Это следует помнить и не считать контроллер контролирующим устройством. Это устройство управления.)

В нашем примере с автоматическим управлением работы двигателя микропроцессор будет работать в режиме контроллера, запрограммированного на вычисление заданных функций f_1 и f_2. И только.

В обычном карманном калькуляторе, выполняющем несколько арифметических и алгебраических функций, тоже используется микропроцессор. Его постоянная память содержит несколько программ (умножения, деления, возведения в степень и т. д.), которые вызываются нажатием соответствующей клавиши на пульте калькулятора.

В современных ЭВМ микропроцессоры нашли очень широкое применение в качестве контроллеров, управляющих работой внешних устройств (дисковой и магнитофонной памятью, печатающим устройством, графопостроителем и т. д.). В последнее время микропроцессоры стали употреблять даже в механических и электронных игрушках, где на них возлагаются функции как контроллеров, так и хранителя различных программ, выполняемых игрушкой. Вызов той или иной программы производится кнопками управления.

Область применения микропроцессоров велика и быстро расширяется. По-видимому, недалек тот день, когда любой прибор, режим работы которого можно изменять, будет содержать микропроцессор. Стоимость его от этого почти не увеличится.

Айболит с компьютером (ЭВМ в процессах принятия решений)

Все помнят доброго доктора Айболита с его градусником и микстурой из доброй сказки К. Чуковского. Говоря современным языком, у Айболита был лишь

один источник (датчик) информации — градусник, и один способ лечения — микстура. А способ (алгоритм) лечения более чем простой: — если температура повышенная (т. е. больше или равна 37°), то давать микстуру, а если меньше, то не давать. Вот и все!

Если это так, то зачем Айболит? Ведь в этом случае его функции могут выполнить не только родители, но и сами дети! К такому выводу неизбежно приходят взрослые, когда подходят со своими «взрослыми» мерками к детским сказкам. Конечно же главное (и это хорошо знает каждый ребенок и забывает взрослый) в Айболите не его градусник и микстура, а то, что он добрый. Именно этим замечателен для детей всякий доктор, а не тем, как он лечит.

Но нам-то, взрослым, приходится сталкиваться с теневыми сторонами жизни, и лечение, к сожалению, является одной из них. Дело в том, что процесс лечения сам по себе связан со всякого рода неприятными ощущениями (горькими лекарствами, уколами и т. п.). Однако не только к этому он сводится.

Всякое лечение довольно четко разделяется на два этапа: диагностику заболевания и собственно лечение, т. е. воздействие на больного всякого рода медицинскими средствами (медикаментозными, хирургическими, физиотерапевтическими и т. д.).

Нас будут интересовать информационные аспекты проблемы лечения: как поставить диагноз и как определить способ лечения, т. е. чем и как лечить (вопросы реализации выбранного способа лечения рассматривать не будем и возложим их решение на медсестер).

Итак, рассмотрим первую проблему — диагностику. Нашему обычному врачу (не Айболиту) здесь приходится трудно. Надо из нескольких тысяч названий болезней, которыми могут болеть люди, для данного пациента выбрать одно и, что самое скверное, надо определить е г о болезнь. Правда статистика и опыт врача, помогают ему в диагностике. Действительно, летом чаще всего к нему обращаются с кишечными заболеваниями, весной и осенью чаще заболевают ОРЗ (острым респираторным заболеванием) или гриппом. Но действовать только в соответствии со статистикой, т. е., например, объявлять больными ОРЗ тех, кто явился осенью и зимой, не будет ни один врач. Он прежде выслушает жалобы больного, выявит симптоматику и

только после этого сделает заключение, чем же заболел больной.

Выглядит это примерно так. Врач прикидывает в уме: повышенная температура, насморк и кашель, да еще в осеннее время, во время эпидемии гриппа — наверняка это грипп. И будет прав не только осенью, но и весной, и при необъявленной эпидемии гриппа. Летом следует проверить живот и спросить: не съел ли чего?

Так врач в зависимости от обстановки в его районе (именно здесь учитывается статистика) проводит обследование больного. Учет статистики очень важен для определения порядка обследования симптомов. Действительно, при эпидемии гриппа прежде всего следует выяснить состояние симптомов, определяющих грипп, а не желудочно-кишечное заболевание. И лишь получив отрицательную симптоматику врач обращается к анализу других симптомов.

Итак, налицо довольно простая и естественная стратегия: сначала искать среди наиболее часто встречающихся болезней, а экзотические заболевания оставить на самый последний этап. При малом числе симптомов (медицинских признаков) реализовать ее просто. А вот что делать, если таких признаков не просто много, а очень много (несколько сотен или даже тысяч!), как, например, бывает при диагностике сердечно-сосудистых заболеваний или рака! Да при этом за выявление симптома надо платить: временем при лабораторных анализах, дорогостоящими материалами при рентгенографии и т. д., не говоря уже о том, что расходуется время высококвалифицированных узких специалистов — рентгенолога, аллерголога, невропатолога и т. д. Как же при этом вести обследование больного, чтобы, во-первых, надежно выявить его заболевание и, во-вторых, затратить на это минимальные средства (временны́е, материальные и т. д.).

Любому врачу приходится каждый раз решать эту задачу. Как он справляется с ней? Не плохо, если случай простой и есть опыт, в котором отражены сведения о наиболее распространенных заболеваниях и их симптоматике. Если опыта нет, как у молодого врача, то затраты будут больше, чем это необходимо (в этом и сказывается отсутствие опыта).

Если случай не простой, то даже опытному врачу приходится туго. И вынужден он обращаться в ЭВМ.

Делает это врач с большим неудовольствием. И его можно понять, так как основная врачебная заповедь гласит, что лечить надо не болезнь, а больного, т. е. человека. А здесь в этот очень человеческий процесс общения между больным и врачом вклинивается бездушная машина. Что она может знать о человеке? Не слишком ли обнахалились инженеры, предлагая использовать ЭВМ в процессе лечения человека?

Чтобы разобраться, надо напомнить, что процесс лечения, как и всякий другой процесс, всегда можно разложить на две составляющие — формализуемую и неформализуемую. Формализуемую, т. е. ту, которую можно описать формальным, например, математическим языком, естественно можно передать ЭВМ. А неформализуемую (точнее: пока неформализуемую) компоненту процесса лечения пусть реализует врач. Заметим, что полной формализации лечения достигнуть никогда не удастся (и слава богу!) хотя бы потому, что психологический контакт врача и больного тоже образует важный компонент лечения. А его формализовать можно только создав робота, внешне и поведением похожего на врача. Такую формализацию всерьез можно воспринимать лишь в фантастике.

Итак, врач, хочет он или не хочет этого, обязан заниматься диагностикой заболевания, которая довольно легко поддается формализации. Покажем это.

Состояние больного (его анкету) можно описать набором нулей и единиц. Представим анкету в виде ряда:

$$x_1, x_2, \ldots, x_n,$$

где x_i — i-й симптом, т. е. значение i-го признака. Если этот симптом имеется у больного, то $x_i = 1$, и $x_i = 0$ в противном случае. Например, x_1 — симптом повышенной температуры, x_2 — симптом больного горла, x_3 — симптом озноба и т. д. Тогда больной с повышенной температурой и ознобом, но не жалующийся на горло, будет иметь анкету в виде 101.

Конечно для хорошей диагностики нужно иметь подробную анкету, содержащую много симптомов. Такую анкету легко ввести в память ЭВМ, которая сравнит эту анкету с эталонными анкетами возможных заболеваний. Таких эталонных анкет можно заготовить много на основе наблюдений за проявлением заболеваний в данной местности (известно, что в разных местах люди болеют по-разному). Очевидно, что анкета больного должна быть квалифицирована по ее близости к

эталонным анкетам. Та эталонная анкета, на которую более всего «похожа» анкета больного, и определяет его предполагаемое заболевание.

Степень «похожести» анкет определить легко, например, по числу совпадающих симптомов. Чем больше таких совпадений, тем вероятней, что пациент болен именно этой болезнью. Окончательное решение принимает врач — ведь ему доступна информация, которой не располагает ЭВМ (например, выражение глаз пациента, его внешний вид и т. д.). Эта информация часто бывает определяющей. Говорят, что старые земские врачи, пользуясь только часами (для измерения пульса) и перкуссией (простукиванием), ставили очень надежный диагноз, в котором неформализуемые симптомы играли определяющую роль: «Что-то, батенька, не нравится мне ваш вид — взгляд какой-то тусклый. Не переели ли вы чего ненароком? В вашем возрасте нужно быть аккуратней! Давайте-ка промоем кишечник». И пациент получает заслуженный клистир.

Но вернемся к ЭВМ. Эталонную анкету болезни часто составить довольно трудно. Тогда можно эталон заменить несколькими анкетами больных, относительно заболевания которых у врачей не было сомнений (такие случаи врачи называют верифицированными). Верифицированные анкеты и играют роль нескольких эталонов одной и той же болезни. Другие болезни будут представлены другими верифицированными анкетами больных.

Теперь анкета пациента, которого надо диагностировать, сравнивается ЭВМ со всеми верифицированными анкетами, хранящимися в ее памяти. Самая похожая анкета и поможет врачу определить диагноз больного.

Такой подход не нов для медицинской диагностики. Он реализует формально старую-престарую идею аналогичных случаев, о которых любят поговорить старые врачи (их обычно называют частичными или полными прецедентами). «Помню, как в тысяча девятьсот (таком-то) году был у меня больной...» Далее описывается его симптоматика, близкая к анализируемому случаю. «Так вот, представьте, батенька, мне пришлось много повозиться с ним. Чего я только не пробовал! А оказалось, это была самая тривиальная (называется болезнь). Сейчас о ней забыли, а тогда... Так что прикинь на него, голубчик, эту версию. Вполне не исключено, что это то и есть».

ЭВМ у нас, как видно, выполняет роль старого опытного врача, в памяти которого хранятся верифицированные прецеденты, столь важные для правильной диагностики больного. Преимущества ЭВМ здесь очевидны: она может оперировать не сотнями случаев, составляющих опыт одного старого опытного врача, а тысячами и даже миллионами. При этом она не подвержена склерозу и находит аналогичные случаи очень быстро.

Как видно, и ЭВМ, и опытный врач ищут в своей памяти аналогичные случаи по числу совпадающих симптомов. Но симптом симптому — рознь! Один почти не несет информации о заболевании, как, например, симптом повышенной температуры (вспомним хотя бы злополучный бестемпературный грипп). А другой обладает огромной информативной силой при диагностике, как, например, рентген или анализ крови. Именно поэтому, определяя степень близости анкет, нужно суммировать совпадающие симптомы, умножая их на определенные числа, характеризующие информативность каждого из совпадающих симптомов (эти числа обычно называют «весами»). Чем более информативен симптом, тем больше он имеет вес.

В результате может оказаться, что для хорошей, т. е. надежной, диагностики достаточно определить один — два высокоинформативных симптома и не обращаться к десятку — другому малоинформативных. Однако было бы неправильно так поступать, так как здесь не учтены факторы стоимости диагностики и ущерба, наносимого больному.

Действительно, проверка наличия или отсутствия каждого симптома связана с определенными затратами (об этом уже говорилось выше). Кроме того, выявление симптоматики часто наносит пациенту прямой вред в виде неприятных и болезненных ощущений. Вспомним хотя бы процедуру сдачи крови на анализы — восхитительной ее не назовешь. А информации о заболевании анализ крови дает предостаточно.

Другой пример. Различить некоторые неврологические заболевания довольно просто путем анализа спинномозговой жидкости. Однако процедура ее извлечения из спинного мозга, называемая пункцией, является весьма неприятной, а иногда и небезопасной для больного процедурой. Вот и получается, что чем больше

информации дает какой-то способ выявления симптомов, тем он «дороже» или «неприятней». Этому досадному свойству информации не следует удивляться. За нее надо платить тем больше, чем она важнее. Вот и приходится расплачиваться затраченным временем и неприятными ощущениями в диагностических кабинетах. Но это необходимо, так как только таким способом будет получена информация, нужная врачу (и ЭВМ) для надежной диагностики заболевания.

Но если приходится терпеть неприятности, то, разумеется, хотелось бы иметь уверенность, что они минимальны. Старая заповедь «из двух бед выбирают меньшую» во многом, если не во всем, определяет поведение человека. И правильно определяет!

Как же минимизировать неприятности при диагностике? Эту задачу можно решать только когда известно, какую информацию дает и какой ущерб наносит выявление того или иного симптома. В этом случае задачу диагностики человек решает плохо, так как надо очень много вычислять. И чем больше возможностей предоставляет медицинская диагностическая техника, тем в более трудные условия попадает врач. Слишком много числовой информации ему приходится перерабатывать. ЭВМ здесь становится незаменимым помощником врача, так как она создана для быстрой и безошибочной переработки информации.

Признаемся, что таких ЭВМ — помощников врача в здравоохранении пока не много. Аналогичные есть в Москве (клиника имени Вишневского), в Каунасе (кардиологический центр), в Братиславе (клиника нервных болезней) и др. Поэтому легко представить, как ЭВМ будет использована в процессе такой диагностики.

Представьте, что вы пришли со своими заботами в поликлинику. Прежде всего вас направят в доврачебный кабинет, где медсестра, сидя за небольшим пультом терминала (он соединен с ЭВМ), выслушает ваши жалобы и введет их в ЭВМ. Кроме того, она (медсестра) сделает простейшие измерения (температуры, пульса, давления и т. д.), а результаты введет в ЭВМ с помощью того же терминала. И направит вас к врачу.

Врач, получив от ЭВМ все данные о вашем теперешнем состоянии (их только что ввела медсестра) осмотрит вас и... может поступить трояким образом:

1. Поставить диагноз (случай тривиальный и диагностика не представляет осложнений).

2. Затребовать историю болезни, которая ведется все той же ЭВМ. На экран видеотерминала врачу будут выдаваться по его команде страницы вашей истории болезни, где в хронологическом порядке собрана информация о ваших прошлых заболеваниях. На последней странице будут приведены данные, только что полученные в доврачебном кабинете. Врач может задать ЭВМ различные вопросы по поводу истории болезни. Например, «когда болел гриппом и ОРЗ?», на что ЭВМ укажет даты всех заболеваний гриппом и ОРЗ, отмеченных в вашей истории болезни. Эта информация поможет врачу понять специфику пациента и принять решение о диагнозе.

3. И наконец, врач может затребовать мнение ЭВМ по поводу заболевания. Так как ваша анкета, составленная медсестрой в доврачебном кабинете весьма неполна, то таким же приближенным будет и ответ ЭВМ. Она укажет несколько наиболее вероятных заболеваний, вычислив их по эталонным анкетам, хранящимся в ее памяти. При этом ЭВМ может указать вероятность каждой из указанных болезней, например, так:

ГРИПП — 0,31; ОРЗ — 0,23; ..., АЛЛЕРГИЧ. НАСМОРК — 0,01, где многоточием обозначены другие болезни с вероятностью меньше, чем 0,23; и больше 0,01.

Указанные вероятности вычисляются ЭВМ на основе вашей анкеты и статистики заболеваемости этими болезнями в вашем районе и в это время года. Статистика такого рода имеется в памяти ЭВМ и влияет на диагностику вашего заболевания (выше об этом уже говорилось).

Врач из списка отбирает те, которые могут быть в данном случае (точнее: отбрасывает заведомо невозможные, например, аллергический насморк и т. д.). Для того, чтобы надежно различить одно из оставшихся заболеваний, нужно получить дополнительную информацию, т. е. произвести какие-то анализы. И перед ЭВМ ставится задача указать симптомы, которые после их выявления позволили бы сделать надежный диагноз (например, с вероятностью 0,99), нанося минимальный ущерб больному, и не затратить при этом значительных средств. Эту трудную вычислительную задачу ЭВМ решает и указывает на экране видеотерминала:

1. АНАЛИЗ КРОВИ (ОБЩИЙ)
2. РЕНТГЕН ЛЕГКИХ
3. АНАЛИЗ МОЧИ (ОБЩИЙ).

Врачу, если он согласен с ЭВМ, надо выписать соответствующие направления. В этом ему поможет электрическая пишущая машинка, связанная с ЭВМ. Направления с указанием вашей фамилии, номера кабинета, даты и времени приема будут напечатаны сразу. Врачу останется лишь подписать их.

Когда вы, сделав необходимые анализы, снова придете к врачу, то по его запросу ЭВМ на экране видеотерминала укажет возможные ваши заболевания с новыми оценками их вероятностей. Результаты после последних анализов, записанные на последней странице вашей истории болезни, также будут выданы врачу по первому его требованию.

Если максимальная вероятность невелика и сомнения врача не дают ему возможности принять решение о диагнозе, то он снова запрашивает ЭВМ, какие исследования надо сделать, чтобы получить надежный диагноз и при этом минимизировать ущерб, наносимый поликлинике и пациенту. И так далее до тех пор, пока не будет поставлен правильный диагноз.

Обратите внимание, что в общении с ЭВМ решение всегда принимает врач, а ЭВМ лишь предлагает варианты возможных решений с оценкой их вероятной реализации для данного больного или варианты исследований, которые позволят быстро и надежно диагностировать данного больного, причем доставляют ему при этом минимальные неприятности.

Как видно, очень нужна современному Айболиту ЭВМ на стадии диагностики заболевания. Нужна она и пациенту.

Так ЭВМ помогает преодолевать трудности диагностики. За диагностикой следует собственно лечение, т. е. назначение медицинских мероприятий, призванных вылечить больного, а точнее: сделать его «практически здоровым». В это понятие медики вкладывают свое представление о здоровом человеке, которое может отличаться от понятия «здоровый». Добавка «практически» позволяет понимать норму, а, следовательно, и свою цель достаточно широко, что неизбежно в процессах лечения, которые имеют дело, пожалуй, с самым сложным и самым важным объектом — человеческим организмом.

В распоряжении врача имеются многообразные лечебные средства, предоставляемые современной медицинской наукой и техникой. Пожалуй, даже слишком

много средств, особенно фармакологических. Фармакологические фирмы всего мира ежегодно выбрасывают на рынок огромное количество новых лекарственных препаратов. Каждый из них снабжен подробным перечнем показаний и противопоказаний, т. е. в каких случаях его следует, а в каких не следует его применять. Все это обязан знать врач для эффективного назначения.

Однако, как показывает опыт, нет лекарств, одинаково действующих на всех больных, и нет единой дозировки препарата. Более того, один и тот же больной в разные периоды заболевания нуждается в разных дозах одного и того же лекарства.

Все трудности лечения больного связаны с трудностями учета его индивидуальных свойств: симптомов, восприимчивости к лекарствам, перенесенных заболеваний, психического состояния, наследственности и многих, многих других. Из этого длинного списка для простоты возьмем лишь симптоматику больного.

Симптоматика определяет заболевания и она же в конечном счете определяет лечение. Оно опирается на известный врачебный афоризм: для устранения болезни достаточно устранить симптомы. А если список симптомов расширить диагнозами заболеваний пациента, то борьба с ним (т. е. списком его недугов) и есть лечение больного. (Мы здесь, естественно, намеренно упрощаем и огрубляем схему лечения, да простят нас врачи).

Всякое лекарство обладает тремя свойствами: оно лечит какие-то определенные недуги, безразлично к другим, но противопоказано при третьих. Кроме того, оно может влиять и на отдельные здоровые органы больного, причем — негативно.

Врач, имея симптоматику больного (т. е. список его недугов), должен из всего множества доступных лекарств выбрать те, которые лечат имеющиеся недуги и не вызывают новых. Задача эта, скажем прямо, не из легких. Врач должен помнить, и помнить очень хорошо, показания и противопоказания всех доступных лекарств. Он не имеет права забыть что-либо. Но ведь он всего лишь только человек! И забывчивость ему так же свойственна, как и необходима, что отличает всякого нормального человека. Как же быть?

Обратиться к ЭВМ, которая уже неоднократно выручала. Сделать это можно так.

Таблица 1

Недуг	Лекарство				
	1	2	. . .	$m-1$	m
1	a_{11}	a_{12}	. . .	$a_{1,m-1}$	a_{1m}
2	a_{21}	a_{22}	. . .	$a_{2,m-1}$	a_{2m}
. . .	. . .	. . .	. . .	. . .	. . .
$n-1$	$a_{n-1,1}$	$a_{n-1,2}$	. . .	$a_{n-1,m-1}$	$a_{n-1,m}$
n	a_{n1}	a_{n2}	. . .	$a_{n,m-1}$	a_{nm}

Давайте рассмотрим табл. 1.

Здесь по вертикали выписаны все недуги, из которых образуется симптоматика больного (для простоты приведены не названия симптомов и заболеваний, а их номера от 1 до n). По горизонтали приведены все известные препараты (для простоты также вместо их названий приведены номера от 1 до m).

Числа, вписанные в таблицу, указывают характер влияния какого-то препарата на определенный недуг. Это влияние может быть положительным, отрицательным и безразличным. Так для i-го недуга и j-го препарата на пересечении i-й строки и j-го столбца стоит число a_{ij}. Оно равно $+1$, если j-й препарат оказывает полезный эффект на i-й недуг, что означает убывание интенсивности патологического признака, отраженного в этом недуге, при приеме указанного препарата. $a_{ij}=-1$, если имеет место отрицательное влияние (усиление степени выраженности недуга или появление другого, побочного). И, наконец, $a_{ij}=0$, если j-й препарат никак не влияет на i-й недуг.

Как видно, таблица состоит из $+1$, 0 и -1. Составляется она на основе описаний действия препаратов и их применения в лечебной практике. Легко заметить, что нулей в таблице будет значительно больше, чем единиц. Это происходит потому, что всякое лекарство изготовляется для лечения определенных недугов. К остальным оно должно быть, по меньшей мере, безразлично — иначе такое лекарство фармакологический комитет министерства здравоохранения не утвердит к мас-

совому изготовлению (если, конечно, препарат лечит тяжелый недуг ценой легких недомоганий — например, некоторые противораковые средства вызывают тошноту, то его применение оправдано). Вот и получается, что таблица эта почти пустая. Поэтому ее удобно разбить на несколько таблиц по группам заболеваний, которым будут соответствовать группы лекарств. Каждая из таких таблиц достаточно велика, но вполне размещается в памяти современной ЭВМ. (Если бы эти таблицы нарисовать на бумаге, то их нельзя было бы развернуть в обычной комнате — нужен по крайней мере зал. Значения m и n для этих таблиц имеют порядок $10^3 - 10^4$.)

Работа с таблицами достаточно проста, хотя и громоздка. Врач определяет симптоматику больного и вводит ее в ЭВМ в виде последовательности чисел — номеров симптомов (недугов). Машина просматривает именно эти строки таблицы и фиксирует номера тех лекарств, которые могут помочь больному, т. е. те, где стоит $+1$. Врач получает таким образом фрагмент таблицы, где по вертикали располагается симптоматика больного, а по горизонтали список нужных препаратов (их обычно бывает много). На пересечении строк-симптомов и столбцов-препаратов будет напечатана одна из двух цифр: нуль или $+1$, характеризующие «взаимоотношения» препарата и симптома.

Кроме того, для каждого препарата будет приведен итог — сумма всех цифр этой строки в таблице. Чем больше итог, тем выгоднее применение лекарства к данному больному, так как оно устраняет сразу несколько симптомов.

Кроме этого, ЭВМ выдает врачу список нерекомендуемых препаратов, то есть фрагмент хранящейся в памяти ЭВМ таблицы, где для заданной симптоматики имеются отрицательные единицы (-1). Эта таблица служит предостережением врачу на случай, если он захочет отклониться от первой таблицы рекомендуемых препаратов и предложить больному другие. Такие соображения возникают всегда у творческого врача, который ищет более тонких путей лечения. В этом случае перед тем, как прописать препарат, не вошедший в первый список, врач может убедиться в его непротивопоказанности больному, использовав второй список. Если же в этом списке он все-таки обнаружит это лекарство, то ему придется крепко задуматься над тем, давать

ли его. А если давать, то чем следует каким-то образом компенсировать негативное действие этого препарата.

Так между врачом и ЭВМ возникает диалог, в процессе которого врач принимает решение о лекарственном воздействии на больного.

Совершенно аналогично врач может выбирать для больного средства физиотерапии (кварц, УВЧ, ванны и т. д.), лечебные воды, климатические и другие воздействия. Важно то, что врачу не нужно механически помнить все показания и противопоказания любого способа лечения. За него с этим великолепно справляется ЭВМ. Его задача — принять решение, совместимое со списком, предложенным ЭВМ. А если решение врача расходится с решением ЭВМ, то это обстоятельство не может быть случайным (например, результатом забывчивости), ЭВМ не допустит такой досадной случайности.

Заметим, что различия во мнениях врача и ЭВМ скорее следует считать закономерностью, чем досадной «оплошностью» ЭВМ. Дело в том, что врач, выбирая средства лечения, располагает значительно большей информацией о больном, чем ЭВМ. Именно избыток информации и позволяет врачу предложить препарат из тех, которые не вошли в список рекомендованных ЭВМ. И наоборот, использовать нерекомендованный. Плох был бы врач, если бы он во всем слушал ЭВМ! Она нужна ему как справочная, которая помогает ему не делать грубые промахи. Именно ЭВМ позволит врачу полностью заниматься лечением больного и не забивать себе голову огромным числом фактов (прецедентов, симптомов, препаратов и т. д.). Пусть это делает ЭВМ с ее большой и надежной памятью. И пусть она станет настоящим помощником врача и избавит его от многих второстепенных забот, которым врачу, к сожалению, сейчас еще приходится уделять слишком много времени, отрывая его от больного.

Очень нужна ЭВМ современному Айболиту. Очень!

Глава 2
ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

Чем плоха ЭВМ?

ЭВМ как электронный автомат для обработки информации представляет собой четырехкомпонентную машину. Она состоит из:

— процессора, выполняющего заданные программой операции переработки информации (он содержит арифметическое устройство, которое производит вычисления, и управляющее устройство, осуществляющее управление процессом вычислений: оно указывает, какие операции должно делать арифметическое устройство);

— оперативной памяти, где хранится выполняемая в данный момент программа, исходные данные для нее и все необходимые для этого вспомогательные программные средства;

— внешней памяти (это магнитные диски и ленты), где хранятся файлы пользователей и другая информация (в основном справочная);

— устройств ввода-вывода, с помощью которых информация вводится в ЭВМ (через перфокарты, перфоленты или клавиатуру терминала) и выводится на бумагу или экран дисплея, также на те же перфокарты и перфоленты.

Эти четыре элемента имеет любая ЭВМ, как бы и кем бы она ни была создана. И маленькие настольные мини-ЭВМ, размер которых не превышает пишущей машинки, и похожие на шкаф гиганты — макси-ЭВМ (десять-пятнадцать лет тому назад — до появления интегральных схем — они занимали большие залы) — все эти ЭВМ состоят из процессора, внешней и оперативной памяти, а также устройств ввода-вывода.

С чем связано удивительное постоянство структуры ЭВМ? Неужели фантазия создателей не пошла дальше? Разве не нужно было придумать что-то еще? Почему столь многообразные потребности в автоматической обработке информации привели к столь уныло-одинаковой

структуре ЭВМ — машин для удовлетворения этих потребностей?

Как ни парадоксально, но именно многообразие потребностей привело к однообразной структуре машин! Будь потребностей поменьше, наверное различных структур ЭВМ было бы больше. Действительно, представьте, что было бы только две потребности: одна — решать задачи, используя только две операции — сложения и вычитания, а другая — операции умножения и деления. Легко видеть, что надо было бы создать лишь два типа машин — для сложения — вычитания и умножения — деления. Ввиду узкой специализации машины отличались бы не только конструкцией, но и структурой (ведь именно в структуре отражается ее самая сокровенная специфика).

Современные ЭВМ решают задачи из тысяч областей человеческой практики, каждая из которых порождает сотни проблем, причем каждая проблема требует решения десятков задач. Опыт и практика решения задач и привели к описанной выше четырехкомпонентной структуре современной ЭВМ. Именно эта структура наилучшим образом в среднем решает все задачи.

«В среднем» — значит, что для каждой задачи можно было бы придумать структуру ЭВМ и получше, т. е. попроще или подешевле или понадежнее или побыстрее решающей эту самую задачу. Решая ее на стандартной ЭВМ (будем для простоты так называть ЭВМ, имеющую указанную четырехкомпонентную структуру), мы будем терпеть некоторый ущерб по сравнению с оптимальной ЭВМ, созданной именно для этой задачи. Но суммарный ущерб, который приходится терпеть на всех задачах, при этом будет минимальным. Это означает, что структура стандартной ЭВМ оптимальна для всего множества решаемых задач, хотя для каждой в отдельности она и далека от совершенства. (Так, автомобиль «Жигули» устраивает огромную массу автолюбителей несмотря на огромное разнообразие их индивидуальных потребностей). Учет специфики задач при этом производится путем всякого рода конструктивных ухищрений, отличающих одну ЭВМ от другой, спроектированной для решения другого круга задач.

Очевидно, что оптимальная «в среднем» схема очень расточительна, так как она по сути почти никогда не

бывает оптимальной для каждой конкретной ситуации, сложившейся при обработке информации. Эти соображения и заставили создавать вычислительные системы. Что же это такое?

Основные черты вычислительной системы

Если относительно ЭВМ можно сказать, что это машина для обработки информации, то сказать такое же относительно вычислительной системы, значит не сказать ничего. Каждая вычислительная система создана для определенного способа ее использования. Пытаться ее использовать по-другому крайне невыгодно, а зачастую и просто невозможно — таков естественный эффект специализации.

Говоря об обработке информации вычислительной системой, необходимо указать, о какой информации идет речь (технической, научной, экономической и т. д.) и какая именно обработка реализуется системой. Более того, необходимо указать, на какой круг пользователей ориентирована вычислительная система и каковы цели ее функционирования. Именно эти и другие факторы специализируют вычислительную систему, хотя при этом она и останется универсальным средством обработки информации, как ЭВМ.

Трудно указать формальное отличие вычислительной системы и ЭВМ. Вычислительную систему обычно отличает не только специализация и большая связь со средой, поставляющей ей задачи. Вычислительные возможности систем всегда превышают возможности обычных ЭВМ. Так, очень часто в вычислительной системе используют не один процессор, а несколько, образуя тем самым многопроцессорную систему.

Как видно, вычислительные системы являются следующим шагом вперед по сравнению с ЭВМ. Они появились благодаря тому, что сильно возросли и специализировались потребности в обработке различного рода информации. Рассмотрим основные типы вычислительных систем.

Вычислительные системы коллективного пользования

Эти системы, сейчас, пожалуй, наиболее популярны, так как они позволяют преодолевать основной недостаток больших ЭВМ — их малодоступность. Действительно,

выход к мощной ЭВМ всегда связан с необходимостью если не прямого, то, во всяком случае, достаточно близкого контакта с ней. Нужно отперфорировать программу и данные (на перфокарты или перфоленту) и сдать ее оператору машины, который из этих отдельных программ, полученных от разных пользователей, составит пакет и пропустит его. Ответ в самом лучшем случае вы получите к вечеру, если сдали колоду перфокарт утром. Если программа отлажена, то это не так страшно, хотя ждать ответа целый день от машины, делающей миллион операций в секунду, нелогично и как-то обидно!

Но еще обидней, если вы отлаживаете программу. Известно, что в процессе отладки необходимо часто выходить на машину, причем ответ нужно получить быстро. А именно этого не позволяет такая организация работы с ЭВМ.

Вот и приходится месяцами отлаживать сравнительно простые программы.

Именно этот недостаток больших ЭВМ и устраняют системы коллективного пользования. Основная идея системы заключается в том, чтобы дать возможность многим пользователям быстро выходить на ЭВМ и оперативно получать результат прогонки своей программы или ответы на поставленные вопросы.

Сейчас в мире существует большое число вычислительных систем коллективного пользования с самыми разнообразными терминальными устройствами от электрической пишущей машинки до многомашинной терминальной системы, которая в состоянии сама решать довольно сложные задачи по переработке информации пользователя. Все эти терминалы отличаются тем, что подключены каналами связи к ЭВМ или системе, имеющей большую вычислительную мощность и значительную память. Протяженность каналов связи в такой системе может быть тоже самой разнообразной — от нескольких метров до тысяч километров.

Примером такой разветвленной вычислительной системы, имеющей терминалы почти во всех городах США, является NASDAQ. Эта система обслуживает пользователей финансовой информацией (курсы акций и проч.), оперативно реагируя на все биржевые изменения. Основная вычислительная и информационная мощность этой системы сосредоточена в г. Трамбулле, где стоит мощная машина Univac-1108. Терминалы же

позволяют обращаться к этой машине и информируют ее о состоянии курсов акций «на местах». Их 1700 штук.

Это помощь рядовому пользователю ЭВМ. Но системы коллективного пользования оказывают помощь и администраторам, избавляя их от необходимости заводить в своем учреждении вычислительный центр со всеми вытекающими отсюда организационными, штатными, финансовыми и другими последствиями. Достаточно стать абонентом системы коллективного пользования, которая, как всякое удачное решение, разрешает сразу несколько проблем. Рассмотрим такую систему подробнее.

Система коллективного пользования представляет собой мощную ЭВМ, которая через каналы связи соединяется с большим числом пользовательских терминалов — их может быть 50—200 штук и более — число их принципиально ничем не ограничено. Каждый терминал представляет собой несложное устройство для введения программы в ЭВМ и приема от нее результатов. Состоит оно из электрической пишущей машинки, которая одновременно является перфоратором для бумажной ленты. Печатая свою программу на такой машинке, вы одновременно получаете перфоленту с программой. Ленту вставляете в устройство считывания, которое с большой скоростью (1500 знаков в секунду) передает программу в ЭВМ по каналу связи.

Здесь она принимается и ставится в очередь, если таковая имеется, к процессору, а освободившийся процессор выполняет программу. Результаты передаются тут же по тому же каналу связи в перфорирующее устройство, которое со скоростью 100 знаков в секунду перфорирует бумажную ленту. Вставив ленту в пишущую машинку, получаете распечатку результата работы ЭВМ по вашей программе на рулоне ленты.

Итак, в комплект простейшего терминала входят:

— электрическая пишущая машинка с перфорацией ленты и управляемая перфорированной лентой,

— считывающее устройство с перфоленты и

— перфоратор результатов на бумажную ленту.

Более сложные (и, соответственно, более дорогие) комплекты содержат не пишущую машинку, а ЦПУ — цифровое печатающее устройство, способное печатать со скоростью 100 знаков в секунду и более. А еще более совершенный терминал называют интеллигентным. Это

по сути дела маленькая вычислительная машина со своей памятью на гибком магнитном диске, где хранится программа пользователя перед отправкой на большую ЭВМ, и свой процессор, решающий задачи по редактированию составляемой пользователем программы, и позволяющий делать несложные вычисления. Как известно, пределов совершенства нет, нет этого предела и у терминальной техники, которая становится все совершенней и, соответственно, сложней и... дешевле. Это проявление одной любопытной закономерности: каждая новая модель вычислительной техники (ЭВМ или отдельных ее устройств) имеет улучшенные характеристики и меньшую стоимость по сравнению с предыдущими моделями. Эта закономерность и способствует широкому распространению и внедрению ЭВМ в нашу жизнь.

Если терминал расположен достаточно далеко от ЭВМ, то его следует дополнить устройством связи с ЭВМ. Устройство кодирует специальным образом посылаемую программу так, чтобы помехи, которые неизбежны в длинных каналах связи, не исказили бы передаваемую и принимаемую информацию.

Располагая таким терминалом, пользователь по праву может считать, что он сидит за пультом ЭВМ, к которой он присоединен и «хозяйничает» с ней как хочет. Действительно, ЭВМ выполняет его программу и все распоряжения этой программы. Что же еще? Вот только другие пользователи...

С ними вы сталкиваетесь, ожидая очереди к процессору ЭВМ. Легко видеть, что большое число таких пользователей заставит вас долго стоять в очереди, дожидаясь, когда процессор освободится для вашей программы. У этой очереди будут свои часы «пик» — в рабочее время, когда она будет максимальной, и свои провалы — ночью, когда никто не хочет иметь дело с ЭВМ и ее процессор будет простаивать.

Неудобство сложившейся ситуации очевидно. Хотя это немного лучше, чем «старый» способ, когда пользователь сам приносит свою программу оператору ЭВМ. Но по-прежнему ответ на короткий вопрос, заданный ЭВМ в «неудобное» время, придется ждать долго. В очереди.

Неудобство очереди состоит не только в том, что приходится просто ждать, а в том, что ждать приходится зря. Дело в том, что пользовательские программы,

стоящие в очереди, обладают различной степенью «нетерпения». Самые нетерпеливые — короткие программы. Это либо простенькие задачки, либо куски большой программы в стадии отладки. В обоих случаях пользователь ждет быстрого ответа.

Если же программа велика, то пользователь психологически уже подготовлен к ожиданию, даже при ее отладке. Поэтому большие программы, а точнее — программы, требующие для своего счета много времени, нельзя ставить в одну очередь с короткими, так как они неравноправны.

Можно было бы сформировать очередь по длине программы: чем длиннее программа, тем ближе к хвосту. Тогда короткие программы будут обрабатываться очень быстро за счет задержки длинных. Но когда очередь все-таки дойдет до большой программы, требующей, например, часа процессорного времени, нетерпеливые пользователи будут раздражены. Ведь им придется ждать не менее часа. Другим — тоже. При этом очередь очень вырастет за счет поступающих программ.

Для того, чтобы преодолеть этот недостаток систем коллективного пользования, был придуман остроумный прием, названный режимом разделения времени. Состоит он в следующем.

Время работы процессора разбито на кванты времени величиной Δ ($\Delta \approx 0{,}1$ с). В течение одного кванта процессор решает только одну задачу. Если эта задача не велика ($\sim 100\,000$ операторов), то она будет решена за один квант времени и результаты отправляются пользователю, а процессор начинает работать со следующей по очереди задачей. Если же задача велика и за один квант времени решить ее не удается, то решение ее прерывается и она направляется в конец очереди. Когда придет ее черед, процессор продолжит решение с того места, на котором он остановился при прошлом обращении к этой программе (о чем позаботится операционная система ЭВМ).

При таком способе короткой программе в любом случае придется ждать не больше, чем $n\Delta$ секунд, где n — число программ в очереди. При $n = 100$ (в часы пик) ждать придется примерно 10 с, что для диалогового режима недопустимо (напомним, что в диалоге с ЭВМ максимальное время отклика должно быть не более двух секунд, иначе пользователь начинает нервничать!).

Это затруднение можно преодолеть, если при организации очереди отказаться от принципа «пришедший должен становиться в хвост». Этот принцип демократичен и широко используется в человеческой практике. Но, что хорошо для людей, не всегда хорошо для машин. (Еще один пример, как принципы, выработанные обществом, не распространяются на технику.)

Действительно, будем строить очередь (вопреки «здравому смыслу») в обратном порядке: все поступающие с терминалов заявки ставить не в конец, а в начало очереди (если их много, то образуется известная «очередь без очереди»). А незавершенные программы по-старому ставить в конец этой очереди. Такая очередь, организованная по принципу «последний пришел — первым обслужен», называется «стеком».

В этом случае, как легко видеть, коротким программам не придется ждать и ответ вычислительной системы будет практически мгновенным. Но это получено за счет того, что большие задачи будут решаться дольше; что впрочем не слишком опечалит их хозяев — они к этому готовы.

Так что, дорогой читатель, если вам придется ждать решения своей большой задачи дольше, чем вы рассчитывали, то утешайте себя тем, что в другой раз вы получите ответ раньше рассчитанного, если, разумеется, задача будет меньше.

Такое перераспределение времени обработки происходит в соответствии с известной и естественной закономерностью, подмеченной Л. Клейнроком, что среднее время обслуживания заявок в очереди не зависит от дисциплины обслуживания, то есть от того, в каком порядке будут поступать заявки из очереди в обслуживающий прибор. Действительно, пропускная способность вычислительной системы зависит только от производительности процессора и сокращение времени простаивания в очереди одного класса заявок неизбежно увеличит простой остальных.

Описанная выше дисциплина обслуживания отдает предпочтение коротким заявкам и производит это за счет длинных, чтобы в соответствии с указанной закономерностью сохранить постоянным среднее время. Таким образом оказывается сильное предпочтение задачам, требующим малого времени для своего решения (в пределах одного выделенного кванта времени). Остальные же задачи (и средние и большие) находят-

ся в одинаковых условиях: они решаются квантами, следующими через промежутки времени, равные времени простоя в очереди к процессору. Очевидно, что такая дисциплина обслуживания, безусловно ориентирована на короткие задачи, «несправедлива» к средним, которые должны иметь преимущество по отношению к большим задачам.

Эту «несправедливость» можно устранить с помощью следующей процедуры. Пусть ведется не одна, а много очередей задач, причем эти очереди неравноправны. Перенумеруем их цифрами 1, 2, ..., m, ...

Очередь с меньшим номером имеет абсолютный приоритет перед очередью с бо́льшим номером. Это проявляется в том, что процессор обращается к задаче из m-й очереди только в тех случаях, когда в первых $m-1$ очередях нет задач, т. е. очереди 1, 2, ..., $m-1$ пусты.

Образуются очереди следующим образом. В первую попадают все вновь прибывшие задачи-заявки. Они должны получить по своему кванту времени процессора прежде всего (в оптимальности такого правила мы убедились). Задача, не решенная за один квант времени, поступает в конец второй очереди (№ 2). Если задача из второй очереди не была решена при втором прохождении через процессор, то она ставится в конец очереди № 3. И так далее.

Как видно, нерешенная задача с каждым прохождением процессора переходит в очередь со следующим номером, к которой процессор обращается реже, чем к предыдущей. К большой задаче, таким образом, процессор обращается все реже и реже.

Вот и получается, что большие задачи в результате решаются еще дольше, а малые, благодаря указанной выше закономерности, должны решаться быстрее, так как среднее время обработки задач должно остаться постоянным. Число очередей при этом определяется числом квантов времени процессора, за которое решается самая большая задача. Причем многие из очередей будут пустые.

Отметим, что никаких физических очередей в системе при этом не бывает. Просто задача располагается в определенном месте оперативной или дисковой памяти и ей присваиваются два имени в виде двух чисел — номера очереди и ее номера в этой очереди. Операционная система изменяет эти номера в соответствии

с указанной дисциплиной обслуживания и каждый раз выбирает задачу с минимальными номерами (очереди и номера в очереди) для передачи ее на обработку процессору в очередной квант времени. Если она будет решена за этот квант, то ее место займет другая, новая задача. А если не будет решена, то номер очереди увеличится на единицу и номер в этой очереди ей будет дан максимальный.

Как видно, система коллективного пользования, работающая в режиме разделения времени, является чрезвычайно удобным средством предоставления вычислительных услуг широкому кругу самых разнообразных пользователей.

Любопытно, что такая система может осуществлять и ряд других услуг невычислительного свойства. Например, она выполняет роль электронной почты, точнее — телеграфа. Вы можете послать письмо любому пользователю на другой терминал. Это может быть, например, поздравление с днем рождения, сообщение о получении новой программы, которая интересует адресата, или сама программа. Делается это следующим образом.

Пусть вам необходимо что-либо сообщить своему приятелю по поводу его программы «НАФ-НАФ», которая у вас не идет, и получении новой, которая его интересует. Сообщение может быть, например, следующее:

412В [СТАРИК, ТВОЙ НАФ-НАФ ЦИКЛИТ. ПРИМИ МЕРЫ. ПОЛУЧИЛ КОРУ, МОГУ ПОДЕЛИТЬСЯ. ПРИВЕТ. КОСТЯ.] ОТВЕТ?

Здесь 412В — пароль вашего адресата (что скрывается за этим паролем, сказать трудно, скорее всего адресату легко было запомнить его). Теперь, как только с таким паролем кто-то выйдет в систему (с любого терминала), он немедленно получит сообщение.

412В ОТ КИЛ (КИЛ — это ваш пароль, его поставила в сообщение ЭВМ сама, пусть, например, это аббревиатура имени, отчества и фамилии — Константин Иванович Логинов) 25.10.83 12 ЧАС. 18 МИН. (это время отсылки вашего сообщения, его проставила ЭВМ сама по своему таймеру и календарю); далее следует текст сообщения, внесенный выше в квадратные скобки.

АДРЕСАТ ЖДЕТ ОТВЕТА (эти слова внесены ЭВМ, как реакция на слово ОТВЕТ?).

Вы же получите квитанцию от машины:

ПИСЬМО от 25.10.83, 12 ЧАС 18 МИН 412В ПОЛУЧИЛ 29.10.83, 17 ЧАС. 29 МИН.

Если ответа от 412В не последует, то ЭВМ будет каждый раз при выходе на связь 412В с ней напоминать:

412В ОТВЕТЬТЕ КИЛ НА ПИСЬМО ОТ 25.10.83, 12 ЧАС. 18 МИН.

до тех пор, пока 412В не ответит вам. Аналогично вы можете направить одно и то же письмо сразу нескольким пользователям и получите столько же квитанций о вручении писем.

Этот, несколько легкомысленный, пример не должен настраивать читателя на представление о вычислительной системе коллективного пользования как о средстве для обмена шутками между веселыми пользователями. Шутки дело хорошее, но при условии, что они не мешают остальным. В данном случае загрузка системы шутками снижает ее эффективность, так как возможности всякой вычислительной системы строго ограничены. Все обмены информацией между пользователями запоминаются и документируются системой. И делается это не только для того, чтобы выявить шутников. Это необходимо прежде всего для защиты против утери ценной информации, циркулирующей в системе.

Вычислительные системы в системах управления

Системы коллективного пользования хорошо удовлетворяют потребности отдельных пользователей, обращающихся к ЭВМ время от времени со своими нуждами по решению каких-то информационно-вычислительных задач. Эффективная работа этих систем, т. е. быстрые (и внятные) ответы на заданные вопросы, безусловно удовлетворяет пользователей. Однако, если ответы будут не столь быстры, то это не слишком сильно огорчит пользователей (ну, чертыхнутся пару раз в ожидании ответа или, в крайнем случае, пожалуются оператору или диспетчеру системы на неоправданную задержку — не больше). Никаких катаклизмов в связи с неэффективной работой системы коллективного пользования не бывает, да и быть не может, ввиду того, что задержка в ответе ЭВМ не имеет для нормального пользователя роковых последствий.

Однако есть довольно большой класс задач, где пользователь ЭВМ не может ждать. Задержка ответа

для него гибельна! Все дело в том, что пользователем вычислительной системы может быть не грешный человек-пользователь, временем которого можно иногда пренебречь, а реальные технические системы, пренебречь которыми нельзя, так как это может обернуться задержкой производства, срывом плана, катастрофой и взрывом, если это производство химическое.

Самым распространенным типом таких технических систем, нуждающихся в помощи ЭВМ, являются технологические системы, которые заняты производством продукции.

Но какое отношение имеют вычислительные системы к производству? Сроду всякое производство обходилось без ЭВМ и... слава богу, так как массовое производство существует по крайней мере лет двести, а ЭВМ... Так неужели за тридцать пять лет существования ЭВМ она стала такой незаменимой, что всякое производство без нее остановится?

Конечно, нет! Но появление ЭВМ изменило современное производство. Точнее не само производство, а управление им. Необходимость в хорошем управлении производством ни у кого и никогда не вызывала сомнения. А вот является ли человек лучшим звеном системы управления в современном производстве? Весьма сомнительно! В чем здесь дело?

Для ответа на этот не совсем простой вопрос, выясним, что же такое управление?

Управление

Под управлением понимается способ достижения в объекте каких-то определенных заданных целей. Субъект, формулирующий эти цели, хочет, чтобы объект был в таком и именно таком состоянии, а не в каком-либо другом.

Например, чтобы выпускаемая продукция была наилучшего качества или заданного качества при наименьших затратах, или достигалась какая-то другая цель.

Поставленную цель следует реализовать в объекте управления. Этот объект представляет собой определенный «кусок» нашего мира, состояние которого волнует субъекта. Говоря слово «объект», мы тем самым делим весь окружающий нас мир на две части: объект и все остальное. Такое деление подразумевает, есте-

ственно, рассечение каких-то определенных связей между объектом и остальным. Обозначим эти связи буквами X и Y. Связь X обозначает влияние среды (так мы назовем весь остальной мир) на объект, а Y является влиянием объекта на среду. Такое взаимодействие объекта и среды удобно представить в схематической форме, показанной на рис. 5. Можно ввести и другую трактовку X и Y (она более удобна для анализа феномена управления). X можно рассматривать как *состояние* среды, которым она воздействует на объект, а Y является *состоянием* объекта, которое, естественно, воздействует на среду. Теперь сам объект можно представить как некоторый преобразователь состояния среды X в состояние объекта Y. Это преобразование удобно написать в виде:

$$Y = F^0(X),$$

где буква F обозначает указанное преобразование, а индекс 0 относит его к объекту. Будем F^0 называть преобразователем объекта. Относительно него мы обычно мало что знаем.

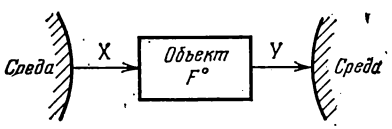

Рис. 5. Говоря об объекте, мы тем самым разделяем весь мир на две части — объект и среду. Именно это обстоятельство отражено на этом рисунке. Но, выделяя объект из среды, мы должны неизбежно рассечь его связи со средой. Эти утраченные связи нужно заменить взаимодействием объекта и среды; они обозначены на рисунке: X — это воздействие среды на объект, а Y — объекта на среду.

Итак, F^0 — преобразователь объекта, который характеризует, каким образом состояние объекта Y связано с состоянием среды X.

Приведем примеры. Пусть F^0 — технологический процесс, тогда X — сырье, поступающее на вход, а Y — готовая продукция, производимая этим процессом F^0, который здесь выступает в качестве преобразователя сырья в готовую продукцию. Если F^0 — какая-то бюрократическая организация (не обязательно в плохом смысле), то X — поток входящих, а Y — поток исходящих бумаг, т. е. F^0 в данном случае является преобразователем одних бумаг в другие.

Автомобиль F^0 преобразует состояние X его рукояток управления (газ, рулевое колесо, тормоза и т. д.) и состояние дороги в движение Y, которое характеризуется направлением и скоростью. Средой для

автомобиля являются его шофер и дорога, которые задают состояние X.

Врач F^0, если его рассматривать как объект, является преобразователем симптомов X больного в диагноз Y, а на следующем этапе диагноза Y — в план лечения Z (но здесь преобразователь F^0 будет уже иным).

Процесс прогнозирования также может быть представлен как объект F^0. Для случая прогноза погоды имеем: X — погода сегодня, а Y — прогноз погоды на завтра. Процедура принятия решения тоже может быть объектом F^0. Здесь X — ситуация, в которой необходимо принять решение, а Y — само принятое решение.

Список примеров можно продолжать долго, так как трудно представить, что не может быть объектом. Действительно, объект может быть не только живым (врач), но и не материальным (прогноз, процедура решения). Поэтому, говоря, что объект является частью нашего мира, под этим миром следует понимать в том числе и его идеальную, нематериальную часть, созданную сознанием человека.

Главное свойство объекта управления F^0 быть преобразователем некоторых определенных причин X в какие-то следствия Y.

Однако, прежде чем управлять, следует сформулировать цель этого управления. Эта цель указывает, каким должно быть состояние объекта с нашей точки зрения. Управление U здесь выступает как дополнительное воздействие на объект, в результате которого объект должен стать таким, каким мы желаем видеть его (см. рис. 6). Это означает, что управление U должно так изменять состояние объекта, чтобы оно совпадало с целевым, т. е. заданная цель реализовывалась бы. Задачу синтеза такого управления выполняет управляющее устройство (см. рис. 6). Оно получает информацию о состоянии среды X, состоянии Y объекта и о цели, которую надо достигнуть. «Продуктом» управляющего устройства является управление U.

Как же работает управляющее устройство? Для него прежде всего необходимо создать алгоритм управления, т. е. правило преобразования информации о состоянии среды X, объекта Y и цели в управление U. Делается это примерно так.

Введение управляющего воздействия U приводит к следующей зависимости состояния объекта Y от его двух входов X и U:

$$Y = F^0(X, U).$$

(Заметим, что, строго говоря, здесь F^0 не то, что было раньше, так как имеет два аргумента X и U.)

Задача управления таким образом сводится к выбору такого воздействия U, чтобы состояние объекта Y

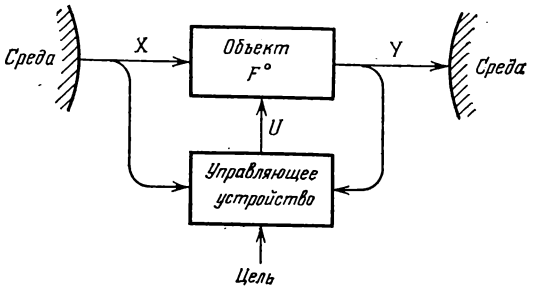

Рис. 6. Это самая общая схема управления объектом. Она применима к объектам любой природы — физическим, техническим, биологическим, социальным и даже нематериальным. Для синтеза управления U всегда нужно знать три вещи: состояния среды X и объекта Y и цель.

стало бы заданным, например, Y^*, т. е. к решению следующего «уравнения»:

$$Y^* = F^0(X, U),$$

где цель Y^* задана, а состояние среды X предполагается известным. Неизвестным здесь является управление U и, что самое неприятное, преобразователь F^0 объекта. (Если преобразователь F^0 известен, т. е. имеется его модельное представление; то все в порядке. Решая это уравнение, можно определить управление U, которое переведет объект в целевое состояние Y^*.)

Но при создании систем управления обычно относительно преобразователя F^0 объекта мы знаем очень мало. Именно незнание этого преобразователя и заставило поставить в кавычки слово «уравнение». Никакое это не уравнение! А, следовательно, и определить управление U, не зная F^0 невозможно!

Тем не менее управление, как способ достижения заданных целей в объекте, существует давно. Это можно

объяснить тем, что такое управление является в большой мере искусством, оно опирается на интуитивные представления об объекте и большой опыт. Недаром хорошими руководителями становятся лишь приобретя большой опыт управления (здесь объектом управления являются люди и их коллективы).

Управление стало наукой после того, как были найдены способы формального описания преобразователя объекта F^0, т. е. способы синтеза его модели. Под моделированием понимается создание возможности определить реакцию объекта в той или иной ситуации без экспериментов с объектом.

Как видно, модель является неким заменителем объекта, его суррогатом. Для чего нужен этот суррогат? В частности для того, чтобы можно было формализовать процесс управления и передать его ЭВМ. Показать, как ЭВМ может управлять объектом, при наличии его модели очень легко.

Обозначим модельный оператор буквой $F^м$. Тогда связь между входами X и U и выходом $Y^м$ модели можно записать в виде

$$Y^м = F^м(X, U),$$

аналогичном предыдущему выражению, с той лишь разницей, что оператор $F^м$ модели известен, а преобразователь F^0 объекта неизвестен. Теперь, если задана цель Y^*, можно составить уравнение (уже без кавычек)

$$Y^* = F^м(X, U),$$

которое можно решать и определять необходимое управление U. Чтобы не забираться в математические «дебри», отметим лишь суть дела. Она здесь состоит в том, что на модели всегда можно проверить к чему приведет то или иное управление — ведь именно для того модель и создается, чтобы заменить объект при выборе управления. ЭВМ может быстро перебрать очень много вариантов управляющих воздействий U и вычислить на имеющейся модели, к каким результатам эти воздействия приведут, т. е. каким будет состояние объекта при различных управлениях. А зная цель Y^*, нетрудно теперь выбрать то управление, которое обеспечит состояние объекта, наиболее близкое к Y^*. Вот и все!

Эта грубоватая схема использования ЭВМ при управлении объектом показывает, что без модели объекта ЭВМ в системе управления делать нечего!

Итак, при наличии модели объекта, для управления им можно использовать ЭВМ. А если модели нет и ее только предстоит сделать? Может ли ЭВМ помочь здесь?

Может и даже очень! Более того, без ЭВМ часто вообще не удается получить модель объекта. Дело в том, что процедура синтеза модели объекта по наблюдениям за его поведением требует очень много вычислений. Это легко показать.

Действительно, если имеется какая-то модель (любая) $F^{м}$, то с помощью ЭВМ легко проверить, соответствует она данному объекту или нет. Для этого достаточно сравнить состояние Y объекта F^{0} в тех или иных ситуациях (а они характеризуются входами X и U) с выходом $Y^{м}$ модели $F^{м}$ в тех же ситуациях. Если состояния объекта совпадут достаточно точно с выходами модели в одинаковых ситуациях, т. е.

$$F^{0}(X, U) \cong F^{м}(X, U),$$

то модель $F^{м}$ хорошая и может представлять объект в системе управления. (Здесь в левой части равенства — измеренное состояние объекта, а в правой — вычисленное на модели в тех же условиях, т. е. при одинаковых X и U.) В противном случае, т. е. при существенном отличии состояния объекта от выхода модели, модель плохая.

Располагая ЭВМ, можно построить процесс синтеза модели, например, таким образом. Пусть в память ЭВМ заложены наблюдения о поведении объекта — его состояния Y в различных ситуациях, которые характеризуются X и U. Эти ситуации случились в прошлом и были зафиксированы путем определенных измерений. Эта информация является исходной для синтеза модели. Будем с помощью ЭВМ генерировать какие-то модели (например, случайные). Располагая сведениями о поведении объекта в различных ситуациях (а они уже заложены в память ЭВМ), всегда можно вычислить, как ведет себя модель в этих же ситуациях. Если она «плохая», то ЭВМ меняет ее на новую, которую так же проверяет, т. е. сопоставляет с данными об объекте.

Легко видеть, что в этом процессе будет найдена модель адекватная объекту, точнее — данным об объекте,

хранящимся в памяти (или достаточно близкая к этим данным). Быстродействие машины для решения этой задачи является решающим обстоятельством. Именно поэтому модели сложных объектов могут быть созданы только мощными вычислительными машинами.

Читатель, наверное, обратил внимание, что в предыдущем абзаце есть одна существенная оговорка: создается модель адекватная не объекту, как это задумано, а данным о нем, хранящимся в памяти. Если эти данные полностью и точно отражают поведение объекта, то эту оговорку можно снять. Но можно ли сделать такие полные и точные наблюдения?

Элементарные рассуждения показывают, что нельзя. Действительно, чтобы полностью представить поведение объекта, необходимо наблюдать его во всех ситуациях, которые могут встретиться в процессе управления. Легко себе представить, что таких ситуаций бесконечное число даже для простых объектов. Чтобы образовать в памяти ЭВМ массив таких наблюдений, не хватит ни времени, ни объема памяти. Приходится ограничиваться относительно небольшим числом характерных для данного объекта ситуаций. Естественно, что модель, созданная на базе такой информации, будет приближенной. Причем степень этого приближения будет зависеть от числа зафиксированных ситуаций и от того, встретятся ли эти и аналогичные ситуации в будущем при управлении.

Точность представления поведения объекта в памяти машины всегда бывает небольшой. Это связано, во-первых, с тем, что измерительные приборы, фиксирующие состояние объекта и ситуации, в которой он находится, имеют ограниченную точность. И на процесс измерения, как и на всякий реальный процесс, накладываются всякого рода посторонние факторы, которые являются случайными помехами. Например, неточные гири весов, деформированные пружины, дефекты теплоизоляции, плохие контакты, наводки электромагнитного поля, создаваемого электрическим транспортом, метеорологической обстановкой (гроза), активностью солнца и т. д.

Далее, далеко не все важные факторы состояния объекта и ситуации вообще удается фиксировать, так как нет приборов для их измерения. Например, потому, что они дефицитны и ограниченные ресурсы управления не позволяют их достать и поставить на

объект. Или потому, что вообще неизвестно, как измерять определенный важный фактор, например, настроение оператора, выполняющего ответственную операцию в технологическом процессе, хотя оно и влияет на этот процесс.

И последнее. Объект управления, особенно сложный объект, всегда изменяется во времени, что связано с его амортизацией, дрейфом его характеристик, с его эволюцией. Это означает, что преобразователь F^0 объекта изменяется каким-то неопределенным образом во времени. Следовательно, мы, в принципе, никогда не сможем получить адекватную модель такого объекта, так как за время сбора информации о его поведении он может измениться.

Все изложенные соображения говорят о том, что модель $F^м$ объекта всегда бывает приближенной и обычно нуждается в постоянной коррекции, подстройке к объекту. Этот процесс называется адаптацией модели.

Вот и получается, что для реализации процесса управления сложным объектом просто необходимо использовать вычислительную машину, так как вычислять (и моделировать) нужно очень много. Именно эту функцию выполняют вычислительные системы.

Вычислительные системы в управлении реальными объектами

Управление реальным объектом накладывает определенные требования на вычислительные средства, используемые для управления объектом. И основным таким требованием является необходимость строгой временно́й согласованности работы ЭВМ с функционированием объекта. Здесь объект задает режим работы вычислительных средств. Именно поэтому вычислительные системы, обслуживающие работу реального объекта в процессе управления, и называют вычислительными системами реального времени. Приведем примеры такого рода управления технологическим процессом.

АСУТП

Автоматизированная система управления технологическим процессом (ее обычно называют сокращенно АСУТП) является типичным примером системы, использующей для управления вычислительную систему

реального времени. Здесь объектом управления является технологический процесс (ТП), на который действует состояние X среды в виде исходного сырья, погодных условий и т. д. и управляющие воздействия U, изменяющие параметры этого процесса, от которых зависит состояние технологического процесса. Это состояние Y фиксируется системой измерительных приборов — датчиков, преобразующих измеряемую величину в определенные электрические сигналы. Сигналы кодируются специальными преобразователями и полученные цифровые эквиваленты измеренных параметров объекта поступают непосредственно в ЭВМ. Аналогично преобразуются и попадают в ЭВМ сигналы X о свойствах поступающего на технологический процесс сырья (например, с помощью датчиков состава, размеров, температуры и т. д.).

Собрав информацию о состоянии технологического объекта и о его среде, и располагая моделью объекта, ЭВМ может вычислить управление U. Однако, для этого ей нужно сообщить цель, то есть указать каким должен быть технологический процесс. Такой целью является выполнение заданных требований к готовой продукции, т. е. требование к ее качеству и количеству, а также соблюдение всякого рода ограничений по технике безопасности, по ресурсам, выделяемым на данный технологический процесс, и т. д.

Если цель задана (точнее: цели — их может быть много, так как каждая из них накладывает требования на различные стороны управляемого процесса), то синтез управления с помощью ЭВМ уже не представляет принципиального труда (технические трудности всегда остаются, о них мы скажем позже, так как именно с ними связан переход от ЭВМ к вычислительной системе в системах управления). Предположим пока, что эти трудности преодолены и управление определено, точнее — вычислено. Это означает, что определены оптимальные значения управляемых параметров технологического процесса, и их следует реализовать, чтобы добиться целей.

Если система управления работает в режиме советчика, то вопрос реализации управления не вызывает трудностей.

Действительно, рассчитав с помощью модели оптимальные значения параметров процесса, ЭВМ предлагает их реализовать в объекте. Это предложение по-

ступает в виде текста на экране дисплея или в виде распечатки на электрической пишущей машинке. Человек должен ознакомиться с этой информацией и принять решение — согласиться с предложенным управлением или искать собственное.

Второй случай связан с тем, что решения ЭВМ не учитывают какие-то обстоятельства, известные человеку, и ему приходится соответственно корректировать управление, предложенное машиной. В таком режиме часто, выслушав совет ЭВМ, человек поступает наоборот. И вина в этом не машины, а плохой системы сбора информации об объекте и его среде или плохая модель объекта. Если бы ЭВМ располагала бы той же информацией, что и человек, и соответствующей программой, то ее решения были бы не хуже, а в ряде случаев значительно лучше, так как машина не подвержена плохому настроению, забыванию, прогулам...

Очень часто ЭВМ перерабатывает столь большой объем информации по столь сложной программе, что оценить качество ее совета человек уже не может.

Именно в этом случае человеку нужно безропотно следовать ее советам, т. е. исполнять ее распоряжения. Этим и занимается человек-оператор. Он, получив от машины указания, какими должны быть параметры процесса, устанавливает эти параметры на заданные ЭВМ уровни. Вот и все.

Однако такой режим можно использовать в очень медленных и неответственных системах управления. Человек исполняет команды машины медленно, неточно (он всего лишь человек) и... неохотно. И его можно понять. Процесс реализации в объекте заданного управления — чисто механический и превращает человека в придаток машины. Он становится, по сути дела, биологическим исполнительным механизмом, причем плохим механизмом, медленным и ненадежным. Именно поэтому так велико желание проектировщиков систем управления заменить здесь человека автоматом, перейти от режима советчика к автоматическому (а точнее: автоматизированному) режиму управления. (Следует отметить, что при большей информированности человека в ответственных случаях управление он берет на себя. Так, например, в спокойных условиях управление самолетом доверяют автопилоту. Но при взлете и посадке, при полете в «болтанку» и т. д. летчик управляет сам.)

В этом случае команды ЭВМ на изменение параметров процесса должны быть автоматически выполнены специальными устройствами, которые называют сервоприводами или просто исполнительными механизмами. На вход такого исполнительного механизма машиной подается команде в виде сигнала определенного вида. Каждому сигналу соответствует определенное значение параметра объекта, которым управляет этот механизм. Если значение параметра не соответствует поданному сигналу, то исполнительный механизм «отрабатывает» сигнал, т. е. приводит управляемый им параметр в соответствие с полученным сигналом.

Например, если таким параметром является давление воздуха в каком-то агрегате технологического процесса, то исполнительным механизмом являются компрессор, увеличивающий это давление, и клапан, стравливающий его. Компрессор включается только в том случае, когда требуемое ЭВМ давление больше имеющегося. Клапан же срабатывает в обратном случае. И лишь при равенстве наличного и потребного давлений оба исполнительных механизма бездействуют. Как видно, для работы исполнительного механизма необходимо кроме всего прочего измерять значение управляемого параметра.

Итак, мы замкнули систему управления — от состояния процесса через ЭВМ к управляемым параметрам этого процесса. Это «железное» кольцо системы управления должно функционировать в очень жестком режиме. Действительно, малейшее изменение среды или свойств самого процесса должно немедленно «отрабатываться» системой управления и соответствующим образом изменять управляемые параметры процесса с тем, чтобы не нарушать выполнение процессом заданных целевых требований. Всякое нарушение целевых требований приводит к браку (в настоящий момент или чуть позже), что недопустимо. Управляемый технологический процесс не может ждать, когда ЭВМ сможет найти управление — оно должно быть тогда, когда нужно процессу. Тем более недопустима неработоспособность ЭВМ (например, по причине ее ненадежности), так как именно в этот момент неработоспособности ЭВМ может понадобиться решение (управление), от которого будет зависеть эффективность управляемого процесса не только сейчас (например, в аварийных ситуациях), но и на много месяцев вперед.

Как видно, от ЭВМ, работающей в режиме реального времени в системе управления, требуется много: большое быстродействие, огромная надежность, чрезвычайная оперативность. Современные ЭВМ еще не обеспечивают всех этих жестких требований. Вот и приходится создавать специальные вычислительные системы, лишенные указанных недостатков.

Другой пример.

Медицинские системы интенсивной терапии

Здесь пойдет речь о грустных, но крайне необходимых мероприятиях в жизни тяжелобольных людей, причем в те их трудные минуты и секунды, когда они слишком близко приближаются к границе, отделяющей жизнь от смерти. Речь пойдет об использовании ЭВМ в процессах лечения больных, находящихся в тяжелом состоянии (например, после операции, при инфаркте миокарда и т. д.). Сложность ситуации заключается в том, что такое состояние может длиться достаточно долго, что затрудняет организацию постоянного врачебного наблюдения. Это и заставляет создавать так называемые мониторные системы, предназначенные для непрерывного слежения за состоянием больного и для выдачи сигнала тревоги, когда состояние становится опасным. Монитор подключается к больному с помощью системы датчиков, измеряющих характеристики и параметры его состояния (обычно пульс, ритм дыхания, температура, электрокардиограмма, энцефалограмма и т. д.). Эти характеристики непрерывно вводятся в специальный прибор, называемый монитором, которому задаются критические значения параметров. Задача монитора проста — сравнивать параметры больного с критическими и вызывать врача при выходе хотя бы одного из этих параметров на критический порог.

Такими мониторами снабжены все палаты интенсивной терапии, которые есть во многих больницах. Но лечение здесь осуществляет врач, вызванный мониторной системой, которая выступает лишь в роли «недремлющего ока», что само по себе немаловажно. Переработка информации, осуществляемая монитором, крайне скудна: это сравнение параметров больного с установленными порогами. Для этого не нужно применять ЭВМ. И ее не применяют.

Легко себе представить, что такого рода мониторная система часто беспокоит врача зря. Это бывает каждый раз, когда состояние «аварийного» параметра больного однозначно указывает на то, что надо делать. Например, повышение сахара в крови требует введения инсулина, а возбуждение, фиксируемое энцефалограммой, преодолевается успокаивающими средствами.

В таких «штатных» ситуациях врач не нужен — достаточно медсестры, которая введет нужное лекарство больному по указанию монитора. Эти указания нетрудно запрограммировать заранее и встроить в схему монитора. Например, при излишнем сахаре в крови зажигать на столике дежурной медсестры табло с надписью «ИНСУЛИН». И лишь в нештатных ситуациях, когда выбор лекарства или другого воздействия на больного должен быть сделан нестандартным образом и с учетом других обстоятельств состояния больного, которые не фиксируются монитором, следует вызывать врача.

А теперь естественно сделать последний шаг и заменить медсестру, т. е. замкнуть контур управления. Это особенно важно в тех случаях, когда вводить лекарства надо быстро, так как малейшее промедление грозит тяжелыми или даже фатальными последствиями. Такие ситуации часто встречаются в процессе лечения тяжелых больных.

Именно поэтому совсем недавно появились иные медицинские системы, называемые системами интенсивной терапии, где лечение возлагается непосредственно на ЭВМ. Для этого к больному подключают специальные аппараты терапевтического воздействия (введения лекарств), которые управляются ЭВМ. Теперь ЭВМ, будучи информированной монитором о состоянии больного, может принимать самостоятельные решения в тех случаях, когда ясно, что надо делать. Именно эти ситуации записываются в память ЭВМ. Как только она распознает одну из таких ситуаций, то дальнейшее поведение системы — что и в каком количестве следует вводить в организм больного — предписано программой, хранящейся в памяти той же ЭВМ. Остается отдать соответствующие команды исполнительным механизмам, которые, например, введут предписанную дозу нужных лекарств в кровь больного.

Если же состояние больного выходит за рамки нормального (точнее: того, которое в настоящий момент врачи считают нормальным) и это состояние не числит-

ся в штатных среди аварийных, то ЭВМ вызывает врача.

Хорошо видно, как удобны такого рода мониторы с ЭВМ. Они делают все, относительно чего есть строгие медицинские предписания и, тем самым, разгружают персонал больницы. Более того, эти системы выполняют свои функции значительно более оперативно и точнее, чем это сможет сделать самая расторопная медсестра.

Однако ЭВМ, которая реализует функции управления в этой системе должна обладать высочайшей надежностью. Действительно, сбой или отказ медицинской системы управления грозит смертью больного, состоянием которого она управляет. Именно это обстоятельство не позволяет здесь воспользоваться обычной ЭВМ, надежность которой еще не велика. Здесь необходимо использовать вычислительную систему реального времени, функционирование которой определяется только объектом управления — состоянием больного, к которому она подключена.

И последний пример.

Управление экспериментом

Еще древние знали, что существуют три способа получения информации: мнение компетентных людей, интуиция (раньше ее называли «гласом божьим») и эксперимент. Самыми почтенными и уважаемыми были два первых способа. Первый был источником осмысленного человеческого опыта, а второй — тоже опыта, но неосмысленного, подсознательного, интуитивного, но тем не менее очень важного. И только последний способ был узаконен в науке лишь со времен Галилея — лет триста тому назад. В эксперименте воссоздается ситуация, которую естественным образом наблюдать или трудно или вообще нельзя. Эксперимент позволяет выделить изучаемое явление из массы других, взаимодействующих с ним, и получить необходимую информацию. Эта информация собирается системой датчиков, показания которых представляют собой сигналы определенного вида. Эти сигналы несут искомую информацию о явлении, ради изучения которого ставится эксперимент.

А так как эксперимент всегда связан с определенными затратами (материальными, энергетическими,

временны́ми и т. д.), то экспериментатор, желая «выжать» из эксперимента все, что можно, «навешивает» на экспериментальную установку столько датчиков, сколько можно (а иногда и больше). При этом в процессе эксперимента появляется огромное количество информации в виде показаний приборов-датчиков.

Показания обычно записывают на какой-то носитель (протокол, фотобумага, магнитофонная лента и т. д.) с тем, чтобы после эксперимента разобраться в записях и сделать какой-то вывод относительно наблюдаемого в проведенном эксперименте явления.

Следует отметить, что совершенствование техники сбора информации привело к тому, что поток этой информации стал настолько большим, что разобраться в нем с каждым годом становилось все труднее и труднее. Создалась противоречивая ситуация: желание получить больше информации во время эксперимента должно было сдерживаться трудоемкостью ее последующей обработки.

Несколько лет назад на Западе был выдвинут лозунг: «Кто владеет информацией, тот владеет миром». На первый взгляд довод «за информацию» вполне убедительный. Но только — на первый. Оказалось, что от владения информацией до ее использования с целью «владения» чем-либо дистанция огромного размера.

Сама по себе информация ничего не дает, если мы не овладеем ею, т. е. не сумеем ее использовать для каких-то конкретных целей. Такой целью при изучении всякого явления является его познание, т. е. создание модели этого явления.

Вот и получается, что многие километры исписанной магнитной пленки или отснятой фотопленки далеко не решают задачи эксперимента. Их надо обрабатывать, на что часто уходит намного больше времени и труда, чем на сам эксперимент. Да и результат будет получен с большой задержкой, что может его полностью обесценить.

Так возникла проблема обработки экспериментальной информации во время ее поступления, т. е. в реальном масштабе времени. Для этого нужно использовать вычислительную систему, которая бы справлялась с потоком информации и была бы достаточно надежной в работе, так как ее сбой приведет к потере информации, которую восстановить уже не удастся (разве что

повторив аналогичный эксперимент). Обычная ЭВМ с этой работой не справляется.

Это одна сторона проблемы современного экспериментирования. Другая связана с методологической спецификой эксперимента. Дело в том, что очень часто ценность экспериментальной информации прямо зависит от ее неожиданности. Если полученная информация не неожиданна для экспериментатора, то и ее ценность невелика. Ценно то, что не укладывается в рамки имеющихся представлений, что неожиданно. И, чем более неожиданно для экспериментатора поведение экспериментальной установки, тем более ценную информацию получает он во время эксперимента.

В результате складывается странная ситуация: экспериментатор ставит эксперимент, ориентируясь на получение каких-то определенных результатов, а ценными для него будут не эти ожидаемые результаты, а те, на которые он не рассчитывает и, естественно, не настраивал свой эксперимент! Если бы экспериментатор ждал эту неожиданность, то эксперимент нужно было бы поставить по-другому. Но тогда и ценность результатов этого эксперимента была бы небольшой.

В этом и заключается парадоксальность всякого эксперимента. Для преодоления этого парадокса нужно иметь возможность управлять экспериментом в зависимости от получаемых результатов. Приведем простейший пример необходимости такого управления.

Пусть результатом эксперимента является некоторая величина. Экспериментатор, планируя эксперимент, должен для ее измерения предусмотреть определенный измерительный прибор. Выбор этого прибора зависит от того, какое значение измеряемого параметра ожидается. Чем уже интервал ожидаемого значения измеряемой величины, тем точнее ее можно измерять. Поэтому экспериментатор заинтересован в том, чтобы как можно точнее оценить этот интервал, так как именно это обеспечит максимальную точность.

Пусть в эксперименте измеряемая величина вышла за пределы интервала. Результат этот неожидан и поэтому очень ценен. Но эксперимент в результате оказался... обесцененным, так как прибор был выбран неправильно. И надо снова ставить эксперимент, но с другим прибором. (Конечно, можно утешить себя, что результатом этого эксперимента был выбор нового прибора, но это утешение слабое.)

А если бы в процессе экспериментирования можно было бы изменять прибор, т. е. корректировать его, используя результаты измерения, то эксперимент не пришлось бы повторять.

Здесь рассмотрен простейший случай управления экспериментом — управление измерительным прибором. Но в зависимости от получаемых результатов таким же образом можно изменять и сам план эксперимента. Так мы приходим к использованию вычислительной техники для управления в процессе экспериментирования. Она обеспечивает не только обработку эксперимента, что само по себе немаловажно, но и позволит изменить этот эксперимент в том направлении, которое нужно экспериментатору.

Эту возможность предоставляют вычислительные системы реального времени. Такая система, фиксируя результаты эксперимента и обрабатывая их необходимым образом, изменяет параметры экспериментальной установки так, чтобы получить именно тот результат (точнее: эффект), который запрограммировал экспериментатор.

Приведем примеры вычислительных систем реального времени, которые управляют экспериментом. Это, например, управление ускорителями. Современный ускоритель является чрезвычайно сложной и тонкой экспериментальной установкой, для управления которой используют вычислительные системы реального времени. Именно системы, а не просто ЭВМ, так как от точности и надежности их функционирования зависят точность и надежность получаемых экспериментальных результатов.

Вычислительные системы реального времени используются широко в космических экспериментах, где они применяются для управления измерительными приборами, например, радиотелескопом на борту спутника и т. д.

Робототехника является одной из новых и весьма перспективных областей, где вычислительные системы реального времени используются для управления экспериментом. Робот, находящийся в новой обстановке (на дне моря, на другой планете и т. д.) должен прежде всего «освоиться» в ней. Для этого он должен сделать ряд экспериментов, с помощью которых он узнает о новой среде все ему необходимое. Надо ли говорить, что для осуществления таких экспериментов

необходимо иметь весьма совершенную вычислительную систему реального времени, которая, по сути дела, и образует мозг этого робота. Ее эффективность и определяет интеллектуальность робота.

Специфика вычислительных систем реального времени

Приведенные выше примеры применения вычислительной техники к задаче управления реальными объектами позволяют сформулировать требования, предъявляемые к вычислительным системам реального времени.

Во-первых, такая вычислительная система должна быть снабжена мощной системой оперативного ввода и вывода информации, надежно связывающей ее с управляемым объектом и средой. Входами для нее являются показания датчиков, регистрирующих состояние среды и объекта управления, а выходами — исполнительные механизмы, изменяющие параметры, структуру и функционирование объекта.

Требование оперативности, очевидно, исключает использование перфолент, перфокарт или дисплеев, традиционных для обычных ЭВМ, применяемых для общения с человеком — пользователем (о них мы много говорили в первой главе). Здесь необходимо иметь электронные преобразователи информации. Они преобразуют информацию из формы, кодируемой датчиком (обычно в виде потенциала, частоты или фазы электрического сигнала), в форму, воспринимаемую вычислительной системой, т. е. в двоичном коде — наборе нулей и единиц. Это преобразование осуществляется так называемыми аналого-цифровыми преобразователями (принятая аббревиатура — АЦП) или короче: преобразователями «аналог-код». Здесь под словом «аналог» подразумевается поступающий с датчика сигнал, кодирующий информацию в непрерывной (аналоговой) форме (уровень сигнала, его частота или фаза). А под словом «код» подразумевается цифровой код — часто это простейший двоичный код. Полученные в результате преобразования двоичные коды сигналов, пришедших с объекта, легко воспринимаются вычислительной системой и тут же перерабатываются в соответствии с заданным и заранее запрограммированным алгоритмом управления объектом.

Пояснить работу аналого-цифрового преобразователя можно на примере электрических часов с «прыгающей стрелкой». Накопление заряда, происходящее непрерывно в конденсаторе (или другой аналогичный процесс), при достижении заранее установленного порога приводит в действие реле, связанное с исполнительным механизмом. С принципиальной точки зрения механизм управления минутной стрелкой часов можно рассматривать как шестидесятипозиционное реле. Этот же принцип применяется в электронных часах с дисплеем на жидких кристаллах, в цифровых термометрах и т. п.

Управляющее воздействие (точнее: информация об этом воздействии), выработанное вычислительной системой, как реакция на поведение объекта, должно быть реализовано в этом объекте исполнительными механизмами. Но представлена эта информация в «машинной», т. е. в двоичной форме. Для ее передачи на исполнение необходимо преобразовать ее в непрерывную форму. Осуществляют эту операцию цифро-аналоговые преобразователи (ЦАП) или короче: преобразователи «код-аналог». Это преобразование осуществляет перевод двоичных кодов в непрерывный сигнал, который способны воспринимать исполнительные механизмы (сервоприводы), изменяющие параметры управляемого объекта.

Как видно, быстродействующие и надежные цифро-аналоговые и аналого-цифровые преобразователи являются необходимыми элементами вычислительной системы реального времени. С их помощью и происходит общение вычислительной системы с объектом управления и его средой.

Другим специфическим требованием, предъявляемым к вычислительной системе реального времени, является ее живучесть, т. е. способность выполнять возлагаемые на нее функции даже при выходе из строя каких-то ее элементов. Это требование связано прежде всего с тем, что процесс управления реальным объектом нельзя приостанавливать ни на мгновение, так как это может привести к снижению эффективности функционирования объекта в лучшем случае и к аварии — в худшем.

Именно поэтому в роли вычислительной системы реального времени не могут выступать обычные ЭВМ для вычислений, надежность и тем более живучесть кото-

рых крайне невелики — для ЭВМ это попросту не нужно.

Для создания живучих вычислительных систем есть два пути и оба они используются. Первый — это применение нескольких ЭВМ, объединенных специальным образом в вычислительную систему. А второй — создание специальных вычислительных систем повышенной живучести. Рассмотрим оба направления.

Надежные машины из ненадежных элементов!

Такой девиз был провозглашен знаменитым американским математиком Джоном фон Нейманом в начале 1950 года. Он хорошо понимал как важно уметь создавать надежные системы. Надежность системы на первый взгляд не может быть выше надежности любого из ее элементов, так как при выходе его из строя должна остановиться и вся система. Так неужели нам никогда не удастся создать надежные системы? А точнее: можно ли создать надежную машину из ненадежных элементов? Оказывается можно! Причем можно сделать сколь угодно надежную машину из сколь угодно ненадежных элементов. И строгое математическое доказательство этого неочевидного положения дал фон Нейман.

Идея, предложенная им, гениально проста: вводить дублирование ненадежных элементов, при котором несрабатывание одного из таких элементов не вызывает сбоя, так как срабатывают другие, дублирующие его. Очевидно, что такое дублирование нужно вводить не кое-как, а «с умом», а как именно, впервые указал фон Нейман. С тех пор создание надежных систем из ненадежных элементов не представляет принципиальных трудностей. Конечно, приходится вводить в систему избыточность и иногда большую, но это только плата за возможность создать надежную машину из ненадежных деталей.

Эта идея и легла в основу первого пути обеспечения живучести вычислительных систем. Реализуется она следующим образом.

Если одна ЭВМ ненадежна, то в соответствии со сказанным, ее следует задублировать второй. Так и делают. Берут две одинаковые ЭВМ (можно и разные, но это усложнит задачу), вводят в них одинаковые программы и сообщают на входы одну и ту же информацию

о состоянии объекта управления. Очевидно, что при правильной работе обеих ЭВМ результаты, т. е. вырабатываемые управляющие решения, должны быть одинаковыми. Это и есть контроль правильности работы ЭВМ, так как одинаковые ошибки в двух ЭВМ возникают крайне редко и ими можно пренебрегать.

Если решения ЭВМ не совпадают друг с другом, то это означает, что в одной из них произошел сбой. Прежде всего надо решить не был ли сбой случайным. Для этого достаточно повторить расчет на обеих ЭВМ — совпадение полученных результатов подтвердит, что сбой был случайным и можно продолжать работу.

Если же сбой оказался не случайным (это будет при повторном несовпадении результатов), то следует искать неисправность в одной из ЭВМ. Делают это специальные диагностические программы — тесты. Они выявляют и локализируют неисправность в одной из ЭВМ. Другая, исправная, ЭВМ после этого немедленно включается в работу по управлению, а неисправная — ремонтируется, после чего снова начинает дублировать работу первой.

Этот режим, как видно, повышает живучесть образованной таким образом вычислительной системы, состоящей из двух ЭВМ. Однако в работе такой вычислительной системы есть два момента, когда быстродействие и живучесть уменьшаются.

Действительно, во время работы диагностических программ, выявляющих неисправную ЭВМ, управление объектом не происходит и тем самым снижается среднее быстродействие вычислительной системы. Для преодоления этого обстоятельства необходимо иметь значительный запас быстродействия, чтобы простой во время прогона диагностических программ минимально сказывался на эффективности управления объектом.

Живучесть вычислительной системы резко снижается и во время ремонта одной из ЭВМ и совпадает с живучестью другой работающей с объектом машины. Это обстоятельство накладывает довольно жесткие тробования на время ремонта ЭВМ. Оно должно быть минимальным. Для этого проще всего заменить неисправный блок другим, для чего надо иметь запас всех блоков ЭВМ и возможность быстрой их замены. Эти технические проблемы вполне разрешимы.

Для преодоления (точнее: смягчения) недостатков описанной двумашинной вычислительной системы можно

предложить трехмашинную систему. Все три ЭВМ в этом случае также работают по одной и той же программе, то есть дублируют (или, точнее, если не бояться новых слов, триплируют) друг друга. Если все три получаемых результата совпадают, то работа всех ЭВМ считается правильной и это выработанное решение реализуется в объекте управления (легко представить, что возникновение одной и той же ошибки в трех ЭВМ практически невозможно). При возникновении сбоя в одной из трех ЭВМ уже нет необходимости повторять счет всеми ЭВМ. Решение принимается по двум совпадающим результатам, а третья ЭВМ, результат которой отличается от двух других, повторяет расчет. Если сбой оказался не случайным, то именно в этой ЭВМ запускается диагностическая программа, а две других ЭВМ продолжают работу по описанной выше схеме.

Легко видеть, что живучесть такой трехмашинной вычислительной системы значительно выше двумашинной, но и она может выйти из строя, когда все три ЭВМ будут неисправны. Это крайне редкое событие, но оно возможно.

При этом нужно считаться и с такой возможностью, когда неисправность двух машин приводит к трем разным результатам, и нельзя сразу сказать, какая машина осталась исправной. Под подозрением — все три! И нужно время для выявления той, которой можно доверить продолжить управление.

Можно ли еще больше повысить живучесть вычислительной системы? Разумеется. Делается это аналогично вышеизложенному путем наращивания числа дублирующих ЭВМ. Живучесть такой системы с ростом числа ЭВМ возрастает очень быстро и может стать сколь угодно большой. Правда за это приходится «расплачиваться» целыми ЭВМ. Но если объект управления этого требует, то на такие расходы обычно идут, тем более, что ЭВМ могут быть не очень надежные, а, следовательно, не очень дорогие.

Следует отметить, что, кроме основных ЭВМ, для реализации описанной схемы нужна еще одна небольшая «машинка», которая бы реализовала описанный способ сравнения результатов, отключения неисправной ЭВМ и т. д. Она выполняет управляющую роль в вычислительной системе. При расчете живучести такой системы необходимо учитывать и эту управляющую

ЭВМ. Для надежных вычислительных систем эта машина должна быть очень надежной. Иначе именно она снизит надежность вычислительной системы.

А теперь рассмотрим другой путь повышения живучести вычислительных систем реального времени. Если первый можно назвать многомашинным, то второй — многопроцессорным. Смысл его сводится к следующему.

Представим себе, что все важнейшие блоки ЭВМ продублированы — две оперативные памяти, два процессора и т. д. Грубо говоря, из двух ЭВМ сделали одну. Между этими блоками стоят специальные управляющие устройства, которые решают, какой из двух блоков в данный момент должен работать (могут работать и оба). При выходе из строя одного из блоков такое управляющее устройство отключает его, сигнализирует о необходимости ремонта и продолжает работать со вторым блоком. И так для каждой пары блоков.

Легко видеть, что живучесть такой двупроцессорной вычислительной системы значительно выше, чем двумашинной, рассмотренной ранее. Действительно, возможность автоматически менять блоки во время работы, обеспечивает работоспособность даже в том случае, если выйдет из строя половина разных блоков, то есть блоков каждого типа. Из оставшихся будет составлена ЭВМ, которая и обеспечит работоспособность. Система откажет лишь в том случае, если одновременно возникнет неисправность в паре однотипных блоков.

Очевидно, что живучесть такой вычислительной системы легко наращивать путем добавления нужного числа тех блоков, которые чаще выходят из строя.

Такую вычислительную систему называют системой с автоматической реконфигурацией (или: «элегантной деградацией»). Ее структура изменяется автоматически в процессе функционирования так, чтобы сохранять работоспособность, столь необходимую для систем реального времени.

Нетрудно видеть, что такая вычислительная система обладает еще одним интересным свойством, важным для системы реального времени. Это возможность параллельной работы блоков системы. Действительно, если оба процессора исправны (за этим следит управляющее устройство), то их можно загрузить решением задач, которые возможно решать одновременно. В результате быстродействие системы увеличивается при-

мерно вдвое. Если процессоров больше, то соответственно возрастет и быстродействие системы во время исправной работы всех ее блоков. При наличии неисправностей быстродействие становится номинальным.

Такое распараллеливание процесса вычислений является одним из способов повышения быстродействия вычислительных систем. Проблема увеличения скорости обработки информации, т. е. повышения мощности вычислительных средств, — особая, и решается она эффективно путем создания многопроцессорных вычислительных систем, которые в английском языке образно называют словом numbercruncher (number число, а cranch — грызть), т. е. «перегрызатель» чисел или, точнее, перемалыватель.

Задачи бывают разные ...

Разработчики современной вычислительной техники наряду с созданием обычных «коммерческих» ЭВМ, где стоимость и удобство эксплуатации являются определяющими, большое внимание уделяют сверхмощным вычислительным системам и машинам, способным выполнять миллиарды операций в секунду. И это не погоня за рекордами. Современная наука и техника ощущают весьма острую необходимость в таких сверхбыстродействующих машинах.

Дело в том, что трудоемкость решаемых задач весьма различна: одни задачи решаются за доли секунды (именно о них говорят, когда хотят показать и удивить возможностями современной вычислительной техники); для решения других очень сложных, машины еще не придуманы. Для удобства классификации способов решения задач на ЭВМ введено их деление на два класса, сложность решения которых — полиномиальная и показательная. Это означает, что время решения задачи может зависеть от ее размерности полиномиально, то есть имеет вид

$$T_1 = An^{\alpha},$$

где A и α — некоторые постоянные (не в них суть), а n — размерность решаемой задачи (она характеризует «размер» задачи), и показательно:

$$T_2 = B\beta^{n},$$

где B и β также некоторые постоянные.

Отличие показательной и полиномиальной зависимостей состоит в том, что для задач большой размерности, т. е. при больших n всегда $T_2 \gg T_1$, т. е. показательная сложность всегда значительно превышает полиномиальную независимо от значений A, B, α и β. Это легко видеть из следующих простых рассуждений. Если удваивается размерность задачи ($n \to 2n$), то при полиномиальной трудоемкости ее решения на нее надо затратить лишь в 2^α больше времени. А при показательной трудоемкости — в β^n раз больше, т. е. чем больше размерность, тем быстрее растет трудоемкость ее решения.

Примером алгоритма полиномиальной трудоемкости является решение линейной системы алгебраических уравнений, где n — число неизвестных (и уравнений). При этом объем вычислений пропорционален кубу числа неизвестных, т. е. $\alpha = 3$. А показательную трудоемкость имеют все переборные методы дискретных задач, например, задача о коммивояжере или бродячем торговце, которому надо построить минимальный маршрут передвижения по n определенным городам. (Заметим, что эта задача имеет и важные технические трактовки, например, в случае необходимости определения минимальной длины траектории движения сверлильной головки автомата). Число возможных маршрутов торговца по n городам равно $n!$, удвоение числа городов, как показывают простые расчеты, увеличивает число маршрутов более чем в n^n раз.

Так, например, при $n = 10$ удвоение размерности линейной системы алгебраических уравнений увеличивает трудоемкость решения задачи в 10^3, т. е. в тысячу раз. А решение задачи коммивояжера при удвоении числа городов станет более трудоемким в 10^{10}, т. е. в десять миллиардов раз! Легко себе представить, что методы решения, имеющие показательную трудоемкость, требуют огромных затрат времени, а при больших n задачи и вообще не решаются.

Отсюда следует весьма простой, но важный для практики, вывод: для решения задач надо изобретать алгоритмы полиномиальной трудоемкости, а не показательной.

Но хорошо сказать «надо»! На деле далеко не всегда удается найти метод, имеющий полиномиальную трудоемкость. А метод полного перебора, имеющий показательную трудоемкость, является универсальным

методом решения задач. Он всегда дает искомое решение, его логика элементарна, что позволяет его легко программировать. Но применение полного перебора даже при не очень больших размерностях n задачи требует больших, а порой и огромных, вычислительных мощностей. Именно это является одной из причин, стимулирующих создание сверхбыстродействующих ЭВМ, — слишком много важных научно-технических и народно-хозяйственных задач надо решать, используя этот неизящный, но продуктивный метод полного перебора.

Другим очень важным классом практических задач, требующих больших вычислительных мощностей являются задачи стохастического моделирования, т. е. задачи, требующие применения метода Монте-Карло (мы уже говорили об этом методе в первой главе в связи с алгоритмическим языком GPSS). Известно, что результат решения задачи методом Монте-Карло является случайной величиной, разброс которой зависит от числа циклов моделирования исследуемого процесса. Действительно, чем больше сделать таких циклов, тем точнее можно оценить интересующие нас показатели процесса. Теория метода дает очень четкую зависимость разброса Δ от числа N циклов моделирования:

$$\Delta = \frac{C}{\sqrt{N}},$$

где C — некоторая постоянная. Отсюда хорошо видно, что для увеличения точности оценки в 2 раза объем моделирования надо увеличить в 4 раза. А желая уменьшить разброс в 10 раз следует число циклов моделирования увеличить в 100 раз.

Как видно, для того, чтобы достаточно точно считать методом Монте-Карло, надо либо иметь очень много времени, либо располагать вычислительной машиной большой мощности. А так как этот метод составляет основу имитационного моделирования — одного из самых распространенных методов решения научно-технических и народнохозяйственных задач, то необходимость в создании сверхбыстродействующих вычислительных средств становится очевидной и важной. Это и есть другая причина, побуждающая создавать мощные компьютеры.

Принципы «цифроперемалывания»

Есть много путей убыстрения процесса обработки информации. Рассмотрим основные.

Первым и самым естественным является создание мощных процессоров, способных сверхбыстро выполнять возлагаемые на них функции — выполнение логико-арифметических операций типа: сложить два числа, сравнить два символа и т. д. Так как быстродействие ЭВМ определяется в основном быстродействием ее процессора, то для создания мощных ЭВМ надо создавать мощные процессоры. И такие процессоры создаются.

Их производительность доходит до миллиона операций в секунду; таков, например, процессор нашей известной (и знаменитой) машины БЭСМ-6. Это очень большая мощность и она будет расти со временем. Но... ее не хватает. Современные задачи требуют ЭВМ, производительность которых исчисляется миллиардами операций в секунду и более.

Что же делать? Ждать, когда такие процессоры будут изобретены? Пожалуй это будет долго даже в наш быстрый электронный век! К тому же есть серьезные подозрения, что вычислительная техника приблизилась к физическому пределу быстродействия процессоров. Во всяком случае, нельзя забывать, что физические носители сигналов никогда не превысят скорость света. Но нельзя ли повысить производительность вычислительной машины другим способом, не увеличивая быстродействия процессора? Ведь можно же сделать надежную машину из ненадежных элементов — мы только что в этом убедились!

Параллельная обработка

Человеческий опыт подсказывает один из таких способов — распараллеливание работ. Если выполняемую работу можно разделить на независимые части, то их выполнение естественно одновременно возложить на разных исполнителей (процессоры). Очевидно, что при этом значительно сократится общее время выполнения работы и тем самым, увеличится быстродействие.

Эта простая и естественная идея была реализована при создании мощных вычислительных систем.

Как осуществить распараллеливание? Ведь программа для ЭВМ представляет собой последователь-

ную цепочку операторов. Но последовательность написания программы вовсе не означает именно такой последовательности ее выполнения. Многие ее участки (с какого-то момента) не зависят друг от друга и с этого момента могут выполняться одновременно. В этом и состоит смысл распараллеливания.

Проиллюстрируем эту процедуру на простом примере. Пусть необходимо вычислить сумму

$$y = \varphi(x) = \sum_{i=1}^{n} f_i(x)$$

для заданных значений аргумента $x_1, x_2, \ldots, x_K$ (например, для того, чтобы построить эту функцию). Хорошо видно, что сначала надо вычислить значения всех функций f_i, а уж потом их сложить. Вычисление функций f_i можно делать параллельно на n процессорах, на что затратится время, необходимое для вычисления самой трудоемкой функции из f_i $(i = 1, \ldots, n)$, и потом сложить n чисел.

Для того чтобы проиллюстрировать это графически, введем одно очень удобное обозначение процессора. Что такое процессор? Это устройство, преобразующее входные данные в результат с помощью заданной ему программы. Поэтому его удобно представить в виде преобразователя двух входов (один — данные, другой — программа) в один выход — результат вычислений.

Он показан на рис. 7, *а*. Здесь u — входные данные, $P(\alpha)$ — программа (от английского Program), специфика которой определяется α — это имя программы. Выход v является результатом применения программы $P(\alpha)$ к данным u. Например, при вычислении заданной функции f_j для заданного значения аргумента x_i входом процессора будут: данные — значение аргумента x_i и программа $P(f_j)$ вычисления функции f_j, а выходом — значение этой функции $f_j(x_i)$ (см. на рис. 7, *б*). Другой пример — суммирование. При этом данными являются два числа z и w, а программой — программа суммирования $P(\Sigma)$. Результат — сумма двух входов $z + w$ (см. рис. 7, *в*).

И все это можно сделать на одном и том же процессоре путем изменения его программы $P(\alpha)$.

Таким образом, располагая достаточным количеством процессоров и задавая им различные программы, можно строить разнообразные вычислительные системы.

Продолжим рассмотрение нашего примера вычисления суммы заданных функций и построим такую систему для этого примера.

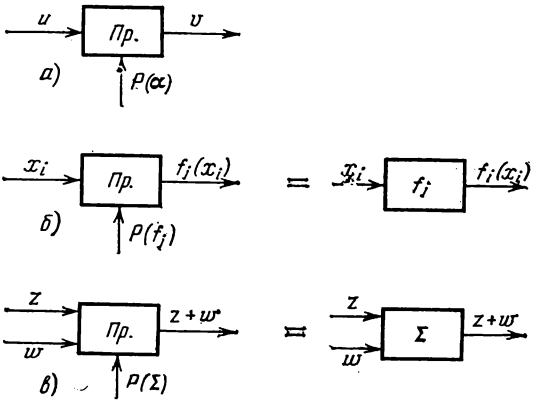

Рис. 7. Процессор (Пр.) является по сути своей управляемым преобразователем, который преобразует вход u в выход v способом, указанным в программе $P(\alpha)$ (см. рис. а). Простейшими примерами такого преобразования являются вычислений значений заданной функции (рис. б) и обычная сумма (рис. в).

На рис. 8 показана структура такой параллельной вычислительной системы из n процессоров Пр1, ..., ..., Прn, на каждый из которых подается программа

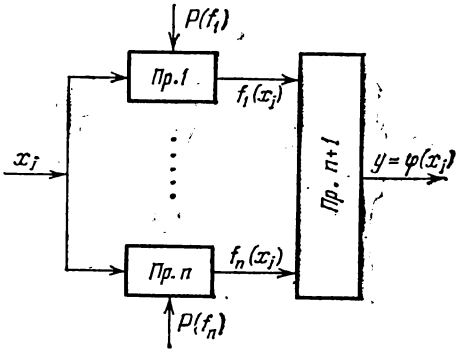

Рис. 8. Вот так реализуются параллельные вычисления с помощью нескольких процессоров. Здесь время лимитируется функцией, требующей максимального времени вычисления.

вычисления «своей» функции f_j (она обозначена $P(f_j)$ и значение аргумента x_i. На выходе j-го про-

цессора Прj таким способом образуется (точнее: вычисляется) значение $f_j(x_i)$. Все выходы этих процессоров суммируются $n+1$-м процессором Пр$(n+1)$. Его программа $P(\Sigma)$ и образует искомую функцию.

Таким образом, $n+1$ процессор образовали вычислительную систему, которая вычисляет функцию $\varphi(x_j)$ за время

$$\tau_{max}+(n-1)\tau^{\Sigma},$$

где τ_{max} — время вычисления самой трудоемкой функции из f_i, а τ^{Σ} — время суммирования двух чисел процессором. При последовательном вычислении этой функции на одном процессоре понадобилось бы время

$$\sum_{j=1}^{n}\tau_j+(n-1)\tau^{\Sigma},$$

где τ_j — время вычисления j-й функции f_j, т. е. значительно больше. Выигрыш от параллельных вычислений, как видно, достаточно велик. Если n велико, а функции f_i «похожи» друг на друга, то выигрыш практически n-кратный. (Заметим, что в действительности временны́е затраты будут несколько больше, так как здесь не учитывается время извлечения из памяти чисел и программ, которые должны выполняться процессорами. Но эти затраты всегда бывают достаточно малыми, и ими можно пренебрегать в грубых оценках, которыми мы здесь и занимаемся).

Нельзя ли еще улучшить время вычисления $\varphi(x)$ за счет введения дополнительного распараллеливания? Ведь процедура суммирования $n+1$-м процессором реализуется последовательно, так как стандартный процессор не может складывать больше двух чисел одновременно. Давайте распараллелим процесс суммирования следующим образом (для простоты рассмотрим случай $n=8$).

1. Сначала (на первом такте) образуем одновременно на четырех процессорах — сумматорах четыре суммы:

$$f_1(x_i)+f_2(x_i),\ \ldots,\ f_7(x_i)+f_8(x_i).$$

2. На следующем такте двумя процессорами образуются две суммы:

$$\sum_{j=1}^{4}f_j(x_i);\quad \sum_{j=5}^{8}f_j(x_i).$$

3. И наконец, на последнем третьем такте получается значение искомой функции $\varphi(x_i)$.

Как видно, для этого понадобилось лишь три такта работы сумматора (вместо семи по последовательной схеме суммирования) и 7 процессоров, работающих по программе суммирования двух чисел (образованная схема вычислительной системы показана на рис. 9).

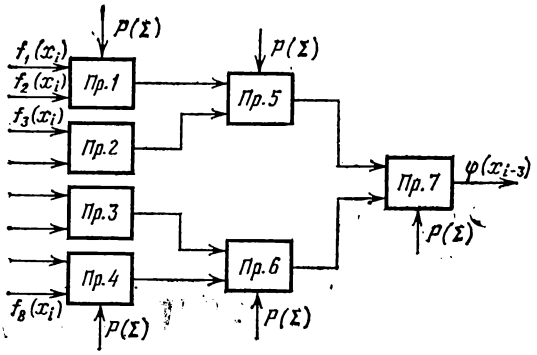

Рис. 9. А вот так можно быстро реализовать суммирование восьми функций. Здесь эффект убыстрения получен за счет того, что суммирование производится параллельно.

Суммарное время, затраченное на всю процедуру определения $\varphi(x_i)$ включая вычисление значений функций f_i $(n = 8)$ равно:

$$\tau_{max} + 3\tau^{\Sigma}. \qquad (*)$$

А теперь подсчитаем, сколько времени понадобится на вычисление всей таблицы $\varphi(x_i)$ $(i = 1, \ldots, k)$. Не торопитесь умножать $(*)$ на k, так как очень простые соображения показывают, что затраты будут меньше:

$$k\tau_{max} + 3\tau^{\Sigma}.$$

Действительно, суммирование $\sum_{j}^{n} f_j(x_i)$ происходит одновременно с вычислением следующей порции $f_j(x_{i+1})$ $(j = 1, \ldots, n)$. А так как $\tau_{max} \gg \tau^{\Sigma}$, то именно эти вычисления лимитируют время.

Конвейер вычислений

Идея распараллеливания применима там, где есть независимые друг от друга участки программы — именно их выполнение и запараллеливают. Но часто про-

грамма не имеет таких участков или они очень малы. Можно ли здесь убыстрить процесс вычислений без увеличения быстродействия процессора? Если такая программа выполняется лишь один раз, то ничего не поделаешь и надо отдать ей все требуемое время. Однако при массовых расчетах, когда одна и та же программа выполняется многократно, можно сократить общее время вычислений. Именно в этом распространенном случае можно воспользоваться идеей конвейера, позволяющей значительно убыстрить процесс вычислений.

Давайте вспомним, чем знаменит конвейер. Вовсе не распараллеливанием, хотя его здесь и можно использовать. Конвейер замечателен другим — разбиением всего процесса изготовления (обычно сборки) на отдельные элементарные операции, выполнение которых возлагается на технологические участки, мимо которых движется конвейер. Чем меньше элементарная операция, тем производительнее конвейер. Действительно, готовая продукция будет сходить с конвейера с интервалом, равным времени одной сборки, т. е. за время одной элементарной технологической операции (все такие операции должны требовать одного и того же времени — это условие конвейера) плюс время передвижения от одного участка к другому. Именно поэтому элементарные операции должны быть очень простыми, а скорость движения конвейера — высокой. (Читатели старшего поколения помнят известную кинокомедию Чарли Чаплина «Новые времена», где элементарной операцией на конвейере сборки автомобилей был один поворот гайки). Число рабочих мест при этом увеличивается вместе с длиной конвейера.

А теперь применим идею конвейера к решению задач, программа которых должна многократно повторяться. Разобьем всю программу A на одинаковые по времени реализации части — подпрограммы (элементарные операции) $A_1, A_2, \ldots, A_N$, последовательное выполнение которых решает поставленную задачу. Теперь N процессоров запрограммируем на решение этих подпрограмм и расположим их в цепочку так, что результат работы k-го процессора будет исходными данными $k+1$-го (см. рис. 10).

Пусть $x_1, \ldots, x_k$ исходные данные k задач, которые должны быть решены программой A k-кратным ее повторением, а $R_1, \ldots, R_k$ — результаты расчетов этих k

задач, которые необходимо получить. Очевидно, что подавать на вход такой цепочки процессоров исходные данные $x_1, \ldots, x_k$ следует не дожидаясь, пока будет получен конечный результат, а через время, равное T/N, где T — общее время решения одной задачи на одном процессоре. В этом случае, как легко заметить, в любой момент времени каждый процессор будет решать свою задачу (если первый — i-ю, то второй — $(i-1)$-ю, третий — $(i-2)$-ю, ..., N-й — $(i-N)$-ю, т. е. одновременно будет решаться N задач (точнее: задача одна, но с N исходными данными).

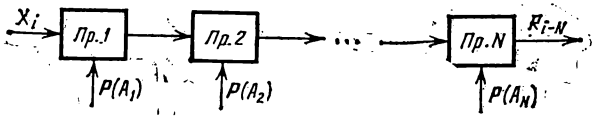

Рис. 10. Так реализуется конвейер вычислений. Здесь важно, чтобы время, затрачиваемое на вычисление отдельных «кусков» программы, было бы одинаковым. В противном случае время вычисления одной реализации будет равно времени вычисления максимального «куска».

Легко подсчитать производительность такого конвейера. Время решения k задач на нем равно $(k+N)\frac{T}{N}$. Это значительно меньше по сравнению с производительностью одного процессора, который затратит на это Tk, т. е. примерно в N раз меньше (для $k \gg N$). Таким образом, чем больше процессоров образуют «конвейер» обработки информации, тем быстрее обрабатывается информация.

При этом следует помнить, что процедура образования той или иной конфигурации вычислительной системы (параллельной, как на рис. 8, или последовательной, как на рис. 10, и любой другой) происходит практически мгновенно. Ведь обмен информацией между процессорами происходит через оперативную память системы и управляющая программа вычислительной системы определяет ее конфигурацию путем изменения адреса, по которому должна поступать пришедшая информация. Так же просто перепрограммируются процессоры — их программы хранятся в памяти системы (или в собственной памяти этих процессоров при их работе) и по команде управляющей программы любая

из программ может быть реализована в любом из процессоров.

Эта удивительная гибкость многопроцессорных вычислительных систем и дает возможность чрезвычайно быстро обрабатывать информацию любого вида. Действительно, если на такую вычислительную систему приходит набор малых задач, то каждой из них будет выделен отдельный процессор (а оставшиеся — поставлены в очередь), таким образом, чтобы время решения всех задач было бы минимальным. Эта минимальность будет обеспечена автоматически за счет их одновременного выполнения на всех процессорах системы.

Если же на такую многопроцессорную вычислительную систему поступит большая задача, то сначала система сделает анализ структуры на предмет ее распараллеливания и использования конвейера (анализ осуществляется довольно просто — путем выяснения связей между отдельными частями программы и их повторяемостью). В результате предлагается конфигурация вычислительной системы для решения поступившей задачи. Эта конфигурация представляет собой план решения задачи, т. е. какие ее части на каких процессорах и когда будут решаться. Реализация плана и решает поставленную задачу. Минимальность времени ее решения будет гарантироваться максимальной загрузкой всех процессоров вычислительной системы, о чем и заботится операционная система при составлении плана решения каждой задачи.

Как видно, из имеющихся процессоров образуется некоторая условная вычислительная машина, специализированная для решения именно заданной задачи (и никакой другой). Такую машину называют виртуальной, т. е. условной. Многопроцессорная вычислительная система и отличается тем, что она способна для каждой решаемой задачи или комплекса задач очень быстро создавать такие виртуальные машины, решать с их помощью поставленные задачи, быстро менять конфигурацию виртуальной машины и снова решать на ней поступившие задачи и т. д.

Именно таким образом были достигнуты вычислительные мощности в 300—800 миллионов операций в секунду на многопроцессорной вычислительной системе. А что дальше?

Пастбище для ... процессоров

Когда зашел разговор о перспективах развития многопроцессорных вычислительных систем, то один из ведущих специалистов в области вычислительной техники проф. З. Л. Рабинович не без юмора сказал, что он видит эту перспективу как огромное поле памяти на котором, как коровы, пасутся процессоры. Большое стадо процессоров.

На этом поле буйным цветом растут задачи, которыми «питаются» процессоры. Есть и пастух на поле — это управляющая программа, которая «сгоняет» процессоры по месту их кормежки да так, чтобы всем хватило, чтобы они не мешали друг другу и даже... помогали.

Эту живописную картину рискованно назвать аналогией, например, при интерпретации режима конвейера. Это просто метафора, показывающая важность большого поля памяти и большого числа процессоров для будущих сверхвысокопроизводительных вычислительных систем. Так, наверное и будет.

Об адаптации вычислительных систем

Как уже говорилось в первой главе процесс адаптации представляет собой приспособление к новым условиям. В вычислительных системах такими условиями являются состояние объекта, связанного с вычислительной системой, состояние элементов самой вычислительной системы и вид критерия оптимальности, т. е. представление о наилучшей вычислительной системе, задаваемое извне.

Нетрудно заметить, что описанные методы перестройки структуры вычислительной системы в зависимости от ее состояния и приспособления ее к состоянию среды (заявок пользователей) являются ничем иным, как адаптацией вычислительной системы к возникающим непредвиденным изменениям в среде и в самой системе.

Как видно, преимущества вычислительных систем связаны в основном с тем, что используются адаптивные приемы. Действительно, эффективная работа системы разделения времени определяется прежде всего дисциплиной обслуживания, адаптирующей систему к запросам пользователей. При этом коротким задачам предоставляется высший приоритет, и они решаются

быстрее, чем длинные, которые решаются во вторую очередь. Адаптация в данном случае заключается в том, что дисциплина обслуживания реализует свои функции, не располагая информацией о том, какой длины задача поступила в вычислительную систему. Именно для этого нужна адаптация.

Повышение живучести вычислительной системы путем «горячего» (т. е. постоянно находящегося в рабочем состоянии) дублирования ее элементов и использования работоспособного блока — является типичной адаптацией вычислительной системы к ее собственному состоянию, точнее — к состоянию ее элементов. Такая адаптация характерна для систем, которые обычно называют самонастраивающимися. Здесь самонастройка дает возможность вычислительной системе сохранять свою работоспособность независимо от того, что отдельные ее агрегаты в неизвестное заранее время выходят из строя.

И наконец, «цифродробители» добиваются своей цели, используя типичные приемы адаптации. Действительно, и распараллеливание и конвейер вычислений могут быть использованы не вообще, а лишь применительно к конкретной структуре алгоритма решения задачи. Процесс создания схемы многопроцессорной вычислительной системы — это прежде всего процесс приспособления, адаптации к структуре выполняемой программы; успех здесь определяется прежде всего тем, насколько удачно удастся распараллелить и «расконвейерить» решаемую задачу, т. е. насколько хорошо удастся адаптироваться к программе решения задачи.

Этим разумеется не ограничивается применение методов адаптации к вычислительным системам. Адаптация здесь может значительно повысить их эффективность за счет учета более тонких свойств пользователей или агрегатов системы. Так, например, в процессе дублирования адаптация может помочь в выборе и включении тех агрегатов, которые наилучшим образом удовлетворяют требованиям надежности вычислительной системы в целом и т. д.

Глава 3

ВЫЧИСЛИТЕЛЬНЫЕ СЕТИ

«Ах, попалась, птичка, стой!
Не уйдешь из сети...»

Песенка

Мегрэ в затруднении

Операция ограбления проходила вполне гладко, если не считать нескольких шероховатостей, которые вроде бы не должны были оказать существенного влияния на ее исход — фактор неожиданности в виде определенного запаса времени был учтен в плане его проведения. А этот запас еще не был исчерпан. Пока слитки золота и пачки денег складывали в мешки (драгоценностей не брали — слишком много возни с реализацией), Джек внимательно оглядывал операционный зал банка и искал зрачки фотоавтоматов. О трех из них он знал заранее и обезвредил (с этого и началась «работа» в банке). Но нет ли еще? «И чего бояться этих клопов? Ведь маска надежно скрывает лицо!» — недоуменно подумал Джек.

А в это время в ближайшем полицейском участке уже была объявлена тревога. Дежурный внимательно следил за телевизионным изображением, передаваемым из операционного зала. Рядом работал видеомагнитофон, фиксирующий все происходящее в банке.

Телевизионный датчик был установлен в люстре, и никто в банке, кроме его президента и его трех «вице» не знал об этом. Размером с карандаш он висел на люстре вместе с другими «висюльками» и почти ничем не отличался от них. Датчик работал непрерывно днем и ночью, посылая изображение в ЭВМ, установленную поблизости. Машина анализировала изображение и классифицировала его как «нормальное» или «ненормальное». В первом случае ничего не следовало, а во втором — изображение вместе с сигналом тревоги передавалось в ближайший полицейский участок, где дежурный должен был принять решение — ограбление это или случайное стечение обстоятельств в зале, классифицированное ЭВМ как ненормальное. Признаков «не-

нормальности» много — быстрое движение людей, их нестандартное расположение, не говоря уж о выстрелах и криках, которые также фиксировались ЭВМ.

На сей раз сработала нестандартность ситуации (у каждого окошка и каждой двери по человеку).

Дежурный вызвал Мегрэ и подготовил все необходимое: наряд полиции с автоматами уже находился в машине, ворота гаража были открыты, движение по дороге к банку остановлено, а на столе лежали план банка и несколько фотографий с телеэкрана, на которых цифрами указан момент съемки с точностью до десятых долей секунды.

Мегрэ, взглянув на экран сразу оценил ситуацию:

— Дать отбой тревоги! — и на недоуменный взгляд дежурного пояснил. — В зале слишком много народу. Видите, на полу лежит несколько человек — они будут первыми жертвами как заложники.

В это время один из стоявших в стороне бандитов поднял автомат и... яркая вспышка была последним, что зафиксировал экран. Раздался резкий звонок из префектуры — звонил полковник.

— Какого черта, Мегрэ, вы не действуете? Пора шевелиться! Бандиты уже более сорока секунд хозяйничают в банке.

— Я отменил тревогу, — сухо ответил Мегрэ.

— Вы с ума сошли! Немедленно отправить наряд к банку или я это сделаю сам.

— И возьмете на себя жертвы среди населения, которые неизбежны и в банке и при погоне! — резко ответил Мегрэ. — Тут пахнет десятками трупов ни в чем неповинных людей, и прежде всего нерасторопных — детей, стариков и женщин!

— Что же вы намерены делать? — снизил тон полковник.

— Искать преступников! Для этого у меня достаточно материала, — ответил Мегрэ, похлопывая по ящику видеомагнитофона, который еще продолжал работать.

— Каким образом?

— Заставлю потрудиться компьютер и досье, которое нынче называют банком данных, — ответил Мегрэ.

— Ну что ж, банкуйте.— (Полковник был картежником и любил каламбуры.) — Только помните, что наверняка эти ребята, если вы их поймаете, имеют прекрасные алиби и вам будет очень трудно доказать, что на видеопленке именно они.

— Искать и доказывать буду не я, а компьютер.

— Неплохо бы при этом и вместо присяжных иметь компьютеры, — съязвил полковник. — Неужели вы не понимаете, что ваша карьера висит на волоске?— И строго официальным тоном продолжил:— Инспектор Мегрэ, если через два часа вы не предъявите мне кого-либо из преступников (я считаю, что «старым способом», т. е. хорошо организованной погоней, двух часов достаточно, хотя трупы, разумеется, неизбежны), то вам будет предъявлено обвинение в сговоре с преступниками, так как только этим можно объяснить вашу бездеятельность. Все! Через два часа жду вашего сообщения.

Мегрэ усмехнулся и положил трубку.

— Ну, старушка ЭВМ, выручай!

Здесь мы оставим на время нашего героя «в минуту злую для него», остановим бег детектива и обсудим некоторые важные детали. Прежде всего об описанной системе оповещения в банке. Не слишком ли она сложна? Автор снабдил ее специальным компьютером, который, анализируя ситуацию, принимает решение, передавать изображение дежурному в полицейском участке или нет, т. е. поднимать тревогу или молчать. Не проще ли посадить за телевизионный экран человека, который бы и нажал кнопку в нужный момент?

Действительно, это проще. Но уж больно накладно, ведь человеку надо платить зарплату, и крайне ненадежно. Опыт показывает, что человек, единственной обязанностью которого является фиксирование редкой аварийной ситуации, как правило, пропускает именно ее. Монотонность нормы ослабляет чувствительность к авариям. Так что решение банка о создании компьютерной системы оповещения было правильным. Ей не надо платить ежемесячную зарплату, а надежность можно сделать довольно высокой, если подозрительные нормальные ситуации относить к ненормальным и тем самым подключать человека (дежурного полицейского) к решению проблемы: грабят банк или нет. Он-то разберется! Правда, при этом ложных тревог будет много, что неизбежно несколько снизит внимательность дежурного. Но не столь сильно, чтобы пропустить ограбление.

Как компьютер определяет ненормальные ситуации? Проще всего задать эти ситуации. Всякое изображение кодируется набором цифр, характеризующих степень яркости каждой из его точек. Например, нуль характе-

ризует черную точку, а единица — белую. Тогда $^1/_2$ определяет серую точку и т. д. Каждая точка изображения на телевизионном экране таким образом характеризуется тремя числами: i — номером строки, j — номером столбца и яркостью, которую обозначим числом a_{ij}. Это число лежит в пределах $0 \leqslant a_{ij} \leqslant 1$ и определяет яркость точки с координатами (i, j).

Как видно, всякое черно-белое изображение можно представить в виде таблицы или, как говорят в математике, матрицы чисел

$$A = \begin{Vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{Vmatrix},$$

где a_{ij} — элемент i-й строки и j-го столбца матрицы, которая имеет всего n столбцов и m строк. (Заметим, что для советской системы телевидения принят стандарт $n = 800$, $m = 625$, т. е. изображение имеет около 500 000 — полмиллиона точечных элементов.)

Матрицу A, полученную с помощью датчика телевизионного изображения, можно легко ввести в память ЭВМ. Так же легко ввести матрицы изображений, на которые ЭВМ должна реагировать тревогой. Это «грабительские» изображения, которые должны быть заранее изготовлены и введены в память ЭВМ. Пусть одна из таких матриц ограбления имеет вид

$$B = \begin{Vmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ \dots & \dots & \dots & \dots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{Vmatrix}.$$

Здесь b_{ij} определяет яркость точки, лежащей на пересечении i-й строки и j-го столбца эталонного изображения ограбления. Очевидно, что сопоставляя эти матрицы, можно определить, совпадают они или нет. Если

$$a_{ij} = b_{ij} \qquad (*)$$

для всех i и j, что легко проверит ЭВМ, то матрицы совпадают и ситуацию в банке следует считать ограблением, так как действительное изображение (A) и эталонное (B) совпадают.

При несовпадении всех элементов матриц сказать ничего нельзя. Действительно, ситуация A может незначительно отличаться от ситуации B, что не изменит

ее «грабительской» сути, но условие (*) не будет выполняться. Именно поэтому следует ввести число Q, характеризующее степень несовпадения матриц. Для этого достаточно просуммировать модули разностей

$$Q = \sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij} - b_{ij}|,$$

т. е. определить степень несовпадения яркостей в каждой точке и сложить их все. Величина Q характеризует степень отличия изображений. Если они совпадают, то $Q = 0$, а для полностью несовпадающих изображений, когда все белые точки поменялись на черные и наоборот, а серых нет в одном изображении, $Q = nm$. Величину Q легко и быстро подсчитает компьютер.

Теперь его решение удобно реализовать с помощью такого решающего правила:

если $Q \ll \delta$, то $A \approx B$, и надо объявлять тревогу;

если $Q > \delta$, то $A \not\approx B$, и грабежа нет.

Хорошо видно, что решение компьютера сильно зависит от величины δ (ее обычно называют порогом). При $\delta = 0$ тревога будет объявлена только при полном совпадении изображения, т. е. при выполнении условия (*), что бывает крайне редко. Это плохой выбор порога.

При большом пороге δ тревоги чаще всего будут ложными. Но среди этих тревог почти наверняка будут и правильные, когда грабеж идет во всю. При уменьшении δ число ложных тревог сокращается, но при этом и появляется риск пропустить грабеж.

Вот и получается, что величина δ должна быть выбрана оптимальной, т. е. не слишком малой, но и не слишком большой. Выбор такого оптимального $\delta = \delta^*$ и составляет проблему оптимальных решающих правил. Эта величина δ^* зависит от стоимости ошибок первого рода (пропуск грабежа) и ошибок второго рода (ложная тревога). В обоих случаях система наблюдения приносит неприятности: в первом — ограбление, а во втором — беспокойство дежурного в полиции, за которые тоже надо платить. Выбор оптимального порога δ^* должен минимизировать суммарную плату.

Легко видеть, что компьютер, снабженный набором аварийных («грабительских») ситуаций $B_1, B_2, \ldots, B_N$ должен каждое из них сравнить с получаемым из операционного зала, да еще делать это для каждого мо-

мента времени. Так что работы ему предостаточно. Именно здесь используется основное качество компьютера — его быстродействие. Величину Q он вычисляет за доли секунды, а на принятие решения потребуется значительно меньше времени.

С этой задачей, задачей сопоставления изображения операционного зала с изображением, хранящимся в памяти, ЭВМ справляется, как видно, легко.

Теперь рассмотрим, каким образом ЭВМ может определить появление бегущего человека в зале, что также характеризует «ненормальность» ситуации. Для этого можно воспользоваться тем же способом сопоставления матриц, который был описан выше.

Дело в том, что телевизионный датчик изображения посылает в ЭВМ 25 изображений в секунду. Обозначим матрицы этих изображений в виде

$$A(t)=\left\| \begin{matrix} a_{11}(t) & \dots & a_{1n}(t) \\ \dots & \dots & \dots \\ a_{m1}(t) & \dots & a_{mn}(t) \end{matrix} \right\|,$$

где буквой t обозначена зависимость матрицы, а следовательно, и изображения от текущего момента времени. Действительно: ситуация в операционном зале изменяется и поэтому

$$A(t_1) \neq A(t_2),$$

где t_1 и t_2 — какие-то два следующие друг за другом момента времени. Если отличие этих матриц невелико, то повода для беспокойства нет, так как это означает, что ситуация изменяется медленно — все ходят чинно, не размахивают руками и т. д. Если же матрицы $A(t_1)$ и $A(t_2)$ отличаются сильно, то это означает, что происходит быстрое изменение изображения — кто-то побежал или взмахнул руками и т. д., что пахнет ограблением.

Поэтому достаточно вычислять степень несовпадения матриц двух следующих друг за другом изображений. Например, по формуле

$$Q(t_2)=\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}(t_1)-a_{ij}(t_2)|,$$

аналогичной рассмотренной выше. Чем больше величина Q, тем сильнее отличаются матрицы, а следовательно,

и изображения. Очевидно, что тревогу надо поднимать при

$$Q(t) \geqslant \varepsilon,$$

где ε — некоторое пороговое значение. Если такой порог выбрать малым, то часто тревога будет ложной, так как незначительное изменение ситуации будет вызывать тревогу (например, достаточно кому-нибудь уронить шляпу). Если же порог ε велик, то имеется риск пропустить ограбление, если бандиты не будут бегать сразу все по операционному залу. (Это все те же упомянутые выше ошибки второго и первого рода.)

Именно поэтому величина порога ε должна быть выбрана не малой и не большой, а оптимальной

$$\varepsilon = \varepsilon^*,$$

при которой минимизируется ущерб, вызванный как ложной тревогой, так и пропущенным ограблением.

Здесь следует помнить, что резкое изменение изображения произойдет, если в операционном зале погаснет или будет включена хотя бы одна лампа. Тревога при этом неизбежна. Поэтому следует предусмотреть такую возможность, что неизбежно усложняет решающее правило. Но суть его остается прежней — следить и фиксировать резкие изменения изображения (независимо от изменения освещенности отдельных его участков).

Для реализации описанного решающего правила в каждый момент времени t достаточно иметь в памяти ЭВМ две матрицы $A(t-\Delta)$ и $A(t)$, где $t-\Delta$ — предыдущий момент, и заменять 25 раз в секунду так, чтобы всегда можно было определить величину различия двух следующих друг за другом изображений. Для этого нужно иметь ЭВМ с весьма высоким быстродействием. Подсчитаем его.

Для вычисления величины Q надо сделать nm вычитаний, столько же вычислений модуля и сложений (не считая $2mn$ обращений к памяти). Будем считать, что на каждую такую операцию затрачивается время τ с. Тогда на вычисление одного значения Q будет затрачено времени

$$T = (\tau + \tau + \tau)\, nm = 3\tau nm.$$

Так как телевизионный датчик дает 25 изображений в секунду, то для того, чтобы машина справилась с поставленной задачей, необходимо, чтобы

$$25T \leqslant 1.$$

Подставляя снова выражение для T получаем требование на время выполнения одной операции сложения (или вычитания):

$$\tau \leqslant \frac{1}{75nm} [\text{с}].$$

Теперь не трудно подсчитать для советского стандарта (и французского тоже):

$$\tau \leqslant \frac{1}{75 \cdot 500\,000} \approx 2{,}7 \cdot 10^{-8} \text{ с}.$$

Это соответствует быстродействию — 37,5 миллионов операций в секунду! На такое способен далеко не всякий компьютер. И это только для сравнения двух следующих друг за другом изображений! А если к этому добавить необходимость сравнивать каждое пришедшее изображение с хранящимися в памяти (а их может быть до ста), то быстродействие ЭВМ должно возрасти до трех миллиардов операций в секунду. На это сейчас способны лишь редкие модели ЭВМ, точнее вычислительных систем.

Можно снизить столь жесткие требования, например, уменьшив число передаваемых изображений до 2—3-х в секунду, да число элементов матрицы до 50 000 (в ущерб разрешающей способности изображения, разумеется). Это дает возможность обойтись ЭВМ с 30 миллионами операций в секунду, что немногим облегчает проблему.

Поэтому банку эта система оповещения обошлась в довольно круглую сумму. Мегрэ хорошо знал это и понимал, как будут реагировать банкиры. Но система оповещения сделала свое дело и дело немалое: почти тридцать метров видеопленки, на которой зафиксированы бандиты «в работе». Да, именно их «работа» интересовала Мегрэ. Фотоавтомат с его десятком снимков не мог дать и тысячной доли той информации, которую ждал Мегрэ. И он начал действовать.

Просмотрев видеопленку, Мегрэ лишь убедился, что простое увеличение ничего не решит. Все бандиты были в одинаковых масках, плащах и перчатках.

— И, как всегда, никаких особых примет, — констатировал Мегрэ.— Никто не отличается от других — словно роботы. Их шеф хорошо понимает, что в таких случаях нельзя обладать индивидуальностью.

— Поль! — обратился Мегрэ к дежурному.— Заведи пленку в компьютер и попробуй меня не беспокоить пару часов.

— Слушаю, месье, — ответил дежурный и начал колдовать с видеомагнитофоном.— Через минуту можно будет с ней работать. А что вы намерены делать, месье?

— Еще не знаю точно. Но попробую их идентифицировать.

— Но ведь они все были в масках!

— А вот изменить свои скелеты они не могли,— задумчиво заметил Мегрэ.

Лицо дежурного вытянулось. Ему представилась мрачная картина: Мегрэ обмеряет линейкой скелеты бандитов, разложенные на столах, и записывает результаты в блокнот.

— Но ведь они... — он поперхнулся и, широко улыбнувшись, сказал: — Вы шутите, инспектор!

— Нисколько! — мрачно сказал Мегрэ и пошел в свой кабинет.

Здесь он сразу связался с профессором Мариаком из Гренобля. Между ними произошел следующий разговор.

— Здравствуйте, профессор. Говорит инспектор Мегрэ. Если помните, не так давно я на конференции по идентификации измучил Вас вопросами об определении параметров организма по киносъемкам его движений.

— А, хорошо помню Вас, дорогой Мегрэ. Меня тогда очень удивило, что знаменитый детектив интересуется чисто научной проблемой.

— Это был сугубо профессиональный интерес, профессор. Сейчас я могу об этом сказать. Меня интересует, сможет ли Ваша программа определить параметры организма человека, ну там особенности скелета, мускулатуры и всего прочего по видеопленке, где зафиксирована его работа?

— Вы же знаете, Мегрэ, что я занимаюсь научной работой, а ловля преступников — ваша задача. Поэтому боюсь, что не смогу быть Вам полезен.

— Погодите, профессор, я обращаюсь к Вам не как частное лицо, а от имени Интерпола. Речь идет об обычной экспертизе, которую мы просим Вас провести с помощью Вашей программы. И наконец, Ваш гонорар сможет вполне поправить финансовые дела Вашей лаборатории, о чем Вы, помнится, жаловались мне на конференции. Считайте, что я хочу Вам помочь, — улыбнулся Мегрэ.

Вздох, который он услышал на другом конце провода, убедил его, что он попал в точку.

— А Вы уверены, что этот человек, которого нужно идентифицировать, преступник? — въедливо спросил профессор Мариак.

— Вы сами убедитесь в этом, когда я перешлю Вам видеозапись. Это обычное ограбление банка.

— Ну, если это обычное ограбление, то почему Вы, дорогой инспектор, не воспользуетесь обычными средствами, ну там стрельбой, погоней и прочими аксессуарами полицейских романов? — язвительно заметил профессор.

— Обычные средства приводят к обычным последствиям,— твердо сказал Мегрэ.— А они нежелательны, — подумал он и тихо добавил. — Ведь речь идет о людях, не имеющих никакого отношения к преступлению. Подумайте о них, профессор!

— Немедленно расскажите мне о Вашей задаче! — запальчиво закричал Мариак.

Мегрэ коротко изложил суть дела. Надо было определить особенности в поведении, скелете, мускулатуре и другие особые приметы семерых бандитов, зафиксированных на видеопленке, с тем, чтобы по этим приметам найти их, используя данные, хранящиеся в полицейском досье.

— Ну что ж,— задумчиво произнес профессор Мариак.— Я думаю, что месяца через два мы сможем Вам дать полный перечень параметров скелета и мускулатуры, а заодно и жирового слоя Ваших клиентов.

— Эти данные мне нужны через два часа,— сухо заметил Мегрэ.

— Вы с ума сошли! О каких часах может идти речь, если используется исследовательская программа. Сама по себе задача идентификации организма имеет огромную сложность. Ведь придется минимизировать функционал по нескольким тысячам переменных! Вы соображаете, сколько понадобится сделать шагов поиска?

На время оставим наших героев и поясним мысль, которую столь сбивчиво высказал профессор Мариак. Речь здесь идет о построении модели человека, поведение которой не отличалось бы от поведения моделируемого человека, а точнее — его изображения на видеопленке. Эта модель характеризуется большим числом параметров, определяющих ее геометрию и динамику. Она-то и является предметом гордости профессора. Задавая этой модели различные параметры и программы, можно видеть на экране дисплея ее поведение. Если теперь сравнить поведение реального человека (объекта) на видеопленке и поведение модели, то они будут отличаться. Степень этого отличия и называют функционалом невязки. Его значение зависит от параметров и ее программы движения. Если подобрать такие параметры и программу, чтобы функционал был минимален (а в идеале — равен нулю), то эти параметры стали бы параметрами объекта, а программа — программой его поведения. Такова в общих чертах идея.

Для ее реализации нужно минимизировать функционал невязки, зависящий от огромного числа параметров модели (это параметры скелета, мышечных тканей, жировой ткани и т. д.) и программы ее движения, характеризующейся распределением мышечных усилий при выполнении различных движений модели.

Если вычисление одного значения функционала невязки отнимает несколько минут машинного времени (таков удел экспериментальных программ, которые сделаны с большим запасом — именно на это сослался профессор Мариак), а для минимизации нужно сделать по крайней мере столько шагов поиска, сколько переменных (а обычно значительно больше), то отсюда и следуют упомянутые два месяца непрерывной работы мощной ЭВМ.

Но вернемся к нашим героям.

— Профессор,— взмолился Мегрэ.— Нет у меня двух месяцев. Может быть можно определять не все параметры, а фиксировать лишь отклонения от нормы — по ним мы легко найдем преступников в досье.

— Нет, это не пройдет, — уныло ответил профессор.— Сначала надо определить все параметры, а уж затем искать отклонения от нормы. Хотя это отклонение найти уже легче — не нужно до конца минимизировать функционал невязки.

— Ну и какое время Вам для этого понадобится?

— Один месяц!

Мегрэ задумался. Неужели из-за экспериментальности этой программы сорвется дело? Ну, если она слишком долго считает, то нельзя ли вычисления раскидать по разным машинам?

— Алло, профессор, а Ваша программа допускает распараллеливание вычислений?

— Разумеется. Ведь невязка суммарная! И одну ее часть можно вычислять на одной машине, другую — на другой и так далее. А потом результаты сложить.

— Тогда давайте возьмем (Мегрэ быстро прикинул на бумаге) 500 ЭВМ, и они справятся с этой задачей за два часа.

— Да чего там! Берите сразу тысячу! А там разберемся! — ехидно заметил профессор Мариак. — Извините, инспектор, меня ждут дела. До свидания! Звоните, если придумаете что-нибудь реальное, — и повесил трубку.

Мегрэ задумался. Как использовать тысячу машин для решения одной задачи? В голову пришла старая шутка о том, что, если собрать девять беременных женщин вместе, то все равно младенец не появится через месяц. Он уже слышал о вычислительных сетях и даже как-то раз воспользовался одной из них, когда получал из США досье на одного подозреваемого. Это был обычный запрос к банку данных по линиям связи. Но можно ли вычислять сразу на всех машинах в сети? Здесь нужна была консультация и он повернулся к терминалу. Набрал номер справочной службы и слово ВЫЧИСЛИТЕЛЬНЫЕ СЕТИ, подумал и добавил знак вопроса. И тут же на экране появился текст...

Оставим на время нашего инспектора и разберемся, что представляют из себя современные вычислительные сети, почему они появились и куда идут в своем развитии. А потом вернемся к Мегрэ (опытный читатель, поднаторевший в детективах, наверняка понял, что именно вычислительные сети дадут ему возможность найти шайку, только что ограбившую банк).

Чем плохи вычислительные системы?

Логика развития, определяемая в основном потребностями человечества, привела к появлению многопроцессорных вычислительных систем, которые по

сравнению с ЭВМ стали шагом вперед и весьма ощутимым. Именно такие системы позволили преодолеть миллионный барьер для скорости вычислений и приблизиться к миллиарду. Именно с помощью вычислительных систем удалось создать живучие и надежные системы реального времени для управления и обработки информации.

Слов нет, всем хороши вычислительные системы, но... и их возможности не беспредельны. Так, например, есть задачи, требующие для своего решения в приемлемое время вычислительные мощности с миллиардом операций в секунду (это, например, те самые задачи с экспоненциальной трудоемкостью решения, которую не удается понизить до полиномиальной). Реальным примером такой задачи является задача составления оптимального государственного плана выпуска продукции. Ни одна современная (да и будущая) вычислительная система не возьмется за ее решение. Ей, вычислительной системе, эта задача не по зубам!

И это лишь один недостаток вычислительных систем. Есть и другие! Попробуем перечислить основные.

Прежде всего — ограниченные возможности. Всякая многопроцессорная вычислительная система, сколь мощной она ни была бы, всегда имеет пределы по быстродействию, памяти и надежности. Эти пределы могут быть очень высоки, но они есть, а поэтому всегда можно указать задачу, которую не сможет решить данная вычислительная система. К сожалению, именно такие задачи все чаще и чаще появляются в науке, технике и народном хозяйстве. Аппетит человечества всегда «приходит во время еды», т. е. потребления предоставленных возможностей. Ему всегда нужно больше, чем дано. В этом, по-видимому, и кроется секрет прогресса.

Так или иначе, наличных вычислительных мощностей всегда не хватает и надо иметь возможность быстро их наращивать настолько, насколько нужно. Вычислительные системы не могут обеспечить этого требования.

Не обладают они и другим очень важным для пользователей качеством — предоставлять возможность доступа к информации, хранящейся на других машинах или вычислительных системах. Легко представить себе, что вам понадобилась программа решения задачи, разработанная вашим коллегой, находящимся на другом

континенте (например, в Гаване). Если понадобилась срочно, то ... смирите свои желания: лучшее, что вы можете сделать, это запастись терпением и попросить его прислать вам почтой пакет перфокарт или магнитную пленку с программой и описанием к ней. Перипетии, которые придется претерпеть на почте посылке (удары при перегрузках, вибрации при транспортировке и т. д.), наверняка обесценят пересылаемую программу и нечего удивляться, что программа не будет работать на вашей машине. Нужно посылать не саму программу, а ее текст.

Именно поэтому образуются всякого рода фонды и банки программ, которые рассылают заинтересованным лицам фотокопии текстов программ. Но опыт показывает, что эффективность таких банков невелика, так как программы, составленные по присланным текстам обычно не работают. Причина этого тривиальна— это ошибки или в тексте программы, допущенные разработчиком при ее сдаче в фонд, или в самой программе, сделанные заказчиком при ее перенесении на перфоленту или перфокарту. И ту и другую ошибку выявить трудно. Здесь срабатывает известный всем программистам закон: легче написать и отладить новую программу, чем найти ошибку в чужой. Поэтому фондом программ часто пользуются как справочным бюро, с тем, чтобы найти разработчика нужной программы. А уж уговорить его приехать и поставить свою программу на вашей машине — это вопрос вашей напористости, обаяния и умения «выбивать» нужные средства из дирекции.

И все эти неудобства и мытарства приходится терпеть из-за отсутствия возможности прямо выходить на удаленную от вас машину или систему, в памяти которой находится интересующая вас программа или какая-либо другая информация.

Оба описанных неудобства вычислительных систем — ограниченная мощность и недоступность других ЭВМ — могут быть преодолены путем создания вычислительной сети или сети из вычислительных машин и систем, т. е. путем их соединения каналами связи (их довольно пышно называют коммуникационными каналами).

Вычислительные сети, образованные путем соединения ЭВМ и вычислительных систем каналами связи,

часто называют системами распределенной обработки данных. На это есть веские основания.

Дело в том, что возможность решать задачу сразу на нескольких ЭВМ или системах, связанных в вычислительную сеть, и реализует процесс распределенной обработки данных. Он открывает большие возможности пользователям. Действительно, процесс обработки информации на ЭВМ подразумевает не только вычисления, но и обращение к ее памяти, где может находиться ценная для пользователя информация. Так, например, желая решить какую-то сложную задачу из области ядерной физики, ее следует направить по сети на ЭВМ, в памяти которой имеется банк данных по этой проблеме. Тогда вся необходимая информация будет получена прямо из этого банка данных.

Можно поступить и иначе — решать задачу на своей ЭВМ, а требуемую для нее информацию запрашивать из соответствующего банка данных. И все это можно делать в автоматическом режиме со своего терминала. Без писем и звонков по телефону. Такая оперативность при решении сложных задач возможна только при использовании вычислительных сетей.

Идея объединения ЭВМ и вычислительных систем в вычислительную сеть возникла сравнительно недавно и первая такая сеть была запроектирована в США в результате... запуска в СССР первого искусственного спутника Земли — нашего спутника.

Бип-бип!

Когда 4 октября 1957 г. Советским Союзом на орбиту был выведен первый искусственный спутник Земли, весь мир был об этом извещен знаменитым «бип-бип», которое спутник непрерывно передавал по радио. Это невинное «бип-бип» было болезненно воспринято в США и в противовес «этим русским», кроме прочего, был выдвинут проект создания первой большой вычислительной сети. Назвали этот проект ARPA (аббревиатура слов Advanced Reserch Project Agency, т. е. управление перспективных исследований, которое подчинено министерству обороны США). Эта сеть, названная ARPANET, т. е. сеть ARPA, вступила в строй в 1969 г.

На эту сеть возлагались большие надежды, сводились они к следующему. Так как «интеллектуальность»

компьютера связана с его мощностью, то, объединяя с помощью сети много компьютеров в один электронный «мозг», можно рассчитывать получить очень умного электронного стратега, которого уж перехитрить никто не сможет.

Ошибочность этих рассуждений стала ясна только теперь, когда хорошо выяснены возможности ЭВМ, т. е. что она может, а чего (может быть, пока) —нет.

Так или иначе, но работа над проектом ARPA началась и была поддержана ведущими специалистами по вычислительной технике, которые видели в такой интеграции вычислительных мощностей новый шаг развития и использования вычислительной техники.

Проект ARPA имел ко всему прочему и другие, более важные, цели — интегрируя ЭВМ, удовлетворить те потребности пользователей, которые в принципе не могли удовлетворить существующие вычислительные машины и системы из-за их ограниченной мощности и невозможности обращения одной ЭВМ или вычислительной системы к памяти другой.

Как видим, понимание необходимости создания вычислительных сетей, как средства объединения вычислительных ресурсов, распределенных на большой территории (и даже на поверхности всей Земли) возникло уже давно. Но несмотря на огромные средства, выделенные правительством США на создание первой вычислительной сети, она заработала лишь через 12 лет!

Естественно задать вопрос, а какие трудности возникают при создании вычислительной сети и окупаются ли они при ее эксплуатации?

Что нужно для вычислительной сети?

Прежде всего вычислительные машины и системы, а также каналы связи между ними. Но и то и другое уже имеется!

Действительно, парк вычислительных машин сейчас настолько велик и разнообразен, что их можно встретить повсюду. Труднее найти место, где их нет. Итак, проблемы с вычислительными машинами вроде бы не существует. Нет, на первый взгляд, проблемы и с каналами связи. Действительно, современная телефонизация охватывает все уголки нашей планеты и худобедно связывает их. Во всяком случае, каналы связи

имеются. Их-то и нужно попытаться приспособить для вычислительной сети.

Наличие вычислительных машин и каналов связи, безусловно, ободряет. Остается вроде бы немного — соединить машины каналами. Но именно на это тратятся все усилия проектировщиков и создателей сетей: простота объединения имеющейся сети связи (телефонной, радио и др.) с вычислительными машинами, эксплуатирующимися «на месте», только кажущаяся простота. Именно на это американцам понадобилось 12 лет и несколько миллиардов долларов при создании первой сети ARPA.

Какие же трудности приходится преодолевать при создании вычислительной сети? Прежде всего необходимо преодолеть «разноязычность» вычислительных машин и систем. Дело в том, что каждый тип ЭВМ имеет свой собственный внутренний язык (язык машинных кодов), разработанный специально для этого типа ЭВМ. Поэтому соединение двух разнотипных ЭВМ требует создания специальной аппаратуры и программы согласования их, которая играет роль переводчика с одного машинного языка на другой. Если пойти по этому пути, то таких «переводчиков» (в чем легко убедиться) нужно сделать $N(N-1)/2$ видов, где N — число типов ЭВМ. При этом разработка новой $(N+1)$-й машины сразу требует создания N таких «переводчиков». А если вспомнить, что сейчас $N > 1000$, то делать новую машину не захочется!

Заметим, что объединение однотипных ЭВМ в сеть уже не вызывает таких трудностей. Но это будет система, а не сеть! Сеть должна объединять любые ЭВМ.

Если бы создать один общий язык типа эсперанто, который могли бы понимать все машины, то станет значительно легче.

Итак, нужно было придумать и разработать такие способы общения разнотипных ЭВМ через систему связи, которые допускали бы быстрый и надежный обмен информацией (программами и данными) между самыми разнообразными вычислительными машинами и системами.

Теперь о каналах связи. Существующие телефонные и телеграфные каналы связи могут быть использованы лишь для передачи малых объемов информации и с небольшой скоростью. Дело в том, что свою функцию передачи информации они хорошо выполняют при

большой избыточности сообщения. Наша речь и телеграммы всегда избыточны. Так что при появлении помех, искажающих речь или телеграмму, обычно удается понять смысл сообщения (при разговоре, если не удалось догадаться, достаточно переспросить).

Можно ли пользоваться такими каналами для связи ЭВМ? Конечно, можно. Но...

Для защиты от помех приходится применять специальные методы кодирования, т. е. вводить все ту же избыточность, которая помогает бороться с помехами в канале связи. Введение избыточности делает связь надежной, но... очень медленной.

Таким образом, для передачи малых объемов неоперативной информации «старые» каналы связи еще годятся. Но для больших объемов и для быстрого обмена информацией надо создавать новые, более мощные, каналы связи (типа радиорелейных, которые передают телевизионные изображения, содержащие очень большую информацию).

Информация, которой обмениваются ЭВМ в вычислительной сети, имеет дискретную форму, т. е. передается в виде импульсов и пропусков (наличие импульса означает «1», а отсутствие «0», что вполне достаточно для передачи информации двоичным кодом). Для того, чтобы передать такую информацию по существующим телефонным каналам, ее надо преобразовать в сигнал, имеющий непрерывную или так называемую аналоговую форму, так как именно такой сигнал в состоянии передать телефонная сеть. Такой простейшей формой является синусоидальный сигнал. Наличие сигнала может означать «1», а отсутствие — «0» (в действительности «1» и «0» передаются сигналами различной частоты — это обеспечивает повышенную надежность передачи). Очевидно, что при этом скорость передачи информации будет невысокой. Такая скорость передачи, свойственная телефонным каналам и удовлетворяющая человека, никак не может удовлетворить потребностям интенсивного общения ЭВМ.

Именно это обстоятельство заставляет либо расширять возможности аналоговых каналов связи (как это делают радиорелейные линии при передаче телевизионных изображений и одновременно телефонных разговоров), либо создавать новые каналы связи, называемые дискретными, которые передают информацию в дискретной форме — в виде импульсов — и, следовательно,

не нуждаются в преобразовании двоичного сигнала в аналоговую форму (отрезки синусоиды разной частоты), которая снижает скорость передачи информации.

Преимущество дискретного канала по сравнению с аналоговым определяется следующими факторами. При прохождении сигнала по каналу связи он затухает. Нужно ставить усилители, которые вместе с сигналом усиливают и шумы, неизбежно накладывающиеся на этот сигнал. В аналоговом канале бороться с шумами, т. е. фильтровать их очень трудно. Для этого надо знать их свойства и, что самое неприятное, любой фильтр вносит задержку сигнала, что недопустимо для скоростных каналов передачи информации.

В дискретном канале шумы попросту «срезаются» формирователями, которые здесь играют роль усилителей. За счет этого такой дискретный канал передает дискретную информацию без помехи с огромной скоростью, в миллионы раз превышающую скорость передачи по аналоговому каналу.

Необходимость в создании таких каналов появилась именно с появлением вычислительных сетей. Так сети ЭВМ стали причиной появления нового типа связи — дискретной связи, возможности которой во много раз превышают возможности «старой» аналоговой связи. Отметим, что первая система дискретной передачи информации была создана в Японии в 1970 г.

Любопытно, что по дискретному каналу можно передавать и речь — существенно аналоговый сигнал. Для этого этот сигнал дискретизируется, т. е. кодируется двоичным кодом, после чего отправляется по дискретному каналу к приемнику собеседника. Здесь он декодируется, то есть снова трансформируется в аналоговую форму, которую и воспринимает собеседник.

Такие «дискретные» разговоры удобны тем, что могут вестись по каналу связи ЭВМ. Качество таких передач значительно лучше, чем «старым» аналоговым способом, так как они значительно меньше подвержены помехам.

Итак, задача создания вычислительных сетей свелась в основном к проблемам реализации дискретных каналов связи и способов взаимодействия ЭВМ с этими каналами (хотя возможность использования аналоговых каналов не только не исключается, но и предполагается, так как подавляющее число имеющихся сей-

час каналов связи имеют аналоговый характер, а дискретные только создаются).

Эти проблемы сейчас в значительной мере преодолены и создано много вычислительных сетей, которые интенсивно эксплуатируются.

А теперь давайте рассмотрим специфику образования и эксплуатации вычислительных сетей. Эта специфика весьма поучительна и опирается в основном на здравый смысл, который прежде всего обращается к различного рода прецедентам, аналогиям и даже аллегориям. Все эти приемы здравого смысла используются при создании вычислительных сетей.

Начнем с того, что обсудим трудности связи двух машин.

Как связываются две машины?

Начнем с простейшей схемы связи двух ЭВМ через обычный канал связи, например, телефонный. Представьте, что мощности вашей ЭВМ не хватает, а ваш приятель где-нибудь на Дальнем Востоке имеет возможность выходить на свою ЭВМ, которая пока не догружена. Вы, естественно, хотите использовать эти пропадающие мощности и договариваетесь с приятелем о ... Говорить о подключении дальневосточной ЭВМ к вашей пока рано. Давайте сначала просто пошлем туда программу, которую ваш приятель там «прокрутит» и вышлет результат — ответ обратно.

Итак, как быстро и надежно переслать программу (и так же быстро получить ответ)? Почта и телеграф отпадают, т. к. почта — слишком долго, а телеграфом — очень ненадежно (всем известны анекдоты, связанные с телеграфными ошибками), а в программе не должно быть ни одной — подчеркиваем: ни одной ошибки!

Остается телефон. Но как по телефону передать программу? Можно, разумеется передать ее, продиктовав по буквам. Но на это уйдет уйма времени (вашего и приятеля), да и обойдется в копеечку. Нельзя ли придумать что-нибудь подешевле? Да и не плохо исключить из процесса людей хотя бы для того, чтобы они занимались делом, достойным людей, а не автоматов. Надо придумать автомат, передающий по телефону программу. И такой автомат был придуман. Назвали его модемом, что есть сокращение двух слов МОдулятор и ДЕМодулятор.

Под модуляцией в данном случае подразумевается процесс представления единицы («1») в виде отрезка синусоидального сигнала одной частоты, а нуля («0») — другой частоты. Таким образом, двоичный сигнал преобразуется модемом в синусоиду переменной частоты.

Если такие модулированные сигналы попадают в телефонную трубку, то они воспринимаются характерным «попискиванием», его иногда можно слышать по телефону. Это самое «попискивание» и дает возможность передавать дискретную информацию по телефонным каналам связи. Естественно, что скорость передачи будет очень невелика — 30—40 знаков (букв и чисел) в секунду (это примерно 300—400 двоичных единиц информации в секунду).

На другом конце канала тоже ставят модем, который уже занимается демодуляцией, т. е. превращает модулированные сигналы в двоичный код, т. е. колебания напряжения с одной частотой воспринимает, как «1», а другой — как «0» (отсутствие электрических колебаний означает отсутствие связи). Легко представить, что один и тот же модем может работать попеременно в обоих режимах (поэтому он так и назван): то модулировать двоичный код, то демодулировать электрический синусоидальный сигнал.

Располагая таким модемом можно легко передать программу по телефону (см. рис. 11). Для этого достаточно отперфорировать свою программу на ленту и вставить ленту в считывающее устройство, которое будет выдавать запись программы в двоичном коде. Двоичный код программы вводится в модем. Полученные из модема электрические сигналы следует преобразовать в акустические (то самое «попискивание») с помощью динамика и послать их в микрофон телефонной трубки. Предварительно, разумеется нужно соединиться по телефонной сети с абонентом, который будет принимать программу.

Абонент может принять акустические сигналы из телефонной трубки с помощью микрофона и полученный электрических сигнал направить в свой модем, который, работая в режиме демодуляции, выдает двоичный код программы. Код уже нетрудно с помощью перфоратора преобразовать в ленту, на которой будет отперфорирована программа.

Если помех в канале связи не было и аппаратура работала нормально, то полученная абонентом перфолента должна быть копией (или, как принято говорить, идентична) вашей. (Пусть читателя не смущают «ручные» процедуры по заправке перфоленты в считывающее устройство при передаче и ее перенесение к ЭВМ при приеме. Этих операций в автоматической системе связи двух ЭВМ вовсе нет, так как нет перфоленты. ЭВМ в этом случае передает код программы прямо

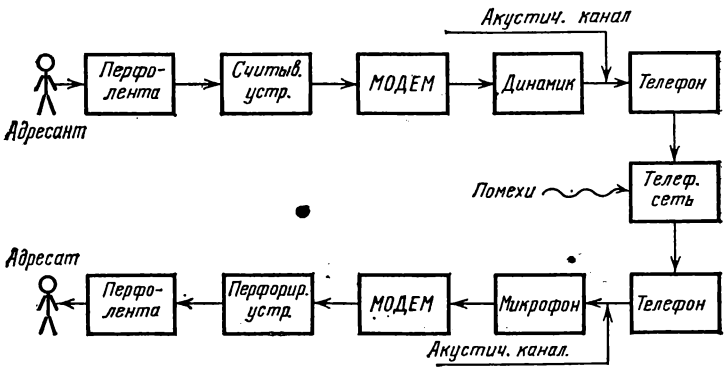

Рис. 11. Такова схема передачи программы (на перфоленте) по каналу телефонной связи. Автоматизировать его, т. е. исключить «человечков» на концах, уже не представляет трудности. Для этого достаточно автоматически ввести перфоленту в ЭВМ и вывести ее.

в модем, а на приемном пункте модем прямо соединяется с ЭВМ, которая запоминает вводимую в нее программу. Перфолента нам здесь понадобилась лишь для наглядности.)

Но вернемся к схеме с перфолентой. Такой способ передачи перфоленты может быть полностью автоматизирован и реализован в виде стандартного устройства, которое выполняет все указанные функции. Вы вызываете абонента по телефону, кладете трубку на это устройство и нажимаете кнопку «передача». Ваш абонент в это время должен сделать то же, но нажать на кнопку «прием». При этом ваша перфолента будет считываться и передаваться абоненту, а из его такого же устройства будет выползать перфолента принятой программы. Аналогично производится прием любой другой

информации, закодированной двоичным кодом, например, данных.

Чем удобен такой способ передачи? Да прежде всего тем, что позволяет пользоваться имеющимися каналами связи без какой-либо их переделки. Любая система связи — довольно консервативная организация (на то есть свои и очень убедительные основания) и очень не любит всякого рода вмешательств. Описанный способ не вмешивается в систему связи, а просто пользуется ею.

Теперь о негативных сторонах такой передачи. Их (увы!) значительно больше. Прежде всего помехи, которые неизбежно присутствуют в любом канале связи, а в телефонном — тем более. И здесь возникает извечная проблема: как надежно работать на ненадежной системе связи, т. е. как бороться со случайными помехами?

С этой проблемой вы столкнетесь при первой же передаче программы. Если вы будете излишне доверчивы и не позаботитесь о защите от возможных ошибок, то такая система связи двух ЭВМ сразу вас разочарует. Полученная абонентом программа почти наверняка окажется не той, которую вы посылали. В этом он немедленно убедится, когда захочет «прокрутить» ее на своей ЭВМ. И виной всему помехи в канале связи. Причем, чем длиннее будет сообщение (для канала связи ваша программа — лишь одно из передаваемых сообщений), тем более вероятно, что оно будет испорчено. Это — не закон подлости, а природа вещей, и наказывает она за неумение обращаться с ней.

Если бы такая ошибка произошла в телеграфном канале, то получатель ее мог и не заметить. Так, получив телеграмму «Встречай втокнир», вы не на минуту не усомнитесь, что надо встречать во вторник. Здесь избыточность слов языка и ожидание определенного вида сообщения позволяют однозначно восстановить его смысл даже при двух ошибках.

Можно ли этим воспользоваться при передаче программы? Кое в чем можно. Для этого следует просмотреть распечатку полученной программы, и если она была написана на алгоритмическом языке, то можно выявить некоторые искажения. Например, встретив слово CRINT и зная, что такого оператора в БЕЙСИКе нет (а на нем была написана программа), нетрудно

догадаться, что здесь был искажен оператор PRINT. И в данном случае мы воспользовались избыточностью слов бэйсика.

Но, если ошибка исказила цифру в программе, то обнаружить это почти невозможно. И этой, одной, ошибки достаточно, чтобы загубить всю программу. Поэтому рассмотрим сначала общие приемы борьбы с помехами.

Война с помехами

Эта война идет уже много столетий. Дело в том, что человечество давно и успешно использует разнообразные каналы связи — электрическими они стали лишь лет сто назад. Простейшим каналом связи является акустический. Именно его мы используем, когда разговариваем друг с другом. Если разговор идет в шумной обстановке, то мы невольно используем методы борьбы с помехами. Наиболее распространенными из них являются введение избыточности и переспроса.

Так, если, например, каждое слово в разговоре повторять несколько раз, т. е. таким не очень элегантным способом вводить избыточность, то собеседник наконец поймет его. Другое дело, что такая избыточность не очень располагает к длительной беседе; но пониманию она способствует. Другим способом повышения надежности канала связи является введение переспроса. Оба эти старые способа используются в современных системах связи.

Попробуем применить их для передачи программы от одной ЭВМ к другой по «шумящему» каналу («нешумящих» каналов не бывает; уровень шума может быть разный и именно это отличает хорошие каналы от плохих, но шум всегда есть!).

Можно попробовать повторять передачу программы до тех пор, пока ваш абонент среди множества полученных перфолент не обнаружит две одинаковых. Скорей всего это и есть неискаженная помехами программа, так как ошибки в канале связи имеют случайный характер и крайне маловероятно, чтобы искажение было два раза в одном и том же месте программы. (Хотя это возможно, например, за счет дефекта вашей перфоленты, но в этом вина не канала, а перфоратор-щицы.)

Но простой расчет показывает, что дожидаться двух идентичных программ вашему абоненту придется долго, а при длинной программе — всю свою жизнь.

Аналогичный результат постиг бы и другую тактику — посылать принятую перфоленту обратно с тем, чтобы вы сравнили бы их и убедились, что они ... разные. Ждать совпадения этих лент пришлось бы так же долго.

Неуспех таких приемов вовсе не означает, что передавать программу по телефонному каналу нельзя. Просто проблема безошибочной передачи по каналу с помехами достаточно сложна. А решается она следующим элегантным приемом.

Давайте разобьем нашу перфоленту на равные фрагменты (последний может быть и меньше) и будем добиваться правильной передачи каждого фрагмента, чем и достигнем правильной передачи всей программы. Итак, передаем фрагментами длиной l бит — двоичных единиц информации. (О выборе величины l поговорим особо позже в разделе, посвященном адаптации сетей.) Эта длина может быть выбрана в широком диапазоне от 10 до 10 000 бит в зависимости от степени «зашумленности» канала связи. Очевидно, чем выше уровень помех в канале, тем меньше должен быть фрагмент передаваемой информации и, наоборот, для хорошего канала фрагмент нужно брать большим, а для идеального (без помех) $l \to \infty$, т. е. программу дробить не надо, что в жизни не бывает в силу неидеальности реального канала связи.

Итак, надо надежно передавать фрагмент из l двоичных символов, т. е. микрососбщение, состоящее из l нулей и единиц. Давайте добавим к каждому фрагменту еще k контрольных двоичных символов, создающих необходимую избыточность. Эти добавочные биты нужно образовать так, чтобы можно было контролировать правильность передачи фрагмента.

Например, очень распространена проверка на четность. В этом случае $k = 1$ и контрольный бит определяет четность фрагмента. Это означает, что в случае, если число единиц в нем четно, то добавляется контрольный символ в виде нуля. А при нечетном числе единиц — добавляется единица. Например, фрагмент длиной $l = 7$

1001101

имеет четное число единиц и поэтому к нему добавляется нуль. Получаем блок

10011010;

в таком виде он отправляется абоненту. Фрагмент с нечетным числом единиц

0101100

отправляется в виде блока

01011001,

который образуется добавлением единицы.

На приемном конце блоки легко проверить на ошибку — надо сложить l первых символов и определить четность. Если она совпадает с $(l+1)$-м символом, то блок скорее всего принят правильно. Если — не совпадает, то где-то наверняка была ошибка — или в фрагменте, или в символе четности. В этом случае нужно повторно передать этот блок, т. е. переспросить передающее устройство.

Как видно, избыточность при этом небольшая (один бит на l битов сообщения), но и защита от помех при этом небольшая. Дело в том, что от одиночных редких ошибок этот способ защищает хорошо. Но, как показывает опыт, ошибки в канале связи «бродят» пачками (они хорошо прослушиваются в телефонной трубке в виде треска, вызванного атмосферными разрядами, искрением аппаратуры, электросваркой и т. д.).

Такая пачка помех искажает подряд много символов блока и поэтому может сохранить его четность (или нечетность). Такая грубая ошибка не будет распознана этим методом.

Именно поэтому добавляют не один контрольный бит ($k=1$), а больше (обычно $k=16$), причем связь между основным блоком и добавкой позволяет выявлять очень «хитрые» искажения блока фрагмента и контрольной добавки. Делается это следующим образом. Пусть A — наш передаваемый фрагмент — это l двоичных символов, а B — контрольная добавка, содержащая k двоичных символов. Связь между ними определяется некоторым правилом φ:

$$B=\varphi(A).$$

При $k=1$ правило φ сводится к вычислению четности (описано выше). При $k=2$ надежность передачи будет

выше, чем при $k = 1$. Так, увеличивая k, можно сколь угодно точно передать фрагмент сообщения при помехах.

На приемном конце, получив блок $A'B'$ (здесь штрихом отмечена принятая информация), эту операцию нужно повторить, т. е. снова определить контрольную добавку, но для принятого фрагмента A':

$$B'' = \varphi(A').$$

Если принятая (B') и вычисленная (B'') добавки не совпадают, $B' \neq B''$, то, очевидно, помеха при передаче исказила либо фрагмент, $A' \neq A$, либо контрольную добавку, т. е. $B' \neq B$. В обоих случаях следует запросить повторную передачу этого блока. Если же $B' = B''$, то считается, что

$$A = A' \quad \text{и} \quad B = B',$$

хотя ошибка в принципе и могла «проскочить». Но ее вероятность крайне не велика — порядка 10^{-16}, т. е. примерно одна ошибка в сто лет будет не замечена при таком контроле. Такое высокое качество передачи информации получено за счет хорошего введения избыточности, т. е. за счет правила φ. Синтезом такого рода правил, называемых алгоритмами кодирования, занимается теория помехоустойчивого кодирования.

Все сказанное имеет смысл при наличии обратного канала, по которому производится переспрос.

А нельзя ли обойтись без переспроса и при обнаружении ошибки, при $B' \neq B''$, попытаться восстановить текст, как это делаем мы, получив искаженную телеграмму? Можно, но при этом следует значительно увеличить избыточность, т. е. увеличить число контрольных битов k. Такие коды называют самокорректирующимися или восстанавливающимися. Они опираются на процедуру восстановления ψ:

$$A'' = \psi(A', B'),$$

которая дает возможность вычислить фрагмент A'' при наличии обнаруженной ошибки, т. е. при $B' \neq \varphi(A')$ по принятым A' и B'. Этот вычисленный фрагмент A'' и считают переданным фрагментом A, т. е. $A'' = A$. Таков алгоритм восстановления.

Однако может оказаться, что $A'' \neq A$, т. е. восстановление было ошибочным. Вероятность такого ошибочного восстановления невелика и может быть сделана сколь угодно малой при большом k и для одиночных ошибок в канале связи. Пачка же помех в канале

почти наверняка приведет к тому, что $A'' \neq A$, т. е. «изуродует» настолько фрагмент и добавку, что даже при очень большой избыточности восстановить блок не удастся. Так, легко себе представить, что помехи на линии так исказят текст телеграммы, что никакая избыточность не поможет понять ее смысл.

Именно поэтому системы связи с переспросом применяются чаще. Именно их используют и в системах связи ЭВМ.

Итак, добавление нескольких контрольных символов к фрагменту в сочетании с возможностью переспроса дает возможность вполне надежно и достаточно быстро передавать программы по телефонному каналу. Избыточность при этом небольшая, а повторные передачи блоков не отнимают много времени. В самом худшем случае, когда приходится дважды передавать каждый блок (что бывает при очень плохом состоянии канала связи), это приведет лишь к удвоению времени передачи программы.

Ну и что? Можно ли считать, что проблема создания вычислительной сети этим решена? Разумеется нет! Прежде всего потому, что скорость передачи информации таким образом слишком низка — примерно 1200 бит/с., т. е. 100—150 знаков в секунду. Небольшой массив, содержащий миллион знаков придется передавать несколько часов!

Нет, слишком медленные телефонные каналы, да и пользоваться ими не слишком удобно. Действительно, хозяином канала является телефонная сеть и она предоставляет их по заказам, когда она этого хочет. Если вам отказали в немедленном соединении, то это значит, что у сети оказались клиенты поважнее вас (если это не авария на линии).

Такая зависимость от «прихотей» сети недопустима для сети ЭВМ, для которой оперативность работы является одним из основных требований.

Вот и приходится с большим административным трудом добиваться не заказных, а выделенных каналов, т. е. таких каналов связи, которые принадлежат и обслуживаются сетью связи, а распоряжаются ими ЭВМ. В этом случае можно прямо включать модем в сеть, минуя телефонный аппарат. И эта возможность выходить в сеть связи в любой момент обеспечивает сети ЭВМ, построенной на ее базе, новые и невиданные доселе возможности.

А теперь представим, что телефонная сеть выделила нам каналы связи — столько, сколько нужно. Как ими распорядиться? Есть три способа использования сети связи в сетях ЭВМ и все три используются, но в разной мере.

Первым и очень естественным способом соединения двух ЭВМ является метод коммутации каналов. Его обозначают КК. Этот метод является прямым заимствованием метода автоматической телефонной связи.

Когда вы набираете по телефону код города (после «8», разумеется), то сеть связи автоматически прокладывает путь от вашего телефона через промежуточные станции к требуемому городу. На каждой станции нужные каналы замыкаются (коммутируются) так, что между вашим телефоном и телефоном вызываемого абонента образуется прямое электрическое соединение. Вы с абонентом как бы оказываетесь связанными одним проводом. (Говоря строго, это не так, но ведет себя эта связь именно таким образом.)

И это последовательное соединение многих каналов в один, обеспечивающий требуемую связь, будет существовать до тех пор, пока вы не положите трубку. Сигнал отбоя будет передан по всей цепи и разорвет установленные связи, т. е. раскоммутирует каналы, участвующие в соединении. Так работает автоматическая междугородняя телефонная связь.

И так же, по сути дела, работает вычислительная сеть, использующая коммутацию каналов. Желая войти в контакт с определенной ЭВМ, расположенной где-то далеко, вы сначала набираете ее код и ждете пока не будет установлено (закоммутировано) прямое соединение между вашим пультом и абонируемой ЭВМ. Если свободные каналы обнаружатся на всем пути от вас до этой ЭВМ, то ждать придется недолго. Если же где-то в промежуточном узле все линии заняты, то вам придется ждать соединения точно так, как приходится ждать соединения по междугородному автомату. Но уж соединившись с абонируемой ЭВМ, вы можете использовать проложенный канал связи столько, сколько нужно, и никто не сможет воспользоваться им кроме вас (разве что найдется в сети клиент «поважнее», чей приоритет будет выше, и автомат предоставит ему

связь, прервав вашу — такое случается не только в телефонной сети.)

Вроде бы всем хороша система коммутации каналов — и проста (это по сути дела автоматическая телефонная сеть) и дешева, так как не требует больших дополнительных затрат, и удобна (всегда хорошо иметь канал связи в своем распоряжении).

Но есть у этой системы один недостаток, который и решил ее дальнейшую судьбу. Этот недостаток является оборотной стороной ее последнего достоинства — удобства. Кому удобна КК? Тому, кто уже получил канал в свое распоряжение! И только!

Ситуация здесь очень похожа на ту, которая нередко складывается у телефонной будки — доволен только один, а остальные волнуются и нетерпеливо притоптывают. Если клиентов сети много, то предоставлять канал только одному — большое расточительство, особенно в случае, если канал составной и образован большим числом последовательно соединенных каналов. Все они будут заняты одним «разговором». Это очень дорогая плата за удобство одного пользователя сети.

Но ... удобства, которые имеет пользователь, получивший канал в свое распоряжение, неоспоримы и привлекательны. Поэтому идея коммутации каналов, когда одному пользователю, которому нужно быстро передать большой объем информации, предоставляется составной канал, выжила, хотя сетей с коммутацией каналов уже не строят.

Как ни странно, при большом числе ожидающих связи клиентов в сети коммутации каналов много каналов будут бездействовать. Это связано с тем, что они заблокированы другими загруженными каналами и для прокладки соединения нужно дождаться момента, когда все каналы требуемого соединения будут свободны. Легко представить, как один перегруженный канал будет препятствовать загрузке десятка других.

Вот и получается, что в сети коммутации каналов, несмотря на большие очереди, всегда много каналов недогружено. Это обстоятельство ограничивает возможности такой вычислительной сети.

Действительно, если не удается загрузить все каналы в сети, то ее пропускная способность не может быть высокой. А это сразу снижает оперативность обработки информации, что является одним из основных показателей вычислительной сети. Отсюда делается неумолимый

вывод: вычислительные сети с коммутацией каналов не имеют перспективы — они ограничены по своим возможностям.

Все это заставило искать новые пути реализации вычислительных сетей. И вскоре такой путь был найден. Его назвали метод коммутации сообщений.

Примите сообщение!

Для реализации этого метода в каждом узле сети связи понадобилось поставить ЭВМ с большой памятью. Именно в эту память передается сообщение из соседнего узла сети. Сообщение (программа, массив и т. д.) имеет заголовок в виде адреса: куда, т. е. в какой узел сети и на какую ЭВМ, оно направляется. Получив сообщение коммуникационная ЭВМ (так наывают машины, расположенные в узлах сети связи, еще их называют коммутирующими машинами или управляющими машинами узла связи) дожидается, когда освободится нужный канал в направлении, указанном в адресе и убедившись, что следующая коммуникационная машина готова принять сообщение, передает его. Вот и все!

Такой способ связи в вычислительной сети значительно лучше коммутации каналов, так как не требует ожидания, пока освободятся все каналы, образующие соединение — достаточно быть свободному одному каналу. Здесь каналы простаивают значительно меньше. И если какой-то канал перегружен, то сообщение может быть послано окольным путем, минуя перегруженный участок. Для этого, разумеется, сеть должна иметь избыточность, т. е. любые два ее узла должны иметь возможность соединяться по крайней мере двумя разными траекториями каналов сети.

Сеть коммутации сообщений передает каждое сообщение медленней, чем в сети коммутации каналов. Это происходит за счет промежуточных запоминаний всего сообщения в каждом узле сети. Однако, в целом сеть работает быстрее, так как ее каналы загружаются больше, чем при коммутации каналов, и поэтому время ожидания в очереди для входа в сеть значительно сокращается.

Правда в каждом узле надо иметь коммутационную машину с большой внешней памятью, но без такой ЭВМ сейчас не обходится ни одна вычислительная сеть. Трудности возникают при передаче больших и очень

больших сообщений. Такие сообщения относительно долго движутся по сети, что снижает оперативность ее работы. С другой стороны, длительное заполнение памяти коммуникационной машины не дает использовать ее же для приема сообщений с других каналов, т. е. блокирует эти каналы, что снижает эффективность сети.

И последнее. Как бы ни была велика память коммуникационной машины, всегда найдется сообщение (скорее всего это будет массив информации), которое не уместится в ней. Поэтому придется дробить сообщение, что вызывает дополнительные трудности.

Все эти обстоятельства привели к тому, что сети коммутации сообщений, хотя и нашли широкое применение, но перспективы, по-видимому, не имеют.

Основные претензии к сети коммутации сообщений следующие: а) надо иметь слишком большую память в узлах сети и б) относительно медленное движение сообщений по сети. Растущие потребности заставляли искать пути очень быстрой передачи информации по сети. И основным «поставщиком» таких потребностей стал диалог использователя с вычислительной сетью.

Дело в том, что процесс общения пользователя с вычислительной техникой вообще, и с вычислительной сетью в частности, имеет ярко выраженный двоякий характер. С одной стороны это задание на трудоемкую обработку информации (вычисления, поиск информации в банке и т. д.). А с другой — диалог, т. е. быстрый обмен короткими порциями информации, как это бывает при отладке программы или при заказе и бронировании билетов.

Если в первом случае излишняя оперативность ответа не обязательна, то во втором она просто необходима. Если ответ в диалоге задерживается, то пользователь начинает или нервничать или скучать. И то и другое снижает эффективность его взаимодействия с сетью.

Поэтому так важна скорость прохождения коротких сообщений по сети. Именно это обстоятельство и побудило искать новый способ коммутации в сети. Он был найден и назван методом коммутации пакетов.

Принцип горячей картошки

Все знают как чистят горячую картошку, сваренную или испеченную в «мундире». Именно так работает сеть коммутации пакетов. Только перебрасывает она

из одного своего узла в другой не горячую картошку, а пакеты.

Под пакетом подразумевается небольшой блок информации пользователя, обрамленный необходимыми символами служебной информации и двигающийся по сети от узла к узлу к своей цели — адресату. Пакеты «перебрасываются» из одного узла в другой так, чтобы они быстрее попали в узел назначения. Организация такой пакетной связи очень напоминает обычную почтовую связь, но в электронном исполнении.

Вы мне писали. Не отпирайтесь...

Давайте вспомним основные принципы почтовой корреспонденции. Именно они отражены в способах организации связи в сети коммутации пакетов.

Всякое письмо состоит из четырех непременных компонентов: текста, адреса и имени получателя, даты написания и подписи автора. Если хотя бы один из этих компонентов будет утрачен, то письмо потеряет всякий смысл.

Действительно, без текста писем не бывает (даже если «вместо строчки только точки — догадайся мол сама», то и эти точки являются весьма информативным текстом письма). Адрес и имя адресата являются необходимым условием получения письма адресатом (адрес «на деревню дедушке» — по-прежнему и в век кибернетики не поможет письму найти адресата даже, если приписать: «Константину Макаровичу», что правда несколько сузит круг «подозреваемых» адресатов). Необходимость даты в письме еще полностью не осознана некоторыми пользователями почтовой связи, однако хорошо понимается учреждениями, бланки которых предусматривают заполнение даты написания письма. Ее необходимость при, мягко говоря, прогрессирующем снижении эффективности почтовой связи (речь идет об ее оперативности), нетрудно усмотреть.

Действительно, ценность пересылаемой информации довольно быстро падает и часто становится равной нулю при большой задержке (так письмо с просьбой о встрече, полученное после прихода поезда, не содержит информации, разве, что для «Крокодила»).

И последнее — подпись. Вряд ли кто сомневается в важности подписи. Иногда текст письма подсказывает

имя адресанта. (А что делать с поздравительными письмами с неразборчивой подписью?) Еще хуже с деловыми письмами, где имя адресанта установить почти невозможно. Нет, без подписи письмо полностью теряет свой смысл.

Итак: текст, адрес, дата и подпись. И все? Нет, еще есть одна деталь, которая ускользает из личных писем и которую строго соблюдают в деловых. Речь идет о поводе — по поводу чего написано письмо. Дело в том, что два лица (адресант и адресат) могут общаться письменно по разным поводам и указание этого повода не просто помогает ответить на это письмо, но и определяет его смысл. Так один и тот же текст письма может быть воспринят по разному, если указать различные поводы. (Например, текст «Выделить тов. Иванову И. Н. 300 руб.» воспринимается одним образом, если повод «о награждении» и совсем по другому при «о материальной помощи».) В деловых письмах, которые пишутся разными лицами, а подписываются одним — директором учреждения — указание повода просто необходимо. Без него директор зачастую просто не будет знать, что нужно делать с этим письмом. А указание повода сразу определяет отдел, начальник которого должен подготовить ответ на письмо (с обязательным указанием повода, чтобы не ставить другого директора в тяжелое положение, когда ему приходится решать, что делать с этим письмом). Канцелярские уложения позаботились об этих заботах руководителей учреждений и требуют введения на учрежденческом бланке специальной рамки (в левом верхнем углу), где необходимо указать повод письма.

Таким образом, всякое письмо должно содержать: текст, повод, дату, адрес и подпись. Но на этом перипетии письма не кончаются. Неоходимо указать приоритет письма (высший — авиа и низший — простое). И надо определить его маршрут — этим занимается почтовое ведомство. (Так дальнее письмо можно направить по разным маршрутам и время его доставки будет разным. Поэтому выбор правильного маршрута часто является решающим фактором при минимизации времени доставки письма.)

Как видно, самое простое письмо (точнее, его текст) для того, чтобы быть отправленным по почте, должно быть обрамлено достаточно обильной информацией вспомогательного характера, без которой письмо теряет

свой смысл, так как либо не будет доставлено адресату, либо будет доставлено слишком поздно, либо адресат его не поймет.

Именно такого же рода процессы происходят при пакетной передаче информации по сети связи. Блок информации пользователя (необходимость дробления сообщений на блоки мы обсуждали выше) многократно обрамляется различной вспомогательной информацией, которая позволяет доставить его в виде пакета адресату, который «распаковав» пакет, находит в нем передаваемый ему блок. Давайте рассмотрим этапы «упаковки» информации пользователя при ее передаче в сети пакетной коммутации. Этапы и выполняемые при этом процедуры определяются так называемыми протоколами.

Протоколов много. Их удобно подразделить на две группы — протоколы высшего и низшего уровней.

Дела протокольные...

Слово «протокол» в русском языке имеет два довольно отличных смысла. С одной стороны, это документ, фиксирующий описание какого-то важного явления, которое произошло, кем-то наблюдалось и кого-то не оставило равнодушным, что и привело к появлению протокола (например, милицейский протокол, протокол допроса, протокол наблюдений в каком-либо естественнонаучном эксперименте — физическом, техническом, социальном и т. д.). Именно с таким использованием этого слова мы чаще всего сталкиваемся. С другой стороны, словом «протокол» определяют строгие правила поведения и общения кого-либо с кем-либо в тех случаях, когда это очень важно и несоблюдение этих правил приводит к серьезным нежелательным последствиям (например, дипломатический протокол предписывает правила поведения дипломатов в ситуациях, когда они представляют свою страну и может быть нанесен ущерб ее престижу). Именно в этом смысле использует протокол вычислительная сеть.

При общении различных ЭВМ через сеть связи также надо соблюдать определенные правила. Их-то и назвали протоколами.

Отличие протоколов верхнего уровня от протоколов нижнего заключается в том, что верхний уровень ведает процедурами взаимодействия решаемых прикладных

задач (вычислительных, информационных и т. д.) друг с другом через сеть связи, а низший — процессами связи в сети. Для того чтобы двум вычислительным процессам, происходящим в различных ЭВМ, находящихся на разных концах сети, обмениваться какой-то нужной информацией, необходима одновременная работа всех протоколов. Но лишь протоколы верхнего уровня способны интерпретировать переданную информацию и тем самым обеспечить взаимодействие обоих вычислительных процессов. Нижний уровень протоколов лишь обеспечивает передачу информации через сеть связи, «не задумываясь» о ее смысле.

Следующий пример иллюстрирует такое разделение процедур на высокий и низкий уровень. Когда вы пользуетесь телефоном для заказа в магазине, то нижний уровень процедур реализуется телефонной сетью, а верхний — вашим разговором с девушкой из стола заказов При этом происходит связь двух процессов — вашего заказа с процессом распределения товаров в магазине. Оба этих процесса понимают друг друга по «протоколам» естественного языка и норм общения. Это естественно протоколы верхнего уровня. Для них смысл передаваемого сообщения является основной заботой (так нарушение протокола норм поведения при грубом общении лишает вас возможности оформить заказ. Девушка попросту положит трубку и прервет связь). Нижний уровень протоколов лишь обеспечивает связь процессов и для него не нужно знать смысла передаваемых сообщений. Он ему не требуется.

«Добру и злу внимая равнодушно», нижний уровень лишь обеспечивает коммутацию и не заботится ни о чем другом.

А теперь рассмотрим эти протоколы. Начнем сверху.

Протоколы высокого уровня

Эти протоколы определяют вид и характер взаимодействия пользователей и вычислительных процессов, происходящих в различных ЭВМ, связываемых сетью связи. Взаимодействие двух абонентов сети (под абонентами здесь понимается и пользователь, решающий свою задачу, и программа, с которой работает пользователь) может быть различным. Наиболее часты следующие виды взаимодействия абонентов:

1. Передача файлов, т. е. пересылка по сети информации, необходимой для осуществления того или иного вычислительного процесса.

2. Удаленный ввод заданий для расчетов, т. е. передача по сети исходных данных для реализации вычислений на какой-то ЭВМ сети, где имеется необходимая программа (вместо того, чтобы эту программу передавать пользователю, что было бы реализацией первого вида взаимодействия — передачей файла, содержащего интересующую пользователя программу).

3. Диалоговое (его часто называют еще интерактивным) взаимодействие пользователя с банками данных. Специфика этого взаимодействия определяется прежде всего многократной реализацией коротких вопросов пользователя и быстрых и коротких ответов банка данных — иначе диалога не получится.

4. Электронная почта является так сказать «бесплатным приложением» вычислительной сети. Она осуществляет передачу текстов писем другим пользователям сети по их имени или паролю, писем заказных или до востребования, с уведомлением о вручении и без. Такая почта объединяет пользователей сети в необычайный коллектив, где отдельные его представители могут лично не знать друг друга, будучи расположенными, например, на разных континентах, и тем не менее успешно сотрудничать при решении на сети своих сложных задач.

Каждый из видов взаимодействий подразумевает определенного рода строгие процедуры, которые регламентируются протоколами верхнего уровня. Это значит, что, желая, например, связаться со своим коллегой на другом конце вычислительной сети, нужно обязательно выполнить определенные действия, определяемые протоколом. При передаче файлов по сети нужно следовать другому протоколу, также как и при диалоге с банком или при удаленном вводе заданий. Для каждого вида взаимодействия имеется свой протокол, т. е. строго определенная форма взаимодействия, одинаковая для всех пользователей сети.

Такого рода протоколы, расположенные на высшем уровне протокольной иерархии, играют фактически роль языков высокого уровня, к которым относятся алгоритмические языки, о которых мы уже говорили в первой главе. Всякий алгоритмический язык, кроме всего прочего, определяет правила общения пользова-

теля с ЭВМ, имеющей транслятор с этого языка. Язык в данном случае выступает как средство правильного общения, обеспечивающего понимание пользователя вычислительной сетью. Именно эту функцию выполняют протоколы высшего уровня.

Так как пользователь обычно общается с ЭВМ на алгоритмических языках (таких как, фортран, GPSS и др.), то при работе с сетью эти языки должны быть расширены средствами, позволяющими пользователю связываться (точнее: отдавать распоряжения на связь) с определенными абонентами (процессами или пользователями), расположенными в других узлах сети. Такое расширение алгоритмических языков должно быть подчинено протоколу высшего уровня — иначе сеть просто не поймет пользователя.

Все остальные протоколы уже прямо не контактируют с пользователем и поэтому в книге для пользователя о них можно было бы не говорить. Однако, для понимания того, что происходит после того или иного действия пользователя, следует хотя бы бегло рассмотреть как и, главное, для чего происходит довольно сложное преобразование информации пользователя, чтобы сеть выполнила заданную ей функцию.

Так, указанное «протокольное» расширение алгоритмических языков должно обеспечить выполнение сетью, например, такого задания: «Из банка данных *A*, расположенного в узле *B* сети, взять программу *C* и переслать ее в узел *D*. Исходные данные *E* взять у абонента *G*, расположенного в узле *K*, и направить в узел *D*. Результат расчета по программе *C* с данными *E* в узле *D* направить в узел *I* и вывести на графопостроитель абонента *L*». Легко себе представить, что для описания этого задания нужно иметь весьма гибкий язык, подчиняющийся определенному сетевому протоколу.

Таким образом, на высшем уровне протокольной иерархии расположен протокол общения абонента с сетью, который должен соблюдаться этим абонентом, если он хочет воспользоваться всеми услугами, предоставляемыми вычислительной сетью.

Дальнейшее преобразование информации, введенной пользователем, осуществляется другими протоколами, расположенными на разных уровнях протокольной иерархии. Всего таких уровней может быть от трех до семи. Рассмотрим семиуровневую иерархию, как наиболеее

детализированную. (Другие варианты иерархии образуются простым объединением этих уровней.)

Протоколы высокого уровня занимают три верхних этажа: седьмой, шестой и пятый. Седьмой уровень, называемый *прикладным,* или *пользовательским* уровнем, рассмотрен выше с позиций пользователя. Однако, этот уровень часто называют также уровнем прикладных процессов, подразумевая под этим то, что именно на этом уровне происходит выход из сети на ЭВМ, с которой работает пользователь (см. рис. 12). Вычислительный процесс происходит в ЭВМ, а седьмой уровень

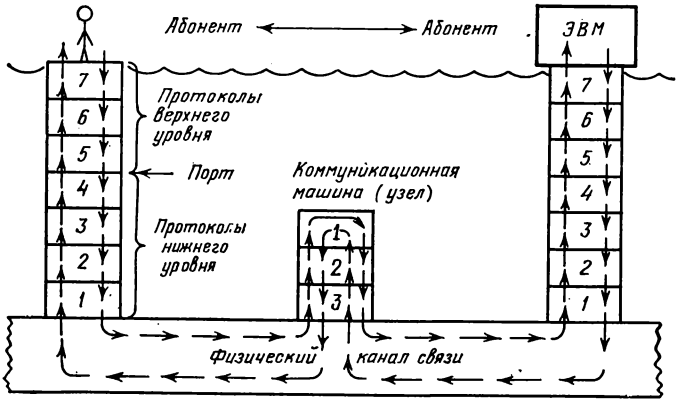

Рис. 12. Такова схема прохождения информации по сети между двумя абонентами (пользователем и ЭВМ). По дороге этой информации приходится проходить узлы, реализованные с помощью коммутационных машин. Здесь кадру «приходится» подниматься до третьего уровня, на котором осуществляется выбор его дальнейшего маршрута.

протоколов связи с этой ЭВМ обеспечивает ее стыковку с вычислительной сетью.

Сигнал от пользователя к ЭВМ и обратно обязательно проходит через все уровни протоколов так, как это показано на рис. 12. Пользователь же никак не ощущает этого. Для него связь как бы осуществляется по прямой стрелке. Он видит лишь «надводную» часть айсберга вычислительной сети. А под «водой» остались все остальные уровни протоколов вместе с каналами связи.

Шестой уровень протоколов называют обычно *представительным* уровнем. Он определяет в основном процедуру представления передаваемой информации в нужную сетевую форму. Связано это с тем, что сеть объединяет разные машины. Если бы ЭВМ в сети были

бы одного типа, то не понадобилось бы вводить процедуру представления к единой форме. Но в сети, объединяющей разнотипные ЭВМ, информация, передаваемая по сети, должна иметь определенную единую форму представления. Именно эту форму определяет протокол шестого уровня. Информация, пришедшая с

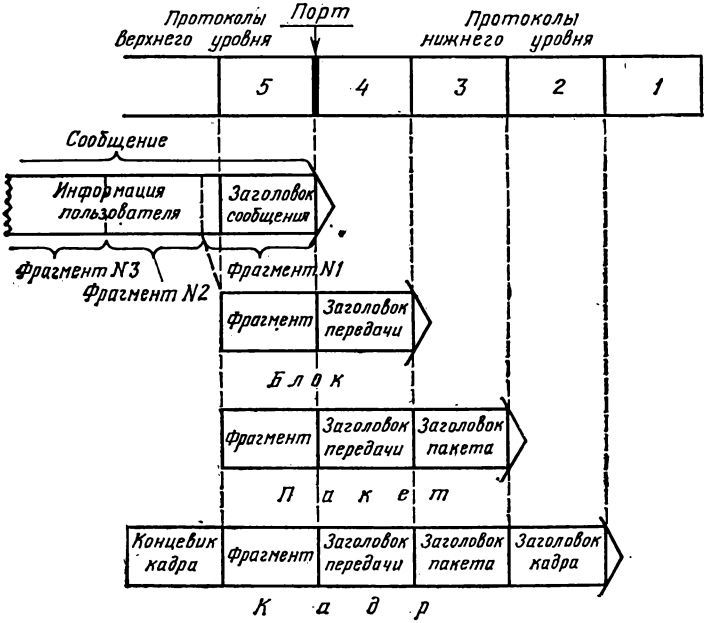

Рис. 13. Здесь представлена схема наращивания блока передаваемой по сети информации при прохождении по уровням транспортного канала. На приемном конце и в узле все происходит в обратном порядке; на втором уровне снимаются заголовки канала, на третьем — заголовок пакета, а на четвертом — заголовок передачи. Полученная при этом информация используется для управления транспортным каналом.

верхнего, седьмого, уровня, имела иное представление. Конечно, можно было бы пользователя заставить протоколом седьмого уровня представлять информацию сразу в требуемой «сетевой» форме и представительного уровня не понадобилось бы. Но это очень усложнило бы его жизнь. Ко всему прочему протоколам седьмого уровня пришлось бы выполнять дополнительные требования протокола представления информации. Но преобразование представления информации в требуемую форму можно осуществлять с помощью специальных

программ. Именно они и реализуют требование протоколов шестого уровня.

Следующий пятый уровень протоколов называют обычно *сеансовым*. Его назначение состоит в организации сеансов связи между прикладными процессами. В соответствии с этим протоколом сеансовый уровень образует (по команде сверху, разумеется) соединение прикладных процессов для их взаимодействия по принципу коммутации каналов, организацию передачи информации между процессами во время взаимодействия и «рассоединение» процессов.

На пятом уровне кончаются протоколы высокого уровня и начинаются протоколы низкого уровня, обслуживающие транспортную сеть — их четыре. Выход информации в транспортную сеть осуществляется через так называемый *порт*. Каждый процесс (пользователь) выходит в сеть через свои порты, а число портов у каждого пользователя равно числу его адресатов, с которыми он взаимодействует в данный момент.

Перед выходом через порт в транспортную сеть информации пользователя приписывается заголовок того процесса, который породил эту информацию. Этот заголовок ничто иное как подпись автора этой информации. В таком виде она называется сообщением (см. рис. 13).

Это сообщение через порт процесса, его породившего, поступает в транспортную сеть для передачи адресату по протоколам низкого уровня.

Протоколы низкого уровня (транспортная сеть)

Эти протоколы обслуживают так называемую *транспортную* сеть. Как любая транспортная система эта сеть транспортирует вверенные ей «грузы» — информацию пользователя, не интересуясь ее содержанием. Ей важно быстро и надежно передать информацию от одного абонента сети к другому. Быстроту обеспечивает электроника, а надежность — способы защиты и упаковки информации (они выполняют роль мягких прокладок и контейнеров, используемых при дорожных перевозках грузов). Аналогия здесь очень глубокая, так как в обоих случаях речь идет о реализации процессов эффективной транспортировки.

А теперь продолжим рассмотрение семиуровневой системы протоколов вычислительной сети. Как сказано,

на транспортную сеть информация поступает в виде сообщения пользователя. На четвертом уровне управления сетью, называемом *транспортным* или уровнем *управления передачей*, в соответствии с протоколом этого уровня это сообщение разбивается на одинаковые фрагменты (№ 1, 2, ...) и каждому фрагменту приписывается заголовок передачи (см. рис. 13), где указывается номер фрагмента и имя порта назначения. В таком виде фрагмент с заголовком передачи называют блоком. На приемном конце сети этот четвертый уровень управления снимет заголовок передачи, прочтет в нем имя нужного порта и номер фрагмента, соберет из фрагментов сообщение и отправит его через указанный порт адресату.

Третий уровень протоколов называют *сетевым*. Он обслуживает процессы управления маршрутизацией пакетов в сети, т. е. определяет траекторию движения блока по узлам сети. Для этого на основе имеющейся информации о состоянии транспортной сети связи, т. е. о загрузке ее каналов, состоянии узлов и т. д. вырабатывается маршрут прохождения пакета. Сам пакет образуется добавлением к блоку заголовка пакета (см. рис. 13). В этом заголовке содержится вся информация, необходимая для определения маршрута образованного таким образом пакета.

Именно до этого третьего уровня «раздевают» пакет при его прохождении узла сети, то есть в коммутационной машине. Этот случай показан на рис. 12. Необходимость указанного очевидна. Действительно, для того, чтобы пакет прошел транзитом через узел, этому узлу нужно знать маршрут пакета. А информация о маршруте содержится именно в заголовке пакета.

На приемном конце сетевой уровень снимет заголовок пришедшего пакета (ему он больше не нужен) и передаст полученный блок наверх — на четвертый уровень.

Далее пакет направляют на второй уровень управления транспортной сетью, обычно называемый *канальным,* работа которого регламентируется протоколом этого второго уровня, или *канальным* протоколом. В соответствии с этим протоколом из пакета образуется кадр, который и начнет двигаться по каналу связи. Кадр образуется обрамлением пакета заголовком и концевиком кадра (см. рис. 13). В заголовок вводится вся информация, необходимая для управления тем

каналом связи, по которому предстоит идти кадру. Такой информацией может быть, например, сообщение о том, что происходит в том узле, откуда только что вышел кадр, и т. д. Концевик кадра образуется теми самыми дополнительными контрольными битами информации (их 16), которые необходимы для надежной передачи всего кадра (о них говорилось выше), а точнее — для проверки правильности передачи на приемном конце канала.

Для обозначения кадра в его начале и конце ставятся так называемые флаги, например, восьмисимвольные последовательности вида 01111110. С помощью флагов и удается выделить кадр: все что между двумя флагами и есть кадр. А для того, чтобы флаговое сочетание нулей и единиц не встретилось в кадре, производится его остроумное преобразование: после каждой пятерки единиц, т. е. после 11111, вставляется нуль, который автоматически снимается на приемном конце канала вместе с флагами. (Эту процедуру называют битстаффингом — дословно: вставка бит.) После этого, кроме флагов, в кадре не будет ни одного сочетания вида 01111110, что однозначно и определяет кадр.

В функцию канального уровня управления входит и проверка правильности полученных кадров, для чего используется концевик кадра, с помощью которого описанным способом можно узнать правильно ли принят кадр. При правильном приеме в соседний узел, откуда прибыл кадр, направляется подтверждение о приеме. В противном случае — требование на повторную передачу кадра. Эта информация (подтверждение или запрос на повтор) передается в заголовке очередного кадра, отправляемого в передающий узел. Если же пакета нет, т. е. передавать нечего, то специально для этого образуется служебный кадр, состоящий лишь из заголовка и концевика (и флагов, разумеется), который отправляется в нужный узел.

Последним (первым) уровнем управления транспортной сетью является так называемый *физический* уровень. Его работа регламентируется протоколом физического уровня, который предписывает правила установления соединения с физическим каналом, поддержания этого соединения и расторжения его. Здесь определяются правила передачи каждого бита через физический канал. Этот канал может быть па-

раллельным и передавать несколько бит сразу, что увеличивает скорость передачи информации. Именно на этом уровне работает модем, рассмотренный выше.

Описанные уровни управления, как и протоколы, регламентирующие управление, являются условными. В существующих вычислительных сетях, некоторые из описанных уровней объединяются. Сами протоколы обязательны для каждой сети в отдельности, но могут значительно отличаться от протоколов других сетей. Однако, международными организациями разработаны рекомендации по некоторым протоколам низкого уровня: транспортным протоколам третьего и второго уровней. Есть рекомендации по протоколам межсетевой связи двух вычислительных сетей.

Следовать этим рекомендациям, естественно, не обязательно, но целесообразно по двум соображениям. Во-первых, в этих рекомендациях отражен большой опыт эксплуатации нескольких вычислительных сетей, действующих в Европе и Америке. Опыт очень ценен, т. к. теории протоколов пока не существует и эффективность работы сети с тем или иным протоколом лучше всего определить, поработав с ним на реальной сети (таким экспериментальным полигоном протоколов является, например, первая сеть ARPA, которая, впрочем, выполняет огромную вычислительную работу — в основном для Пентагона — своего бывшего хозяина).

Во-вторых, протоколы являются чрезвычайно сложными конструкциями, в которых должно быть предусмотрено заранее огромное число различных факторов и ситуаций, складывающихся в процессе функционирования сети (выше указаны лишь основные функции протоколов). Разработка каждого нового протокола связана с большой затратой интеллектуальных, временны́х и материальных ресурсов. Именно поэтому так прислушиваются разработчики к международным рекомендациям по протоколам.

Описанная выше семиуровневая система управления процессом связи в вычислительной сети обладает широкими возможностями. Она может функционировать в различных режимах. Наиболее распространенными из них являются датаграммный режим и режим виртуальных соединений. Рассмотрим их.

Датаграммы

Это странное название (datagramm — в дословном переводе: «записанные данные», читается как «дейтаграм»; однако в русском языке укоренилась «датаграмма», которая здесь и используется) получил самостоятельный пакет, который движется по сети независимо от других пакетов, даже порожденных одним и тем же процессом. Вспомним, что пакет в своем заголовке имеет сведения о своем маршруте, что более чем достаточно для его доставки адресату (так, например, работает сеть ARPA, эта датаграммная вычислительная сеть). Если же этих сведений нет, то вполне достаточно иметь адрес, по которому послан пакет. Делается это, например, так.

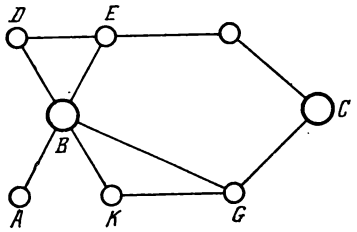

Рис. 14. Пример вычислительной сети, где распределение датаграмм в узле *B* определяется сложившейся ситуацией. Пакет с бóльшим преимуществом направляется в узел, ближайший к абоненту, с учетом того, как он (узел) принимал пакеты в ближайшем прошлом.

Получив датаграмму узел коммутации направляет его по правилу горячей картошки в сторону того узла, куда она направлена. Например (см. рис. 14), получив датаграмму из *A* узел *B*, выяснив из заголовка пакета, что его адресат находится в узле *C*, решает, куда ее следует направить. Сначала он его направляет в узел *G* ближайший к *C*. Если после такой отправки из *G* получена положительная квитанция о получении пакета, то узел *B* снимает с себя заботу об этом пакете (точнее: датаграмме), т. е. стирает его в своей памяти. Теперь его опеку берет на себя узел *G*. Если же положительная квитанция от *G* не поступила (а это означает, что либо память *G* переполнена, либо он неисправен, либо...), то *B* направляет злополучную датаграмму в узел *K* или в узел *E* и уж при их отказе в узел *D*.

Таким образом, все узлы, окружающие данный, ранжируются по близости к адресному. Первый ранг получает ближайший к адресату узел (в данном случае узел *G*), второй ранг — ближайший из остальных (*E* или *K*) и т. д. Пакет посылается сначала в узел первого ранга, при неудаче — в узел второго ранга и т. д.

Правила отправления пакетов образуют алгоритм маршрутизации. Таких алгоритмов существует много. Простейший из них описан выше (отправлять сначала в узел ближайший к адресуемому, при неудаче — в ближайший из остальных и т. д.).

Легко заметить, что этот алгоритм никак нельзя назвать наилучшим, хотя бы потому, что при неисправности одного из узлов, туда все равно будут направляться пакеты. Поэтому адаптивный алгоритм маршрутизации, учитывающий опыт функционирования узла, т. е. опыт посылок пакетов в разные узлы, следует считать более оперативным и эффективным. Состоит он в следующем. Каждый узел сети, смежный данному, имеет в каждый момент определенный вес, характеризующий его перспективность для передачи ему пакета. Этот вес можно определить, например, как относительное число удачных посылок за последние T секунд:

$$a_i = \frac{N_i^*}{N_i},$$

где N_i — общее число пакетов, посланных в i-й смежный узел за последние T секунд, а N_i^* — число принятых им пакетов, т. е. число полученных подтверждений из i-го узла.

Пусть q_i — ранг i-го узла ($i=1, \ldots, k$), а k — число узлов, смежных данному. Очевидно, что отношение a_i/q_i характеризует перспективность i-го узла для передачи датаграммы. Теперь можно поступить двояко.

При детерминированном подходе посылку датаграммы следует делать в направлении узла с наибольшим значением отношения a_i/q_i. Далее, при неудаче — с наибольшим из оставшихся и т. д.

При вероятностном подходе направление отсылки датаграммы случайно. Это означает, что i-й узел выбирается случайно, но с вероятностью пропорциональной величине a_i/q_i, т. е. с вероятностью

$$p_i = \frac{a_i}{q_i \sum_{i=1}^{k} \frac{a_i}{q_i}} \qquad (i = 1, \ldots, k).$$

Легко видеть, что неработающие или плохо работающие узлы будут иметь $p_i = 0$. Таким образом, они будут автоматически исключены из списка узлов, куда

может быть направлена датаграмма, что, естественно, ускорит ее доставку адресату.

Очевидно, что при такой адаптивной маршрутизации каждая датаграмма будет идти по своей случайной траектории, а, следовательно, и момент поступления ее к адресату будет случайным. Это обстоятельство не должно смущать, так как свойствами случайности можно управлять, т. е. добиваться, чтобы среднее время доставки датаграммы не превышало заданное, а вероятность того, что какая-то датаграмма задержится более чем t секунд была бы достаточно малой.

При этом поступать адресату датаграммы будут очевидно не в порядке их посылки. Поэтому собрать передаваемое сообщение можно не раньше, чем придет последняя датаграмма. Это несколько задерживает процесс пересылки сообщения по сети. Для преодоления этого недостатка был предложен режим виртуальных соединений каналов в сети. Рассмотрим его.

Виртуальные соединения

По сути дела это коммутация каналов, но не напрямую, а через память коммутационных машин и с использованием пакетов при передаче сообщения. В виртуальной сети прежде чем начать передачу пакетов адресату направляется служебный пакет, который прокладывает виртуальное соединение. Это означает, что в каждом узле он оставляет распоряжение вида: пакеты этого (k-го) виртуального соединения, пришедшие из i-го канала, следует направлять в j-й канал. В этом и состоит виртуальность (условность) соединения — оно осуществляется программно. Дойдя до адресата, управляющий пакет запрашивает у него разрешение на передачу, сообщив, какой объем памяти понадобится для приема сообщения. Если машина — адресат располагает такой памятью и свободна, то она посылает адресанту (откуда получен управляющий пакет) свое согласие на прием сообщения. Это оформляется в виде служебного пакета. Получив подтверждение, адресант начинает передачу своего сообщения обычными информационными пакетами. Эти пакеты беспрепятственно проходят друг за другом по виртуальному соединению (в каждом узле их ждет соответствующая инструкция, оставленная управляющим пакетом, которая обрабатывается коммуникационной машиной) и в

том же порядке попадают адресату, где, освободившись от заголовков и концевиков, образуют передаваемое сообщение, которое и направляется адресату на седьмой уровень. Виртуальное соединение между двумя абонентами (пользователями или процессами) может существовать сколько угодно, пока в нем не отпадет необходимость. Сеанс связи прекращается путем передачи одним из абонентов этой связи по виртуальному соединению служебного кадра на прекращение сеанса. Этот кадр стирает инструкции в узлах, в результате чего виртуальное соединение распадается.

Легко видеть, что режим виртуальных соединений более удобен и эффективен при передаче больших массивов информации по сети, он обладает всеми преимуществами методов коммутации каналов и пакетов, описанными выше. Для коротких сообщений более эффективен датаграммный режим, не требующий довольно громоздкой процедуры установления виртуального соединения между адресантом и адресатом сообщения.

Поэтому очень естественно объединить оба режима работы — датаграммный и виртуальный — в одной вычислительной сети (такую сеть называют виртуально-датаграммной), которая бы использовала преимущества обоих режимов и применяла бы тот, который в сложившейся ситуации выгоднее. Преимущества этого неоспоримы.

А нельзя ли проще? (связь через спутник)

Выше мы рассмотрели весьма сложную семиуровневую организацию процессов передачи информации по транспортной сети в виде пакетов. Вся эта сложность была порождена тем, что одной и той же сетью связи пользуются сразу несколько абонентов (пользователей и процессов).

Нельзя ли упростить систему связи за счет того, что в каждый момент связь осуществляется между двумя абонентами сети? Можно. И здесь очень пригодится спутник, находящийся на стационарной орбите, высотой примерно 35800 км от Земли. Такой спутник «висит» над одной точкой земли (над экватором) и может обслуживать связью всех, находящихся в радиусе 14 тыс. км.

Как же организуется связь ЭВМ через спутник? Прежде всего отметим, что спутник выступает здесь простым ретранслятором, т. е. принимает сигналы на одной волне, усиливает их и тут же передает обратно на Землю на другой волне. Его приемнопередаточная антенна направлена на всех, кто может оказаться в прямой видимости спутника.

Таким образом, передав сообщение через спутник, мы всегда можем проконтролировать, что принял адресат. Для этого достаточно иметь приемник, настроенный на волну передатчика спутника. Кстати, эта возможность позволяет не добавлять контрольные биты для проверки правильности передачи. Достаточно сравнить переданное сообщение (оно хранится в памяти) с принятым и при их несовпадении повторять передачу.

Рис. 15. А так осуществляется связь через спутник. Каждая станция посылает свой пакет и принимает результат ретрансляции пакета спутником. Самая главная проблема здесь заключается в том, чтобы не наложились пакеты, передаваемые различными станциями. Именно для этого необходима адаптация.

Однако источник помех может находиться рядом с адресатом, который поэтому не сможет узнать, правильно он принял кадр или нет. Поэтому контрольная добавка (16 бит) все же сопровождает передаваемый пакет и в случае спутниковой связи.

Итак, спутник принимает сообщение в виде пакетов от всех передатчиков и ретранслирует их всем приемникам (см. рис. 15) сразу. На первый взгляд такой способ лишен смысла, так как возможно наложение передач, что сразу и полностью обесценивает все одновременные передачи. Что же может передать такая система связи? Оказывается, очень много.

Для того, чтобы разобраться в этом, введем понятие конфликта в такой системе связи. Будем считать, что конфликт возникает тогда, когда два и более пакетов, переданных различными станциями, наложатся, что сразу фиксируют все станции, обнаружив ошибку в передаче с помощью контрольной добавки к принятому пакету. Пакет проходит бесконфликтно только не «сталкиваясь» ни с одним другим.

Бесконфликтный пакет принимают все станции в том числе и та (или те), которой он послан. В его заголовке указывается адресат, который и начинает дальнейшую обработку пакета. Следует отметить, что в такой системе секреты скрывать довольно трудно, разве что договориться с адресатом о специальном коде, засекречивающем передаваемую информацию. Но зато легко передавать информацию всем сразу. Для этого есть специальный адрес «ВСЕМ».

Приемные станции сделаны так, что получив пакет с чужим адресатом, они просто забывают его и реагируют лишь на свой адрес или на всеобщий — «ВСЕМ».

Итак проблема состоит в том, чтобы конфликтов было бы меньше и, в идеальном случае, чтобы в каждый момент времени передавала бы лишь одна станция — тогда конфликтов вообще не будет.

Этого можно добиться разными способами. Первое, что приходит в голову, это выделять каждой станции периодически какой-то квант времени так, чтобы станции поочередно выходили на связь со спутником. Это обеспечит бесконфликтность передач, но ... слишком дорогой ценой. Действительно, в этом случае неработающей станции тоже будет выделяться время, что приведет к потере «эфирного» времени системой связи, т. е. к потере ее пропускной способности. Можно было бы соединить все станции наземной системой синхронизации и регламентации ее работы. Но это уже сеть связи между станциями и не проще ли именно по ней передавать нужную информацию?

Остроумным выходом из такой ситуации было предложение отправлять пакеты через случайные промежутки времени. В этом случае (в силу случайности момента передачи пакета каждой станцией) всегда может оказаться, что при передаче пакета одной станции другие молчат и пакет пройдет канал спутниковой связи бесконфликтно, в чем эта станция сразу удостоверится. (Не совсем «сразу», т. к. на прохождение радиосигнала до спутника и обратно тратится вполне внушительное время — более четверти секунды.)

Алгоритм работы станции в этом случае таков: в случайные моменты времени передавать пакеты, а в случае конфликта — повторять передачу конфликтного пакета. Для этого однако следует хранить в памяти все

пакеты, переданные за последние четверть секунды. Вот и все!

Число конфликтов здесь зависит от среднего значения величины временного интервала между посылками пакетов. Чем больше эта величина, тем меньше конфликтов, но при этом увеличивается число «пустых окон», когда вовсе нет передачи ни одного пакета ни одной станцией. Уменьшение интервала между пакетами уменьшает число «пустых окон», но зато увеличивает число конфликтов, когда связь также отсутствует. Как быть?

Очевидно, что максимальной пропускная способность такого канала спутниковой связи будет при оптимальном выборе средней величины случайного интервала между посылками пакетов.

Расчеты показывают, что средняя скорость передачи такой системы связи равна 18,4% от максимальной, которая будет при одной передающей станции, т. е. когда конфликтов и пустых окон нет.

Можно ли увеличить эту пропускную способность? Можно. Но для этого надо синхронизировать работу станций, т. е. отправлять пакеты не в любой момент времени, а лишь в строго синхронизированные моменты. Это означает, что все время работы канала связи разбивается на окна, длина которых равна времени передачи одного пакета (все пакеты предполагаются одинаковыми).

Каждая станция может начинать передачу пакета только в начале окна. Очевидно, что посылать пакеты в каждое окно нет смысла, т. к. почти наверняка возникнут конфликты с пакетами других станций. Надо пропускать окна. Как это сделать? Проще всего — случайно.

Для этого перед каждым окном каждая станция решает: передавать пакет или не передавать. И решение это доверяется случаю, т. е. задается вероятность p передачи пакета. Соответственно с вероятностью $1-p$ пакет не передается. И так поступают все станции!

Легко заметить, что эффективность канала связи зависит от величины p и от числа работающих станций N. Если N велико, то возникнет много конфликтов, которые снизят пропускную способность такого канала связи. При очень малом N конфликтов мало, но много пустых окон, что также снижает эффективность связи.

Очевидно, что существует оптимальное значение вероятности $p = p^*$, при котором эффективность связи максимальна. Теория показывает, что эта оптимальная вероятность равна

$$p^* = \frac{1}{N}.$$

В этом случае скорость передачи вдвое больше, чем при отсутствии синхронизации, т. е. 36,8% от максимальной.

Для реализации такой скорости должно быть известно N — число работающих станций. Но это число все время меняется — ведь это зависит от того, есть запросы на передачу сообщений или нет на каждой станции. Именно поэтому определить N практически невозможно.

Что же делать? Значит ли это, что эта скорость недостижима? Нет, достижима.

Как всегда, когда решение затрудняется из-за отсутствия необходимой информации, на помощь приходит адаптация. Реализуется она следующим образом.

Пусть сначала вероятность p равна любому числу $p_0 (0 < p_0 < 1)$. Приняв решение по этой вероятности, т. е. отправив пакет с вероятностью p_0, следует ожидать результата, который даст возможность откорректировать значение p_0.

Если пакет был отправлен и возник конфликт, то это означает, что p_0, по-видимому, велико и ее надо уменьшить, т. е. заменить на p_1:

$$p_1 = \gamma p_0,$$

где $0 < \gamma < 1$.

Если конфликта не возникло, то величина p_0 наверное выбрана правильно и

$$p_1 = p_0.$$

Если же пакет не посылался, а послал кто-то другой (был конфликт или его не было, станцию не интересует), то «непосылка» правильна и, следовательно, $p_1 = p_0$.

Но если при отсутствии посылки пакета было пустое окно, то скорее всего его нужно было посылать, т. е. p_0 надо увеличивать:

$$p_1 = p_0 (1 - \gamma) + \gamma$$

(здесь $p_1 \geqslant p_0$ и $p_1 < 1$. Этим и определяется сложность формулы).

Таким образом, адаптация вероятности посылки пакета в $t+1$-м окне реализуется следующим выражением:

$$p_{t+1} = \begin{cases} \gamma p_t, & \text{если пакет был отправлен и возник конфликт} \\ p_t, & \text{если пакет был отправлен и не было конфликта или пакет не отправлен, но отправил кто-то другой} \\ p_t(1-\gamma)+\gamma, & \text{если пакет не отправлялся и было пустое окно} \end{cases}$$

Такие формулы называют рекуррентными. Они очень удобны для вычислений, так как позволяют определять последующее значение p_{t+1} по предыдущему p_t и ситуации, сложившейся к моменту $t+1$.

Работая таким образом, станции будут поддерживать вероятность p_t на уровне близком к p^*, т. е. канал связи будет работать почти оптимально (может быть, чуть хуже).

Является ли скорость 36,8% от максимальной предельной? И вообще, нельзя ли устранить конфликты в этой системе связи?

Можно, если предоставить возможность станциям договариваться о порядке работы. Сделать это можно, например, так.

В начале связи все станции посылают пакет с адресом «ВСЕМ». В этом пакете сообщается о том, сколько пакетов подряд будет передаваться этой станцией. Эти пакеты — заявки принимаются всеми станциями в описанном выше адаптивном режиме. Каждая станция, получая пакеты от других станций, строит очередь заявок в порядке их передачи через спутники. Точнее, сама очередь эту станцию не интересует — ей важно лишь определить момент, когда можно включаться в передачу. Этот момент вычисляется просто: он определяется суммой чужих заявок, принятых до удачной передачи заявки этой станции (удачная передача — это и есть постановка в очередь).

Например, приняв заявки трех станций на передачу 100, 150 и 500 пакетов и сообщив свою заявку (она будет четвертой), данная станция должна отсчитать с

момента начала передачи $100 + 150 + 50 = 800$ окон и на 801-м включаться в работу.

Очевидно, что расположение заявок при этом будет случайным. Но не это важно. Важно то, что эти очереди будут одинаковыми у всех станций. Это и есть договоренность. Теперь остается дождаться своей очереди, отослать уже без конкуренции, а следовательно, без конфликтов и пустых окон все свои заявленные пакеты и приготовиться к посылке очередной заявки после окончания передач, предусмотренных очередью.

Легко заметить, что скорость передачи при такой работе резко возрастает. Действительно, при передаче заявок она не превышает 36,4% от максимальной. Но это только для N пакетов (по одному пакету от каждой станции). Когда же начинается передача «по очереди», то скорость максимальна, так как ни конфликтов, ни пустых окон возникнуть не может.

Как видно, описанная система спутниковой связи ЭВМ вполне эффективно справляется с задачей пакетной связи ЭВМ и при этом не требует громоздкой иерархической системы уровней — здесь все значительно проще. Более того, установление в каждый момент одного канала связи позволяет этот канал сделать широкополосным (мешать-то некому), что значительно увеличивает пропускную способность этого канала.

Так, в сети ARPA широкополосный спутниковый канал связи с Гавайями позволяет пропускать информацию со скоростью 50000 бит в сек. Это значит, что стандартный пакет в 1000 бит требует окно размером всего в две сотых секунды.

Но за все надо платить. Приходится платить и за простоту спутниковой связи. И эта плата довольно тяжелая — четвертьсекундная задержка сигнала. Лишь через этот промежуток времени мы можем проверить правильность передачи пакета. Но зато этот пакет сразу будет на месте. При передаче по сети при этом будет загружено несколько узлов и в каждом из них должны сработать много программ, что, естественно, понижает надежность системы связи.

Как обычно, любая система связи имеет свои «за» и свои «против». Хорошее понимание этих ее свойств дает возможность хорошо воспользоваться всеми «за» и нейтрализовать «против».

Так, идея спутниковой связи в сети ЭВМ нашла свое применение при создании локальных сетей ЭВМ. В этом

случае все ЭВМ сети объединяются одним каналом (его называют «моноканалом») — электрическим, оптическим или радиоканалом. В него и посылают свои пакеты ЭВМ сети. Этот канал имеет малую протяженность, а поэтому не дает злополучной задержки спутникового канала.

Адаптация вычислительных сетей

Чем сложнее система и среда ее окружающая, тем труднее предвидеть, в какие ситуации она попадет в процессе своего функционирования и как будет вести себя. Это обстоятельство и заставляет вводить процедуры адаптации. Вычислительные сети являются сверхсложными системами и поэтому введение адаптации здесь просто необходимо. Один из примеров необходимости такой адаптации уже приведен выше. Это сочетание датаграммного и виртуального режимов работы транспортной сети. В зависимости от состояния загрузки адаптивная сеть может переходить с одного режима работы на другой или по одним каналам использовать датаграммный режим, а по другим — виртуальный.

Выбор того или другого режима или их оптимального сочетания в той или иной ситуации, сложившейся при функционировании сети, и осуществляется методами адаптации.

Рассмотрим некоторые другие аспекты применения адаптации в вычислительных сетях. Так как свойства сетей в значительной мере определяются протоколами, то естественно рассмотреть адаптацию протоколов.

Простейшей формой адаптации протоколов является их своевременная замена. Это означает, что при определенной ситуации, сложившейся в сети, все протоколы или их какая-то часть заменяются на другие протоколы. Реализовать эту адаптацию можно методом «двурукого бандита», рассмотренным в первой главе. Основания для такой замены имеются вполне определенные. Дело в том, что существует достаточно много различных протоколов, решающих одни и те же сетевые задачи. (Практически все существующие сети имеют свои протоколы управления процессами взаимодействия в сети И каждая из этих систем протоколов оптимальна в каком-то своем собственном смысле. Именно поэтому нельзя выбрать заранее наилучшую систему протоколов.)

Дело в том, что изменяются не только условия эксплуатации сети, ее структура и свойства, но и представление о том, «что такое хорошо и что такое плохо», т. е. критерии эффективности функционирования сети. В такой обстановке нужно иметь запас эффективных решений в виде различных комбинаций протоколов, чтобы поддерживать в сети именно ту комбинацию, которая наиболее эффективна по заданному критерию в сложившейся ситуации. Если число таких комбинаций равно m, то для их использования следует воспользоваться алгоритмом «m-рукого бандита».

Кроме того каждый протокол имеет параметры и альтернативные варианты, назначение которых происходит обычно достаточно волевым порядком, исходя из представлений об ожидаемых режимах работы вычислительной сети. Именно эти параметры и альтернативные варианты должны определяться адаптивным образом в зависимости от состояния сети и критерия эффективности.

Приведем несколько простейших примеров. При передаче пакета по каналу очень важно выбрать длину пакета. Эта длина зависит от интенсивности помех в канале, по которому будет передаваться пакет. Если помехи слабые, то лучше назначать большую длину пакета. При высоком уровне помех следует информацию передавать короткими пакетами. На первый взгляд напрашивается естественный подход: определить интенсивность помех в канале и на этой основе рассчитать оптимальную длину пакета.

Такой путь безусловно хорош, но только тогда, когда помеха стационарна, т. е. ее свойства не изменяются во времени или изменяются очень медленно. Если же помеха изменяется быстро, то оценка ее свойств так же быстро «стареет» и ее использование только портит дело. Вот и приходится адаптировать длину пакета в процессе эксплуатации канала связи, минуя стадию определения свойств канала.

Это можно сделать так. Сначала назначаются две из альтернативных длин пакета (они обычно предусматриваются протоколом второго уровня). Эти две альтернативы включаются в адаптацию по алгоритму «двурукого бандита», который определяет наилучшую из них. Следующая пара альтернатив образуется полученной лучшей и «соседней» с ней. Эта пара альтернатив снова вступает в «игру» и т. д.

Таким образом, независимо от уровня изменяющихся помех в канале будет поддерживаться именно та длина пакета, которая позволяет передавать информацию по каналу с максимальной скоростью.

Другой пример сетевой адаптации. В узле сети, как уже говорилось, расположена коммутационная машина, которая обрабатывает и направляет пакеты в нужном направлении. Все эти пакеты проходят через память коммутационной ЭВМ. Такая память является своеобразным бункером, куда складываются пакеты, ожидающие обработки и пересылки. Обработка пакетов происходит в соответствии с выбранным алгоритмом — дисциплиной обслуживания этих пакетов. Действительно, коммутационная машина является по сути дела пунктом обслуживания, а пакеты — клиентами, ждущими обслуживания. В зависимости от выбора дисциплины обслуживания результаты работы узла будут различными. Поэтому выбор такой дисциплины должен зависеть от ситуации, сложившейся в сети, и от критерия эффективности. Приведем альтернативные дисциплины обслуживания.

Простейшей и пожалуй, самой естественной будет дисциплина: «первый пришел — первым обслужен». Она «демократична» и всем пакетам уделяет равное внимание. Однако оптимальность этой дисциплины в критических ситуациях сомнительна. Действительно, при переполнении бункера или перегрузке одного из каналов возможна значительная задержка. И поэтому часто используют другую, обратную дисциплину «последним пришел — первым обслужен», которая отдает предпочтение новым пакетам ценой задержки «старых». Это делается для того, чтобы не допускать задержку для всех пакетов и если уж произошла задержка, то не ждать, что она сама рассосется (на это уйдет слишком много времени, в течение которого все пакеты будут задерживаться), а перейти на новую дисциплину, которая еще больше задержит имеющиеся в памяти пакеты, но зато не будет задерживать новые. Иногда такое поведение безусловно разумно и даже оптимально.

Таким образом, в вычислительной сети иногда складывается такая ситуация, которая требует смены дисциплины обслуживания пакетов в узлах сети. Реализовать это можно все тем же алгоритмом «двурукого бандита», у которого вместо «пистолетов в руках»

находятся те самые альтернативные дисциплины обслуживания, которые целесообразно применять и менять в узлах коммутации в зависимости от сложившейся ситуации (в узле и в сети) и критерия оптимальности функционирования узла или сети в целом.

И последний пример. Одним из важных свойств любой вычислительной сети является свободный доступ к информации, хранящейся во всех машинах сети (конечно, ограничения всегда существуют, но они имеют не технический, а специальный характер — просто хозяин этих данных ограничивает доступ к ним).

Банки данных в вычислительной сети обычно специализируются в зависимости от потребностей пользователей сети (например, по определенным разделам физики, химии, технологии и т. д.). Очевидно, что банки расположены в тех узлах сети, где сконцентрированы пользователи и составители этого банка. Так, например, банк по атомной и ядерной физике следует создавать на базе ядерного центра. Смешно его было располагать где-либо в другом месте, даже в соседнем городе. Так происходит специализация и фиксация места расположения банка. Его содержимым однако может пользоваться любой абонент сети.

Но при частом обращении к банку каналы связи с ним окажутся перегруженными и сеть с популярным банком данных окажется заблокированной. Для того, чтобы избежать такой перегрузки сети и не заставлять абонента долго ждать затребованной информации, естественно создать на абонентской машине небольшой банк данных, наиболее часто используемых этим абонентом. Большим он быть не может — для этого нужна большая память. Возникает задача, какую именно информацию хранить в локальном банке данных? Это зависит от потребностей пользователя, которые изменяются во времени. Вот и получается, что содержимое такого индивидуального банка должно адаптироваться к потребностям пользователя, т. е. изменяться вместе с изменением его потребностей, которые проявляются в виде запросов.

Такой адаптивный локальный банк данных сразу обеспечивает пользователя часто запрашиваемой информацией, а за редкой он обращается по сети к центральному банку данных по этой проблеме, минимизируя тем самым среднее время ответа на запрос и загрузку транспортной сети.

Как видно из примеров, адаптация здесь выступает в роли «усилителя возможностей». Это означает, что при введении адаптации эффективность вычислительной сети повышается и ее возможности расширяются не только количественно, как при адаптации длины пакета или содержимого локального банка данных, но и качественно — обеспечивая функционирование там, где оно без адаптации будет нарушено, как при адаптивной смене дисциплин обслуживания. Вот и получается, что адаптация является обязательным компонентом вычислительной сети, обеспечивающим стабильность и гибкость ее функционирования, независимо от состояния среды и критериев эффективности.

Мегрэ выходит в сети

Справочная разъяснила Мегрэ принципы обработки информации с помощью сетей (они описаны выше) и указала, что требуемую ему мощность могут обеспечить семь вычислительных сетей:

ARPA (США), имеющая спутниковую связь с Европой через Лондон;

CYCLADES (Франция), имеющая абонентскую машину в Гренобле (Мегрэ отметил себе это обстоятельство, ведь программа идентификации находится именно в Гренобле);

EVRONET (Европейское экономическое сообщество) имеет 4 узла в Лондоне (а это связь с ARPA отметил себе Мегрэ), Париже (связь с CYCLADES), Франкфурте и Риме;

EIN (Международная европейская вычислительная сеть) имеет 5 узлов и среди них Лондон, Париж и Цюрих («Вот и выход на банк данных Интерпола» — подумал Мегрэ);

TRANSPAC (Франция) — 12 узлов в основных городах Франции («Берем всю» — усмехнулся Мегрэ);

TELENET (США) — 20 узлов, выхода в Европу нет («Свяжемся с ней через сеть ARPA»);

DATAPAC (Канада) — 4 узла, имеется связь с сетями ARPA и TELENET («А, следовательно, и с Парижем через Лондон с помощью TRANSPAC и ARPA» — отметил Мегрэ).

Ему понравилось, как коммуникационные каналы сетей связывают его с тысячами ЭВМ, расположенных на двух континентах и способных выполнить его зада-

ние. Осталось заставить их сделать это, что зависело не только от него. Он набрал номер.

— Алло, полковник, расследование дела об ограблении банка требует обработки видеоинформации, полученной телесторожем. Прошу Вашей санкции на выход в вычислительные сети ARPANET,... и ДАТАРАС под девизом Интерпола.

— Сколько из этих сетей за пределами Франции? — спросил полковник.

— Три,— не моргнув глазом ответил Мегрэ, подумав, что международные сети нельзя считать находящимися за французской границей, если есть хотя бы один терминал во Франции.

— Не боитесь утечки информации? Ведь бандиты, узнав о существовании видеопленки, могут что-то предпринять!

— Они не успеют — спокойно ответил Мегрэ.— Ведь осталось всего полтора часа.

— Да, и не минуты больше,— отчеканил полковник и строго официальным голосом закончил: — Инспектор Мегрэ, санкционирую выход на указанные Вами вычислительные сети с полученными видеоматериалами под девизом Интерпола. Все.

Мегрэ усмехнулся. Он представил какой кислой станет физиономия полковника, когда он получит счет от всех семи сетей. С ценами его познакомила справочная. Эти цены зависели от приоритета задания. Самый высокий (нулевой) приоритет прерывал все процессы кроме процессов управления в сети. И обходилось это очень дорого, т. к. в этом случае сети приходилось выплачивать штрафы всем ее пользователям с более низким приоритетом, чьи задачи задерживались на время работы программ с нулевым приоритетом.

— Ну что, надо начинать.— Мегрэ набрал номер профессора Мариака.

— Алло, профессор, будем считать на семи вычислительных сетях. Бандитов тоже семь, так что мы эту «великолепную» семерку разбросаем по всему миру. Об этом позабочусь я. А Вам следует (извините за жаргон, но я уже включил Вас в группу розыска) направить программу идентификации диспетчерам всех семи сетей. Укажите программе, что идентифицировать надо человека в белом контуре — я их обведу. И последнее, идентифицировать надо лишь отклонения от нормы.

При достоверности свыше 99% результаты следует занести в формат Интерпола. О его отправке в банк данных я позабочусь сам. Формат вы получите сейчас. Действуйте, профессор!

— Поль! Отправьте по видеотелефону форматы Интерпола в Гренобль для профессора Мариака,— бросил Мегрэ в селектор. — А теперь займемся дележом бандитов.

Мегрэ прошел в аппаратный зал, сел за графический дисплей и взял в руку световое перо. На экране возникла уже знакомая картина операционного зала. Он выбрал наугад фигуру, стоящую у двери, и обвел ее пером (это была небольшая, размером с карандаш, палочка, пишущий конец которой имел миниатюрный фотоэлемент, а из другого конца выходил тонкий шнур, уходящий куда-то за экран. При прикосновении фотоэлементом пера к экрану в этой точке загорелась яркая звездочка. Стереть ее можно только световой «резинкой», функцию которой выполняло то же световое перо, но работающее по другой программе — ее надо включать кнопкой «стереть». Этим пером можно делать на экране любые контурные фигуры. Так кадр за кадром Мегрэ обводил ярким кружком одного и того же человека, чтобы он ни делал. Покончив с одним, он вернул видеопленки в исходное состояние и занялся другим бандитом и т. д. На эту не очень интересную работу ушло минут двадцать. Остался час.

А тем временем Поль отправлял размеченные фильмы диспетчерам вычислительных сетей и добавлял, что результат идентификации должен быть направлен в банк данных французской криминальной полиции в Париже и одновременно в международный банк данных Интерпола в Цюрихе для поиска досье наиболее «подходящих» кандидатур:

«Эти досье направлять сразу на видеотерминал инспектора Мегрэ в Париже. Его же необходимо информировать о ходе вычислительного процесса. Окончание расчета — по получении результата или, если результата не будет, то через час».

Теперь осталось только ждать. На экран стали поступать сообщения о том, что задача принята и начала выполняться («Молодец, профессор, успел во время»). Указывалось и число машин, которые решали задачу. Это количество все время изменялось, видно, что-то мешало сети отдать всю мощность одной задаче не-

смотря на ее нулевой приоритет. Мегрэ этому не удивлялся, он хорошо понимал, что в большой сложной системе всегда что-нибудь будет происходить, непредусмотренное программой. А вычислительные сети близки к пределу сложности систем, созданных человеком.

Число машин, решающих задачу, колебалось около 500÷600. Только один раз их было 900 штук. «Вот тебе и реклама» — подумал Мегрэ, вспомнив ответы справочной.

На 17-й минуте отключилась сеть ARPA. И хотя вскоре снова включилась, но выключилась EVRONET, сообщив, что прерывает свою работу на неопределенное время.

На тридцатой минуте ARPA выдала результат и отключилась, оставив на экране такой счет, от которого Мегрэ поперхнулся кофе.

Ответ из Парижа и Цюриха был обильный и невразумительный. Полученным данным соответствовали 83 человека из Парижского банка и 112 из Цюриха.

— Вот тебе и достоверное отклонение — присвистнул Мегрэ. — Поль, пусть ARPA продолжит счет до 99,9% достоверности, а из этих,— он кивнул на экран — попросите лишь парижан, может что и найдем.

На 43-й минуте сеть EIN получила результат. Парижское досье по этим результатам показало на Весельчака Жака, старого «знакомого» Мегрэ. Он тут же вспомнил, что Жак немного хромал и, наверное, именно эту примету выявила программа Мариака.

В груди похолодело. Мегрэ знал, что так бывает, когда он нападал на след. Жак, судя по всему, вполне мог попасть в эту компанию, хотя обычно «работал» один. Специализировался он по мелким ограблениям и поэтому подолгу не сидел.

— Поль! Кто дежурит сейчас на улице Капуцинов? Срочно передайте: взять Жака Волье, да, Весельчака, дом 16, и сообщить мне немедленно. Пусть ищут поблизости от дома: он не будет прятаться, ему нужно алиби!

Через 10 минут, которые тянулись очень долго для Мегрэ, поступило сообщение, что Жака взяли.

— Соедините меня с ним! Алло, Жак. Это говорит инспектор Мегрэ. Ты узнаешь меня? Хорошо. Слушай очень внимательно. С этого момента твоя судьба зависит только от тебя. Ты первый, кого мы взяли по делу ограбления банка, которое было совершено два часа

назад. Да не перебивай ты меня! Мы вас вычислили, слышишь? Вычислили на ЭВМ, на многих ЭВМ. И здесь у меня лежит список твоих друзей.— Мегрэ указал на гору бумаги, лежащую на полу перед печатающим устройством, из которого вылезала очередная порция с очередным досье, найденным по результатам, полученным какой-то сетью.

— Мне нужно только уточнить список. И запомни, еще такой возможности помочь мне у тебя не будет. Все что ты скажешь, нигде не будет записано. Это я гарантирую. Мне нужно только подтверждение. Алло, Луи, переключите передатчик на секретный канал и выйдите из машины. Пусть Жак останется один с микрофоном. Алло, Жак, я слушаю имена.

— Инспектор, у меня железное алиби. Что Вы ко мне пристаете?

— И еще Жак, помнишь малютку Марго? Ведь дело можно было повернуть и иначе. И у меня есть материалы для этого поворота. Ты знаешь, чем это пахнет. Итак, тебе на раздумье 30 секунд. Больше дать не могу — у самого нет.— Мегрэ взглянул на часы, осталось 3 минуты до срока, указанного полковником.

После недолгого молчания Жак заговорил. Мегрэ быстро записал имена на клочке бумаги и бросился к списку подозреваемых, отпечатанных на ленте. Быстро сверив свою запись со всеми списками он задумчиво достал трубку, зажег зажигалкой смятый клочок бумаги и уже ею стал раскуривать трубку. Затянувшись два раза, он взглянул на часы. Срок истек.

Мегрэ подошел к телефону и набрал номер полковника. Он был занят. Нажав кнопку аварийного вызова, он соединился с ним.

— Инспектор Мегрэ, я жду объяснений,— резко сказал полковник.

— Докладываю: задержан один из бандитов — Жак Волье. Остальные известны и их розыск начинается немедленно, — по-военному отрубил Мегрэ.

Отрезвимся немного...

Читатель, наверное, давно заметил, что история с Мегрэ придумана автором для того, чтобы показать возможности вычислительных сетей. Неизбежное преувеличение здесь было лишь приемом, чтобы выпукло показать основную мысль о больших возможностях,

которые могут открыть вычислительные сети. А современные сети пока не столь удобны для несведущего пользователя, как это ему хотелось бы. Для работы с сетью надо (увы) много знать, учитывать и предвидеть.

Хочется закончить этот раздел словами одного из крупнейших теоретиков вычислительных сетей Л. Клейнрока, сказанными им в 1978 г.:

«В заключение следует подчеркнуть, что «идеальное» коллективное пользование вычислительными средствами сети в существующих сетях пока еще не реализовано. Можно много отдать за такую сеть, которой можно было бы представить задание, для того, чтобы его эффективное выполнение было организовано автоматически, и положиться на то, что сеть сама подберет подходящие ресурсы для выполнения этого задания. Пока же необходимо самому указать ЭВМ, в которой программа должна храниться, ЭВМ, на которой должна выполняться работа, место, где должны храниться полученные результаты, место, где эти результаты должны быть распечатаны, а также указать время, когда это все должно произойти».

Хочется надеяться, что в ближайшие годы положение изменится к лучшему.

Что же дальше? (Заключение)

Итак, мы проследили динамику и диалектику появления электронных вычислительных машин, систем и сетей. Они возникли по воле необходимости как, наверное, создается все, что является плодом рук человеческих. А неумолимые законы диалектики неизбежно пророчат появление новых потребностей человечества и удовлетворяющих их новых... Автор здесь затрудняется назвать то, что должно появиться завтра в ответ на сегодняшние запросы.

А запросы сегодня большие. Уже появились задачи, которые не могут быть решены даже с помощью сетей ЭВМ. Примером такой задачи является задача автоматического управления современным аэропортом. Она требует быстродействия 10^{14}, т. е. сто миллиардов операций в секунду. Напомним, что вычислительная мощность самой большой вычислительной сети ARPA равна 350 миллионов операций в секунду (по данным 1975 года). Нужно триста таких сетей, чтобы справить-

ся с задачей автоматического управления одним аэропортом. Нереальность такого решения очевидна. Так что, надо отказаться от задач, требующих больших вычислительных мощностей, или ждать, когда появятся ЭВМ еще большей мощности?

Дело в том, что ждать особенно нечего. Современная вычислительная система близка к пределу физических возможностей электронных схем. Этот предел связан с конечностью скорости распространения электрического сигнала и катастрофическим увеличением тепловых шумов при уменьшении размеров микросхем. Эти шумы являются электрическим прообразом броуновского движения, от которых избавиться нельзя и которые «охраняют» подступы к микромиру. Чем меньше микросхема, тем больше уровень тепловых шумов и тем труднее с ними бороться. И, наконец, с некоторого уровня микроминиатюризации затраты на борьбу с шумами станут слишком велики, т. е. не окупятся получаемым эффектом быстродействия. Это и есть предел для микросхем. Лучше уже сделать ничего нельзя. (Правда, есть запасной вариант перехода к криогенной технике, то есть к снижению тепловых шумов путем глубокого охлаждения электронных схем ЭВМ. Но и он по оценкам физиков не позволит поднять быстродействие ЭВМ выше 250 миллионов операций в секунду. Изобретение голографической памяти, картинной световой логики и световодов безусловно отодвинет этот барьер, но не снимет его).

Вот и получается, что осталось всего каких-нибудь два — три порядка увеличения для быстродействия ЭВМ и дадутся они с большим трудом. Так всегда бывает, когда приближаются к пределу, установленному природой.

Что же дальше? Смирить свои разбушевавшиеся потребности или искать другие пути обработки информации, позволяющие если не перешагнуть, то обойти этот критический порог. У человечества пока нет опыта «смирения». Его бурная история показывает, что выход всегда находится. Есть он и здесь.

Это — адаптация (или обучение, что по сути дела одно и то же), т. е. приспособление машины к решаемой задаче, к сложившейся ситуации, к новым целям, поставленным перед ней.

Мы уже говорили об адаптации существующих ЭВМ, вычислительных систем и сетей. Но эта адаптация

была ограничена жесткими рамками имеющейся структуры, системы команд, типом памяти и т. д.

Адаптироваться здесь можно только в рамках заданной структуры, т. е. только варьируя программой. Но, варьируя программу, нельзя изменить саму машину. И хотя адаптация программы и программного обеспечения ЭВМ, вычислительных систем и сетей ЭВМ безусловно позволяет значительно улучшать их свойства, но не в решающей мере. Так, адаптируя программу можно повысить ее эффективность в два-три раза, но не на два-три порядка.

Для того, чтобы получить очень сильный эффект от введения адаптации, надо варьировать структурой вычислительных средств, т. е. нужно иметь возможность изменять сами схемы, из которых составлена вычислительная техника. Таких возможностей современные ЭВМ не предоставляют и не могут предоставить. В этой «окаменелости» структуры и состоит слабость электронных компьютеров, которая в настоящее время приводит к исчерпанию их возможностей.

Как видно, эра электронных машин близится к своему концу. Они выполняют свою функцию там, где... выполняют, но для решения новых задач (сверхзадач) нужно привлекать новые средства, позволяющие адаптировать структуру компьютера. Есть ли такие средства?

Да, есть. Это оптоэлектроника — новое направление в вычислительной технике. Сочетание оптических и электрических процессов позволяет создавать принципиально новые функции и элементы, не доступные обычной электронике, и в конечном счете создать вычислительные машины с чрезвычайно гибкой структурой, которую легко адаптировать, т. е. перестраивать на эффективное решение различных задач.

Эти новые компьютеры структурно очень сильно отличаются от «старых» схем. Так, память здесь распределена по всем элементам машины, а не сосредоточена в одном месте, как в ЭВМ. Каждый элемент может в широких пределах изменять свою функцию, т. е. адаптировать ее к новым задачам. Элементы такой новой машины должны обладать большой связностью, т. е. возможностью непосредственно обмениваться информацией с большим числом других элементов этой машины.

Не заметил ли читатель, как все это очень напоминает устройство мозга? Это не случайное совпадение.

Наш мозг является пока самой совершенной вычислительной машиной (заметим, что сравнение старой ЭВМ с мозгом не выдерживает самой элементарной критики. Структура ЭВМ никак не напоминает структуру мозга и это сравнение скорее эмоционального толка, чем рационального).

Идея нового компьютера, принципиально отличающегося от ЭВМ, была навеяна вовсе не анализом структуры мозга. К ней пришли в результате сложной и мучительной переоценки тех ценностей, которые были созданы за последние тридцать лет. Бионика здесь оказалась вовсе не при чем. И то, что схема нового компьютера напоминает структуру мозга, является лишним доказательством правильности выбранного нового направления развития вычислительной техники.

Таких машин пока еще нет, но уже готовы их оптоэлектронные элементы. Расчеты показывают, что при решении некоторых задач такие адаптивные (или обучающиеся) машины позволят повысить быстродействие на несколько порядков. Так, например, при решении задачи синтеза одного предсказывающего автомата сверхмощная вычислительная система типа БАРРОУЗ должна затратить 1000 часов, а оптоэлектронному компьютеру весьма скромных габаритов (примерно со средний телевизор) понадобится для этого несколько... секунд! Структура такого компьютера изменяется на 30% на каждом такте его работы. Такое глубокое изменение структуры в процессе адаптации и отличает новый компьютер от «старых» ЭВМ. А, как известно, изменяя структуру, можно получить любые схемы, т. е. любые вычислительные машины. Именно здесь открывается совершенно новая возможность каждый раз для новой задачи строить новую машину, адекватную задаче. Осуществить это можно только методами адаптации структуры связей и функций элементов этой машины.

Так адаптация становится решающим фактором коренного изменения направления развития современной вычислительной техники (не вычислительной — тоже).