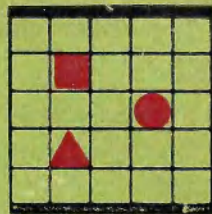


БИБЛИОТЕКА

КИБЕРНЕТИЧЕСКОГО

СБОРНИКА



Супервизоры и операционные системы



COMPUTER MONOGRAPHS

10

GENERAL EDITOR: STANLEY GILL,
ASSOCIATE EDITOR: J. J. FLORENTIN

**EXECUTIVE
PROGRAMS AND
OPERATING SYSTEMS**

edited by
G. CUTTLE
&
P. B. ROBINSON

MACDONALD : LONDON
AND
AMERICAN ELSEVIER INC. : NEW YORK
1970

БИБЛИОТЕКА _____
„КИБЕРНЕТИЧЕСКОГО
СБОРНИКА“ _____

СУПЕРВИЗОРЫ И ОПЕРАЦИОННЫЕ СИСТЕМЫ

Под редакцией
ДЖ. КАТТЛА и П. РОБИНСОНА

Перевод с английского
Г. Н. ЕЗЕРОВОЙ и Вик. С. ШТАРҚМАНА

Под редакцией
Вс. С. ШТАРҚМАНА

**Издательство „Мир“
Москва 1972**

Коллективная монография, авторами которой являются известные английские специалисты, охватывает круг вопросов, связанных с созданием супервизоров и операционных систем, играющих важную роль в программном обеспечении вычислительных машин.

После рассмотрения основных концепций, положенных в основу конструирования операционных систем, и освещения роли супервизоров в качестве иллюстраций приводятся конкретные операционные системы. Обсуждается идея «разговорных» компиляторов и указываются специальные требования, предъявляемые к средствам программного обеспечения для управления вычислительной системой при работе в реальном времени.

Книга представляет большой интерес для специалистов по вычислительной технике и программированию. Она будет полезна студентам и аспирантам соответствующих специальностей.

Редакция литературы по математическим наукам

ПРЕДИСЛОВИЕ РЕДАКТОРА ПЕРЕВОДА

История развития электронных вычислительных машин только еще преодолевает свой 30-летний рубеж, а мир уже стал свидетелем смены трех «поколений» машин.

Первое поколение (1949—1950) — машины на электронных лампах, второе (1950—1964) — машины на полупроводниковых дискретных элементах и третье (с 1964 г. до настоящего времени) — машины на интегральных схемах.

Термин «поколение» вошел в употребление вместе с появлением машин третьего поколения в 1964 г., и, хотя первоначально он связывался с изменением технологического уровня производства аппаратуры, в настоящее время им принято характеризовать весь комплекс программно-аппаратных средств, составляющих современную вычислительную систему.

Первое поколение машин не имело систем программного обеспечения. Все программирование для этих машин проходило на уровне пользователя. Со вторым поколением машин родилось системное программирование, т. е. создание библиотек программ, трансляторов с различных языков программирования и, наконец, создание мониторных систем, управляющих самим процессом прохождения задач через машину и обеспечивающих тот или иной уровень автоматизации функций, которые на машинах первого поколения выполнялись человеком-оператором.

Эти мониторные системы явились родоначальниками операционных систем для машин третьего поколения, основные функции которых можно сформулировать следующим образом:

- инициация и завершение выполнения задач пользователя;
- управление ходом их выполнения;
- обработка различных особых ситуаций, которые могут возникнуть в процессе работы (например, обработка ошибок в программе или в аппаратуре);
- распределение ресурсов машины между задачами;
- обеспечение возможности использовать различные программные средства (файлы, архивы, трансляторы, системы отладки и т. п.);

- взаимная защита программ и информации, принадлежащих различным пользователям;
- регистрация и учет всей выполняемой работы, позволяющие, в частности, провести необходимые финансовые расчеты с пользователями системы.

Как нетрудно видеть, это те функции, которые, с одной стороны, нельзя выполнить в программах пользователя и которые, с другой стороны, слишком сложны, чтобы их можно было реализовать чисто аппаратными средствами.

Поэтому несколько с другой точки зрения операционная система — это часть общего программного обеспечения машины, которая:

- занимает промежуточное положение между аппаратурой и программистом — пользователем машины;
- дополняет аппаратуру теми возможностями, которые трудно или экономически невыгодно реализовывать чисто аппаратными средствами;
- создает условия для эффективной разработки и отладки программ;
- снижает стоимость использования вычислительных средств за счет разделения между различными пользователями ресурсов машины и накопленной информации.

Таким образом, операционная система — это основа всей системы программного обеспечения современной ЭВМ, определяющая операционную обстановку, в которой работают операторы, программисты, инженеры и даже руководители вычислительного центра. Нет ничего удивительного в том, что операционные системы оказывают огромное влияние на развитие как аппаратных возможностей машин, так и систем программирования.

В первых операционных системах осуществлялась так называемая пакетная обработка задач. Пакет заранее заготовленных программ (точнее работ) вводился в машину, и эти программы выполнялись без какого-либо оперативного взаимодействия с программистом.

Это было большим прогрессом, так как позволяло автоматически переходить от одной работы к следующей, свести к минимуму простой машины при смене задач и увеличить ее производительность. Однако то оперативное взаимодействие, тот диалог между машиной и человеком, который был возможен в машинах первого поколения, когда программист сам выполнял функции оператора за пультом, был нарушен. Поэтому следующим шагом в развитии операционных систем было восстановление потерянного оперативного взаимодей-

ствия человека с машиной, т. е. появление систем с разделением времени, рассчитанных на одновременное обслуживание многих пользователей, каждый из которых работает за своим выносным пультом, например телетайпом.

Существующие в настоящее время операционные системы отличаются очень большим разнообразием сочетаний диалоговых и пакетных возможностей, степенью универсальности, различной способностью адаптироваться к изменяющимся условиям эксплуатации, структурой внутренней организации, алгоритмами функционирования и другими параметрами. Однако, несмотря на это разнообразие, представления об основных функциональных возможностях операционных систем можно считать достаточно сложившимися, и можно ожидать в ближайшем будущем появления работ, систематизирующих множество методов реализации этих возможностей.

Предлагаемая вниманию читателя книга дает достаточно полное представление о круге идей, относящихся к операционным системам. В ней вводятся и раскрываются такие фундаментальные понятия, как мультипрограммирование, приоритетное планирование, автономизация ввода-вывода, язык описания работ, мультидоступ, реальное время, архив файлов, шаговые диалоговые трансляторы и многое другое.

В процессе изложения авторы часто обращаются к истории развития ЭВМ, показывая, как жизнь выдвигала те проблемы, которые решаются в современных машинах операционными системами.

Книга рассчитана на читателя, знакомого с ЭВМ либо по линии программирования (системного или прикладного), либо по линии разработки аппаратуры, либо по линии инженерной эксплуатации. Однако глубина этого знакомства может быть самой различной. Человека с небольшим опытом книга введет в общий круг идей операционных систем. Прикладному программисту она даст возможность почувствовать ту проблематику, с которой сталкиваются системные программисты. Ему будет легче представить перспективы развития операционных систем и более грамотно подходить к оценкам возможностей тех систем, с которыми он сталкивается на практике. Что же касается системных программистов, то они оценят в книге систематичность и широту освещения вопроса.

Главы 1, 3, 8—10 переведены Вик. С. Штаркманом, главы 2, 4—7 переведены Г. Н. Езеровой.

Вс. С. Штаркман

ПРЕДИСЛОВИЕ

Супервизоры и операционные системы относятся к быстро развивающемуся направлению системного программирования. К моменту написания этой книги большинство наиболее высокоразвитых систем находилось в эксплуатации относительно короткий промежуток времени, и лишь немногие из них (например, супервизор машины АТЛАС) можно считать прошедшими через все стадии совершенствования и полностью сформировавшимися. Поэтому среди разработчиков до сих пор существует большое разнообразие подходов к проектированию систем. И хотя весь спектр возможностей, которые должны обеспечивать операционные системы, уже четко вырисовывается, остается широкое поле деятельности по наилучшей реализации этих возможностей.

В связи с этим при подготовке книги мы столкнулись с двумя серьезными проблемами. Нужно ли пытаться описать все существующие методы или сконцентрировать внимание на одном из них? Должны ли мы остановиться на окончательно выработавшихся точках зрения или же нам следует также попытаться передать целый ряд идей, находящихся в стадии развития?

Для обеих проблем мы выбрали компромиссное решение. Чтобы проследить некоторые идеи достаточно глубоко, мы сконцентрировали свое внимание на одном ряде операционных систем, а именно на системах GEORGE для машин серии ICL 1900. Однако в книгу включены и главы, имеющие целью сравнить эти системы с одним из иных основных подходов.

Выбор систем серии 1900 как основы для этой книги был отчасти обусловлен удобством (оба автора работают в фирме ICL) и отчасти тем, что стадия их разработки чрезвычайно соответствовала нашим намерениям: во время написания книги работа над системами GEORGE 1 и GEORGE 2, имеющими более простые возможности, была полностью завершена, а система GEORGE 3 находилась в последней стадии отладки, и это позволило нам включить в книгу самые новые идеи и возможности.

Для того чтобы как можно шире охватить разнообразие идей, разные главы написаны разными авторами, каждый

из которых активно участвует в различных видах работ по операционным системам и супервизорам; мы же взяли на себя лишь функции редакторов. По очевидным соображениям были привлечены авторы из фирмы ICL, но мы умышленно пытались подобрать их из разных отделов компании, с тем чтобы иметь возможно более широкую трактовку вопросов. Естественно, что результатом явилось разнообразие стилей и точек зрения, но мы надеемся, что это можно оправдать, имея в виду непрерывное развитие и изменение взглядов, что является отличительной чертой этой области знаний в наши дни.

*Дж. Каттл
П. Робинсон*

ВВЕДЕНИЕ В ОПЕРАЦИОННЫЕ СИСТЕМЫ. ОСНОВНЫЕ ПОНЯТИЯ

Мартин Уорвик

Чтобы ввести основные понятия операционных систем, Мартин Уорвик вначале задается вопросом: «Зачем нужна операционная система?» Он проследживает историю развития и возникновения трех основных типов современных операционных систем: пакетной обработки, коллективного пользования и реального времени. Затем он рассматривает различные группы людей, так или иначе связанных с операционными системами, и анализирует специфику их требований. В заключение обсуждается вопрос, во что обходятся услуги операционной системы и какими качествами должна обладать хорошая операционная система.

1.1. Введение

В последнее время поняли и признали значение операционных систем. В современном вычислительном центре создание соответствующей «операционной обстановки» не менее важно, чем то, какая применяется машина. Понимание этого факта привело к появлению ряда операционных систем, предоставляемых разработчиками машин, причем наиболее развитые системы свели к минимуму вмешательство человека в управление вычислительной машиной, а работу оператора сделали элементарной. Операционная система любой вычислительной установки представляет собой определенную организацию для выполнения вычислительных работ, которая в сфере своей деятельности вовлекает некоторый комплекс, состоящий из людей, оборудования и программ. Хотя в этой книге нас интересуют главным образом операции, которыми управляет сама машина, мы все же должны иметь в виду общую операционную картину всей вычислительной установки. Ни одна установка полностью не автоматизирована, но с ростом сложности и скорости вычислительных машин требование автоматизации приобретает все большее значение. Операционная система, которую поставляет изготовитель машины, служит лишь базой, на которой можно строить весь организационный комплекс.

В этой главе мы рассмотрим общие требования к операционным системам, но это не значит, что все эти требования должны удовлетворяться в каждой конкретной системе. Система должна быть спроектирована так, чтобы обеспечить наилучшее использование ресурсов установки, людей, оборудования и времени. Анализ системы требует ответов на

несколько фундаментальных вопросов. Зачем нужна операционная система? Кому она нужна и для чего? Сколько она стоит? Какие свойства делают операционную систему хорошей? Системы, описанные в последующих главах, должны рассматриваться в свете ответов на эти вопросы.

1.2. Зачем нужна операционная система?

Первые вычислительные машины были всего лишь совокупностью необходимых устройств, и описание любой выполняемой работы конкретизировалось непосредственно до уровня машинных операций. Каждый программист был экспертом по машине, знал все ее хитрости и сам работал за пультом. С возникновением библиотек программ общего пользования, которые стали частью вычислительной установки, операции стали более унифицированными. Это позволило воспользоваться услугами оператора-профессионала, обученного стандартной операторской работе. Программист, придерживающийся некоторых соглашений, мог сэкономить свои усилия и время, пользуясь стандартными программами для выполнения часто встречающихся операций. В функции оператора входило наблюдение за машиной, картами, бумажной или магнитной лентой, печатающим устройством, а также регистрация пользователей. Машины были медленными, и эффективность работы оператора не имела большого значения. Но когда на смену первому поколению пришло второе поколение машин, скорость которых увеличилась в 100 раз, встали новые проблемы. В частности, нельзя было больше пренебрегать неэффективностью работы оператора. Если запуск задачи занимает 5 минут, а счет выполняется за 2 часа, то пропадает лишь немногим более 4% полезного машинного времени. Однако если для запуска короткого арифметического счета требуется 5 минут, а счет выполняется за 30 секунд, то в действительности весь день расходуется на работу оператора, а вычислительная машина в основном простаивает. В этом случае очень важно работу по запуску задач поручить машине. Однако разделение функций управления между оператором и машиной сразу же порождает новые трудности. Программист может злоупотреблять машиной и без ведома оператора изменять последовательность выполнения работ или незаконно использовать ленты и т. д. Такого сорта трудности можно преодолеть только в том случае, если функции управления программой будут «обезличены» и перейдут от программиста к операционной системе.

На этом пути возникла концепция машины с двумя уровнями: уровнем прерываний (или супервизорным уровнем) и пользовательским уровнем. При определенных условиях ма-

шина должна аппаратно переключаться с последовательности команд пользователя на команды супервизорной программы. Такими условиями могли бы быть, например, попытки выполнить определенного типа команды, с помощью которых система управляет собственной работой, или попытки неправильно использовать аппаратуру; такие условия также могли бы возникать после выполнения заданного количества команд, либо по истечении определенного времени. Обладая такими возможностями, система способна осуществлять строгий контроль за прохождением программы пользователя. Она может устанавливать ограничения по времени на пользователя и вылавливать ошибки в программе, она также может сообщать оператору о действиях, которые она сама выполняет, и сообщать о действиях, которые требуются от оператора.

Кроме увеличения скорости, для машин второго поколения характерно то, что их устройства ввода-вывода стали автономными и теперь могли работать параллельно с процессом выполнения команд. Этим свойством можно воспользоваться двумя путями: во-первых, все программы нужно по возможности писать так, чтобы процесс выполнения команд всегда шел во время ввода или вывода (для чего потребовались известные методы двойной буферизации и балансирования времени передачи данных и времени их обработки), во-вторых, нужно использовать мультипрограммирование. Супервизорная программа (для машины с двумя уровнями) вполне в состоянии зарегистрировать, что некоторая пользовательская работа приостановилась в ожидании ввода или вывода. Если в это время в машине находится другая работа, то процессор может переключиться на нее на время, пока не кончится передача данных первой работы. Это и есть мультипрограммирование. Для многих машин второго поколения это же называли разделением времени (time-sharing), но сейчас термин «разделение времени» общепринято связывать с мультидоступом (multiaccess), или коллективным использованием.

Для машины с двумя уровнями нужна резидентная¹⁾ управляющая программа. В разных машинах она называлась по-разному: исполнительная (Executive), супервизорная (Supervisor), программа-директор (Director), руководящая программа (Master Routine), управляющая программа (Master Control Program). С появлением резидентных управляющих программ возможности операционных систем существенно возросли. У пользователя не было иного выхода, кроме безропотного принятия условий, диктуемых резидентной про-

¹⁾ Резидентная программа — программа, постоянно присутствующая в оперативной памяти. — *Прим. перев.*

граммой. Операционная система получила теперь все полномочия, на то, чтобы заведовать прохождением программ, решать, какие программы загружать и в каком порядке, реагировать на незаконные действия, завершать работу программ, следить за соблюдением программистами границ использования памяти и времени, контролировать использование периферийных устройств.

Благодаря резидентной управляющей программе стало возможным автоматическое управление установкой. Управление вычислительной системой теперь перешло из рук пользователя в руки создателя операционной системы. Между пользователем и аппаратурой машины был введен искусственный интерфейс¹⁾, и любая работа, не отвечающая его требованиям (не важно, из-за ошибок в описании работы на языке управления работами или в самой программе), не допускалась к счету. Резидентные управляющие программы как бы ставили свою «авторскую печать» на операционные системы. Их действия и влияние сказывались на всем. Для пользователя они являлись фактически продолжением аппаратуры.

Третье поколение машин возникло на основе дальнейшего развития принципов наиболее совершенных машин второго поколения. Желание свести к минимуму вмешательство человека в управление и облегчить работу пользователя привело к значительному увеличению размеров и расширению функций операционных систем. Аппаратная часть машины все больше удалялась от пользователя, и ее место заняла операционная система. Для каждого типа пользователя она предоставляла некоторый интерфейс. Для программиста во многих случаях — это файлы на уровне логической структуры и язык управления работами. Существующие операционные системы можно грубо расклассифицировать, используя следующие три типа и их комбинации.

Первый тип — пакетная обработка (Batch) — получил название от способа использования. Пакет работ подается на машину, и работы выполняются одна за другой в порядке, избранном операционной системой, обычно в режиме мультипрограммирования. Такого типа операционная система имеет много свойств, необычных для пользователя прежних машин. Система «проглатывает» работу пользователя и позже выдает результаты в виде переработанных файлов и в виде печатного документа. В машинах первого поколения пользователь

¹⁾ Термин «интерфейс» (interface) часто употребляется в английской и американской литературе для обозначения некоторого способа связи между двумя объектами: человек — человек (всякого рода документация), человек — машина (всевозможные языки программирования и т. п.), процессор — терминал (канал связи) и т. д. — *Прим. перев.*

наблюдал за процессом выполнения своей программы, следил, верно ли она идет, вводил новые параметры, меняющие ее ход, оперативно вносил в нее исправления.

Второй тип системы — мультидоступ (или коллективное пользование) — позволяет восстановить нарушенный контакт пользователя с машиной. Каждому пользователю дается некоторое терминальное устройство ввода-вывода, телетайп или дисплей¹⁾, и имеющаяся в наличии вычислительная мощность распределяется между всеми пользователями. Каждый раз при создании новой системы или расширении старой возникают новые проблемы. Связь с терминалами — одна из них. Вообще, их функционирование отличается от работы стандартных устройств ввода-вывода и обычно осуществляется через специальный мультиплексный процессор. То, что система обслуживает сразу многих пользователей, также создает большие проблемы в использовании оперативной памяти и хранении файлов во вспомогательной памяти. При таком большом количестве пользователей сохранность данных имеет первостепенное значение. Высказанные соображения приводят к созданию гораздо более тщательно разработанных и автоматизированных систем, чем это требуется обычно для пакетной обработки.

Третий тип — реальное время (Real Time). Слова «реальное время» относятся к той ситуации, когда требуется выполнять действия за интервал, который прямо или косвенно связан со временем реакции человека. Может показаться, что определение системы реального времени почти совпадает с определением системы мультидоступа. В действительности имеются в виду два четко отличающихся понятия. В системе мультидоступа обрабатывается обычный набор работ пользователей, и каждый пользователь может пускать разные программы. Слово «мультидоступ» подчеркивает лишь тот факт, что всем пользователям по очереди предоставляются машинные ресурсы. В реальном времени, наоборот, большинство пользователей работают с одной и той же программой или с небольшим набором программ, и систему, следовательно, можно оптимизировать так, чтобы выполнить работу возможно большего количества пользователей. Обычно пользователем системы реального времени является клерк, а системы мультидоступа — программист. В системе реального времени может быть до 1000 терминалов, а в системе

¹⁾ Дисплей — устройство с электроннолучевой трубкой для приема (или передачи) изображения из (на) машины(у). Существует большой диапазон таких устройств от самых дешевых, которые часто имеют клавиатуру и используются как быстрые телетайпы, до очень дорогих с богатыми изобразительными средствами. — *Прим. перев.*

мультидоступа — едва ли больше ста. Обычно приводится пример использования системы реального времени для резервирования мест на самолеты. Проблемы реального времени рассматриваются в предпоследней главе. Здесь же отметим только, что на данном уровне системного программирования реализация черт, необходимых как системе мультидоступа, так и системе реального времени, чрезвычайно сложна. Это объясняется очень высокими требованиями, предъявляемыми к автоматизации управления.

Теперь вернемся к вопросу, поставленному в заголовке этого раздела: «Зачем нужна операционная система?» Начав издалека, от первого поколения машин, легко было забыть те логические шаги, которые привели нас к современному состоянию дела.

1. Сравнение скорости машины со скоростью человека диктует нам, чтобы машина принимала решения, которые раньше принимал человек.
2. Современная система слишком сложна, чтобы ею мог управлять оператор.
3. Контроль за пользователем должен быть на уровне программ пользователя.
4. Основной набор средств и возможностей должен обеспечиваться системой.

Этот перечень далеко не полный, другие соображения будут приведены в следующем разделе.

1.3. Кому нужна операционная система и для чего?

Здесь перечислены лица, заинтересованные в операционных системах, и требования, предъявляемые этими лицами (услуги, которые они хотели бы иметь от операционных систем). Ранее говорилось, что не все эти требования должны удовлетворяться в каждой системе. Можно предположить, что на самом деле многие из перечисленных операций экономичнее выполнять вручную.

1.3.1. Директор вычислительного центра

Профессиональные интересы: возможность управлять машиной или всей системой; эффективность в работе; рентабельность системы.

1. *Регистрация всех действий, связанных с использованием машины*

Все операции, выполняющиеся в системе, должны документально фиксироваться, включая использование фай-

лов, действия оператора, пуск и завершение работ пользователей.

2. *Определение стоимости работы*

Кроме регистрации действий, нужен метод определения платы, которая взимается с пользователя за выполнение его работы. Плата зависит не только от программы пользователя, но и от степени использования системы в отношении файлов и других системных возможностей и услуг.

3. *Управление работами*

Одни работы очень срочные, другие должны быть выполнены к вполне определенному сроку, третьи можно использовать как фоновые задачи, решая их в мультипрограммном режиме. Директор заинтересован в том, чтобы система обеспечивала выполнение каждой работы в положенное время.

4. *Эффективность операционной системы*

Накладные расходы, связанные с использованием операционной системы, должны окупаться ее эффективностью в работе.

5. *Сохранность работ пользователя*

Каждая работа должна быть защищена от каких-либо посягательств со стороны. Работы, одновременно находящиеся в оперативной памяти, не должны портить информацию друг другу.

6. *Защита файлов от некорректного использования и случайных изменений*

Пользователь может оставить на хранение вычислительной системе свои файлы, расположенные на лентах, дисках и т. д. При этом должен быть исключен доступ к чужой информации.

7. *Быстрое восстановление работы после сбоев или отказов машины*

После неизбежных неисправностей в аппаратуре требуется быстрое, а иногда немедленное возобновление работы. Система файлов не должна противоречить возможности продолжить работу с минимальными потерями.

1.3.2. Оператор на машине

Профессиональные интересы: управление машиной.

1. *Требование действий*

Оператору нужно сообщать всякий раз, когда операционная система требует каких-либо действий от него, например загрузить файл или дать требуемый ответ на телетайпе. Нужна стандартная форма для представления таких сообщений.

2. *Состояние аппаратуры*

Оператору нужно сообщать, какие устройства заняты, какие требуют загрузки, сообщать о ситуациях, когда устройства требуют его внимания (например, кончилась бумага в печатающем устройстве и т. д.)

3. *Состояние работ*

Оператор должен знать о каждой выполняющейся в данный момент работе, о каждой запускаемой работе и устройствах, которые она будет использовать, а также о всех выполненных работах. Оператор должен иметь возможность временно приостановить работу или вовсе ее снять, если обнаружена ее некорректность.

4. *Загрузка устройств*

Оператор должен иметь возможность устанавливать сменные тома (магнитные ленты или пакеты дисков) на любое свободное устройство; система должна уметь распознавать каждый такой том по его имени. Это называется автоматическим распознаванием томов (Automatic Volume Recognition). *а*

1.3.3. Пользователь системы

Профессиональные интересы: представление работы и ее исполнение.

1. *Управление работой*

У пользователя двойственные интересы. С одной стороны, он хочет, чтобы описание работы для системы было простым, с другой, чтобы существовало большое число приемов, с помощью которых можно было бы точно выразить его требования относительно крайнего срока исполнения или важности его работы, способа использования файлов и элементов аппаратуры (память, каналы и т. д.), а также относительно продолжительности его работы.

2. Представление данных

Обычно некоторые данные для своей работы пользователь пробивает на перфокартах или перфоленге. Эти данные вместе с описанием работы на языке управления работами образуют внешний ввод в систему. Операционная система должна уметь обрабатывать данные и управляющую информацию в едином потоке, так как желательно накапливать впрок полные комплекты работ для дальнейшего планирования работы операционной системы.

3. Файлы

Пользователь должен иметь возможность обращаться к своим файлам, хранящимся на магнитных лентах или дисках, по именам. Предполагается, что система сама ведет «информационное хозяйство» пользователя, гарантирует сохранность предыдущих вариантов и использование требуемого варианта. Сохранность и неприкосновенность файлов имеют крайне важное значение.

4. Получение результатов

Где бы ни запоминались результаты — на дисках или на ленте, для их последующей печати должен существовать внутренний алгоритм, гарантирующий пользователю наиболее быстрое получение результатов.

5. Требования к мультидоступу

Эти требования довольно обширны. Пользователь «разговаривает» с системой, и его диалог с машиной должен быть легким и понятным. Система по самой своей природе должна быть разговорной, т. е. она должна уметь спрашивать и отвечать на вопросы. Отвечать нужно быстро. Под руками у пользователя должно быть большое количество вспомогательных программ, готовых в любой момент помочь ему в решении его задачи. Пользователь не ощущает и не должен ощущать той работы, которую проделывает система для поддержания режима мультидоступа.

6. Сбой аппаратуры

Пользователь должен быть уверен, что в случае возможного сбоя от него не потребуются никаких действий. Система должна сама, не причиняя каких-либо беспокойств пользователю, уметь возобновить работу.

1.3.4. Программист

Профессиональные интересы: разработка и отладка программ.

1. Использование отладочных программ

Во многих вычислительных центрах на отладку программ тратится большой процент машинного времени. Поэтому методы и средства отладки чрезвычайно важны. Они состоят из отладочных программ, компиляторов, текстовых редакторов, компоновщиков, программ прокрутки-транслировки и т. д.

2. Интерфейс между операционной системой и программой

Программист обращается к услугам операционной системы, когда нужно работать с записями в файлах, выполнять операции ввода-вывода и макрокоманды, запросить или освободить место в оперативной или вспомогательной памяти. Операционная система дает возможность программе передавать сообщения оператору.

3. Библиотеки часто употребляемых программ

Должен существовать набор подпрограмм и простой способ включения каждой из них в программу пользователя.

1.3.5. Инженер, ведающий операционной системой

Профессиональные интересы: поддержание в рабочем состоянии и модификация системы.

1. Легкость введения модификаций в программное обеспечение

Должны существовать документация и блок-схемы, помогающие вносить исправления и изменения. Программное обеспечение должно иметь модульную структуру.

2. Генерирование системы

При создании нового варианта программного обеспечения необходима генерация операционной системы с ленты или дисков. В процессе генерации для данной установки создается конкретная операционная система, в качестве параметров которой фигурируют размер оперативной памяти, заданный набор периферийных устройств и требуемые функции.

1.3.6. Инженер по эксплуатации машины

Профессиональные интересы: возможно более раннее обнаружение и устранение неисправностей аппаратуры.

1. Сообщения о неисправностях

Все отказы аппаратуры должны регистрироваться системой, даже единичные сбои, аналогичные тем, которые возникают при работе ленты и самоустраняются при повторном обращении к ней. Инженер-эксплуатационник должен иметь возможность получать полную информацию обо всех неисправностях.

2. Диагностика неисправностей

Если система обнаружила ошибку и еще не утратила контроля над машиной, она должна попытаться провести диагностику неисправности, чтобы локализовать ее для быстрого исправления.

1.3.7. Проектировщик аппаратуры

Профессиональные интересы: интерфейс между аппаратурой и операционной системой.

1. Интерфейс с системой

От того, как спроектирована аппаратная часть машины, зависит логика операционной системы и, в частности, ее эффективность. Операционная система вынуждена иметь дело с «голым» оборудованием и должна обеспечивать пользователя некоторым интерфейсом. От того, как выполняются операции ввода-вывода и какие допускаются прерывания, во многом зависит вся работа операционной системы.

2. Программная интерпретация команд — экстракоды

Иногда одна операционная система предназначается для ряда машин от дешевых до самых дорогих. В таком случае при проектировании аппаратуры для менее мощных машин возникает желание интерпретировать некоторые операции с помощью программ. Программная интерпретация может быть экономически оправдана и в том случае, если какая-то возможность (например, плавающая запятая) редко используется. Для большинства вычислительных машин коды операций, не реализованных аппаратно, обозначаются и обрабатываются программами операционной системы. В ICL 1900 такие операции называются «экстракодами» (Extracode).

1.4. Сколько стоят услуги операционной системы?

Вообще, не существует общепринятых методов, позволяющих оценить стоимость услуг операционной системы. Было время, когда оборудование оценивалось в терминах «смеси

команд»¹⁾, но для операционных систем аналогичных методов не нашли. Пользователь системы не знает, за какое время будет выполнена его работа. Это время зависит от того, как операционная система планирует исполнение работ, какие возможности системы используются, а также от специфики самой программы пользователя. Для оценок «смесь команд» нужно заменить «смесью работ», но не найдены те коэффициенты, по которым эту смесь можно было бы вычислить. Смесь работ относится только к пакетной обработке и вряд ли применима для мультидоступа. Однако, применяя некоторые существенные критерии, можно выявить положительные и отрицательные факторы, влияющие на оценку услуг операционной системы.

Существуют два фактора, которые могут быть учтены и отнесены к минусам системы. Первый — это размер той части аппаратуры, которая постоянно работает на операционную систему. В пользовании системы может быть значительная часть оперативной памяти — от 1000 слов для простейших систем до 25 000 слов для самых развитых. В добавление к этому требуется определенный объем вспомогательной памяти, например на магнитных лентах, дисках и барабанах. Второй фактор — время, которое система тратит на свои действия. Эти два фактора можно измерять, хотя второй фактор — время — создатели операционной системы могут скрыть и скорее отнесут его к работам пользователя, чем вычислят отдельно. Кроме этих факторов, поддающихся учету, может быть еще один отрицательный фактор. Операционная система — единственный интерфейс, связывающий пользователя с машиной. Пользователь не может его обойти и целиком находится в той обстановке, которую ему создала операционная система. Пользователю могут быть неудобны ее условия и ее возможности. Она может быть слишком общей для его специфических применений, и тем не менее он будет вынужден мириться с ее неэффективностью. Другой неприятностью, которой часто досаждают операционная система, особенно первым пользователям, являются ее многочисленные ошибки, и чем сложнее система, тем их больше на ранней стадии эксплуатации.

К плюсам системы следует отнести возможности и услуги, которые она предоставляет. Можно легко упустить важность этого и забыть, что с аппаратным интерфейсом работать труднее, чем с программным. Операционная система должна обес-

¹⁾ С термином «смесь команд» обычно связывается сумма $c = \sum k_i t_i$, где k_i — коэффициент, определяющий относительную встречаемость i -й команды, а t_i — время ее выполнения. Очевидно, c представляет собой среднее время выполнения команды и определяет быстродействие процессора. — *Прим. перев.*

печивать значительное увеличение общей пропускной способности вычислительной машины, т. е. число работ, выполняемых за единицу времени, и, следовательно, увеличивать прибыль, получаемую с машины.

Однако важнее всех соображений о плюсах и минусах системы остается вопрос: может ли вообще машина работать без операционной системы? В любой комплексной работе в конце концов достигается точка, в которой автоматизация становится абсолютно необходимой. Человек не в состоянии за приемлемое время принять столько решений, сколько их принимает операционная система. Реакция операционной системы в 10 000 раз быстрее, чем реакция оператора. И кроме того, какая современная вычислительная установка может обходиться без защиты файлов?

Теперь пользователь не покупает одно оборудование. Он покупает «комплект», в который входит аппаратура, операционная система, пакеты прикладных программ и другое программное обеспечение. Операционная система является основным элементом комплекта.

1.5. Какую операционную систему можно считать хорошей?

Эффективность операционной системы в большой степени зависит от условий, в которых она применяется. То, что хорошо для одного пользователя (имеются в виду возможности и правильная стратегия операционной системы), может оказаться далеко не совершенным для другого пользователя с другими задачами. Существует много универсальных систем, призванных удовлетворить своими свойствами всех пользователей, но каждая новая функция несет в себе накладные расходы по затратам памяти и времени, ограничивающие эффективность операционной системы в целом.

Эффективность операционной системы, таким образом, представляет собой функцию перечисленных ниже параметров.

1. Коэффициент полезного действия

Меру накладных расходов операционной системы можно выразить в терминах использования памяти, а также с помощью соотношения $1 - T_c/T$, где T_c — время, затрачиваемое операционной системой на свои личные нужды, а T — общее время использования системы.

2. Функции и возможности

Система должна уметь выполнять все возложенные на нее функции. Она не должна ограничивать пользователя

в использовании естественных и логически возможных операций.

3. *Легкость в обращении*

Система должна быть простой в использовании и обозримой, она не должна требовать больших усилий в изучении ее свойств и при подготовке данных.

Качество системы зависит от того, насколько хорошо сбалансированы ее возможности и эффективность. При достижении этого баланса неизбежны различные компромиссы. В системах, поставляемых изготовителями на широкий рынок для общих применений, акцент делается на возможностях, и система редко бывает оптимальна для каких-либо специальных работ. В специализированных системах можно достичь значительно лучшего баланса, если акцентировать внимание на эффективности. Для изготовителя возникают дополнительные рассматривания, касающиеся совместимости внутри ряда машин, а также совместимости с предыдущими или существующими в настоящее время системами. Любой пользователь хотел бы иметь операционную систему, скроенную в соответствии с его требованиями и нуждами, но стоимость системы так велика, что очень редко она создается по индивидуальному заказу.

РОЛЬ СУПЕРВИЗОРНЫХ ПРОГРАММ

Брайен Миллис

От общих вопросов мы перейдем теперь к более частным. Для того чтобы можно было достаточно детально представить себе одно семейство операционных систем, мы остановимся на операционных системах машин серии ICL 1900. Брайен Миллис начинает рассмотрение с супервизора, который является сердцем всех операционных систем серии 1900 и который выполняет функции скромной операционной системы на малых машинах серии. Он продолжает более детальное рассмотрение некоторых исторических предпосылок, затронутых Мартином Уорвиком, и вводит важные понятия, относящиеся к параллельным процессам: — мультипрограммирование, автономный ввод-вывод (off-lining), подзадачи, мультидоступ и прерывания по сигналам от устройств ввода-вывода.

2.1. Введение

Эта глава посвящена супервизорным программам. Как было объяснено в 1.2, супервизор — это резидентная управляющая программа вычислительной системы, его функции — управлять выполнением всех других программ, использующих эту систему, и обеспечивать разнообразное обслуживание, за которым эти программы могут к нему обратиться. В последнее десятилетие контуры супервизорных программ стали более четкими. В этой главе мы рассмотрим некоторые предпосылки их развития, а также обсудим, что такое супервизор по существу.

2.2. Программное обеспечение

О разработке программного обеспечения путем создания подпрограмм упоминалось в 1.2. Для того чтобы действительно понять, какое именно место занимают супервизоры по отношению к оборудованию, с одной стороны, и к более общему программному обеспечению, с другой, необходимо рассмотреть их более детально. Универсальная электронная вычислительная машина способна выполнять сравнительно небольшой набор элементарных операций. Могущество машины заключается в ее способности крайне быстро выполнять огромное количество таких операций над большим числом вариантов данных и в ее умении изменять последовательность выполняемых операций в зависимости от проверок, проводимых над данными. Хотя написание последовательности строк из десятка элементарных операций не требует большого интеллекта, число таких строк, требующихся для выполнения сколько-нибудь существенной работы, настолько велико, что усилия, затрачиваемые

при написании всей последовательности, качественно изменяются. Более того, ограниченные размеры имеющейся оперативной памяти усугубляют положение из-за необходимости поддерживать небольшим общее число строк последовательности — *программы* и, следовательно, как можно чаще повторно использовать уже имеющиеся элементы — куски программы. Составление программы, которая эффективно выполняет задание, и приведение ее в рабочее состояние становится довольно трудным интеллектуальным занятием, поэтому уже на раннем этапе истории программирования разработанная и отлаженная программа ценилась высоко. И если в программе нужна была некоторая функция, которая уже была запрограммирована в ранее встречавшейся программе, то преимущество, получаемое от возможности скопировать ее команды, было так велико, что стало выгодным создавать библиотеки программных секций — *подпрограмм*, которые реализовывали различные, часто встречающиеся функции. (Библиотека подпрограмм для машины EDSAC 1 Кембриджского университета положила начало этому методу.)

Далее, скоро было обнаружено, что, помимо обеспечения большим набором заготовок программ, выполняющих важные функции, работа программиста могла быть облегчена, если ему дать возможность использовать не простые машинные операции, а более сложные. Написанное должно затем автоматически транслироваться в последовательность большего числа элементарных операций, дающих тот же эффект. Это положило начало языкам высокого уровня и компиляторам.

После того как эти факты были установлены, разработчик машины получил реальную возможность делать элементарные операции машины менее удобными для программиста, если это помогало ему сделать машину более дешевой и быстрой. Неудобные стороны этих операций могли бы быть скрыты от программиста, который пишет программу на языке высокого уровня или всегда использует подпрограммы в тех случаях, где неудобства касаются одного класса операций (например, чтения или записи на внешний носитель). Конечно, соответствующие компиляторы и подпрограммы приходится писать на менее удобном языке, но, как правило, это необходимо сделать один раз, и, следовательно, можно себе позволить иметь высококвалифицированных людей для их написания.

При таком подходе компиляторы и подпрограммы становятся неотъемлемой частью системы, так как для обыкновенного пользователя они так же необходимы, как и электроника. Для обозначения всех этих программ и было введено новое слово — *software* (мягкие товары) — *программное обеспечение*. Программное обеспечение — это совокупность программ,

которые должны быть добавлены к физической машине — к *оборудованию* — hardware (жестким товарам), — чтобы сделать систему в целом пригодной для обычного пользователя.

Как и в большинстве случаев, здесь имеются недостатки. Компиляция программы или создание ее из универсальных подпрограмм почти неизбежно не дают такого эффективного и компактного результата, какой давали наилучшие программы, написанные с учетом специфики элементарных операций. Будучи универсальными, такие подпрограммы делают обычно не только то, что вам нужно, а гораздо больше. Это открытие заставляет многих программистов на каком-то этапе их программистской деятельности выступать против универсального программного обеспечения, часто вплоть до полного отрицания его потенциальных преимуществ.

Конструирование сложных систем из стандартных блоков является общепринятым инженерным приемом во всех технических областях, включая и вычислительные машины. Здесь, однако, это получило широкое распространение скорее всего из-за уменьшения стоимости, достигаемой за счет большого количества продукции. Стоимость самого физического носителя программы (скажем, отперфорированной колоды карт) сравнительно невелика, и поэтому даже те дополнительные расходы памяти и времени при каждом пропуске программы, которые возникают в результате компоновки программы из стандартных блоков, полностью компенсируются экономией времени программиста и средств на разработку программы. Конечно, использование элементарных операций при создании максимально хороших программ для некоторых критических частей работы часто оправданно. Но и это становится редкостью, поскольку работы делаются все более сложными, а цена вычислительной машины, имеющей любую заданную скорость, падает, чего нельзя сказать о стоимости программирования. Мы еще не достигли стадии, на которой для большинства машин программирование в операциях, близких к машинным, запрещено. Но сейчас уже общепризнано, что обычные программы в состоянии управлять периферийными устройствами только через подпрограммы супервизора.

Поскольку обыкновенный пользователь видит машину только через программное обеспечение, то последнее является для него частью машины. Если же функции, которые ему требуются, выполняются с нужной скоростью, то программист перестает интересоваться, *что* выполняется оборудованием, а *что* — программным обеспечением. Они сливаются в единое целое. Это обстоятельство наиболее явно проявляется на микропрограммных машинах, где операции, употребляемые программистом, интерпретируются и выполняются программами

нижнего уровня — *микропрограммами*, которые в свою очередь используют очень примитивный набор приказов, реализованный в электронной логике. В итоге в известном смысле все программы оказываются построенными из подпрограмм. На практике микропрограмма обычно «зашивается» в специальную быструю память, из которой можно только читать, и обычные программы совершенно лишены возможности непосредственно использовать элементарные машинные операции. Программист совсем и не рассматривает их как машинные операции, а считает интерпретируемые операции (которые как правило, находятся на уровне обычных машинных операций) операциями этой машины. Микропрограммирование, следовательно, — это пример «мягкого» программного обеспечения, располагающегося на самом нижнем уровне, которое в силу своего крайнего положения слилось с оборудованием и стало «жестким». Многие подпрограммы супервизора аналогичны микропрограммам, но они находятся на несколько более высоком уровне и все еще остаются «мягкими» в том смысле, что они расположены в памяти, в которую можно записывать. Итак, супервизор представляет собой вид программного обеспечения, находящегося на стыке с оборудованием, и функции, выполняемые супервизором, могут рассматриваться обычным программистом как часть операций машины.

2.3. Параллельная обработка

Основная цель, ради которой были созданы супервизорные программы и операционные системы, — это лучшее использование вычислительной системы, т. е. выполнение большего числа работ в заданное время. Специализированные системы достигают высокой производительности по отношению к конкретному типу работ за счет учета специфики этих работ. Примером могут служить специализированные системы для выполнения небольших учебных программ, написанных студентами на языке FORTRAN. В этой главе мы сосредоточим внимание на универсальных системах, которые, функционируя в вычислительной машине, могут обслуживать программы всех типов. Эти системы увеличивают производительность машины, во-первых, за счет того, что позволяют пользователю предвидеть события и заранее выдавать системе инструкции о том, как действовать, не затрачивая времени на ожидание ответа человека в момент, когда событие происходит; и, во-вторых, за счет того, что дают возможность выполнять два или более процесса одновременно. Первое направление почти целиком принадлежит области операционных систем, а не супервизоров, и поэтому здесь мы рассмотрим лишь второе. Основные

процессы, которые имеет смысл выполнять параллельно, — это вычисления в центральном процессоре и работа различных «периферийных» устройств.

Исходные данные для работы вычислительной машины должны быть введены, а результаты выведены. Файлы, которые понадобятся только для последующего использования машиной (т. е. представляют собой промежуточные результаты), должны быть выведены на некоторый носитель и введены вновь, когда они потребуются, так как неэкономично хранить их в быстрой памяти все время. Самый естественный способ — это читать запись непосредственно перед тем, как она нужна, и выводить результат целиком, как только он получен. К сожалению, конечно, суммарное время, затраченное на выполнение любой работы в такой последовательности, складывается из времен, затраченных на чтение, обработку и вывод. Однако относительные диспропорции между временем чтения с внешнего носителя и временем выполнения одной внутренней операции таковы, что суммарное время обработки одной записи (для большинства работ) сравнимо с временем ввода или вывода. Следовательно, оборудование, предназначенное для выполнения любой из этих операций, будет простаивать большую часть времени. Более того, не так уж много оборудования нужно добавить, чтобы стало возможным выполнение всех трех операций одновременно. Запись пересылается от центрального процессора к периферийному устройству и обратно либо через дополнительный буфер быстрой памяти, способный вместить блок данных для периферийного устройства, либо более распространенным в настоящее время способом «захватывания» цикла главной памяти в момент, когда следующее слово или символ готовы к пересылке. Дальнейшие детали этой механики для нас несущественны — важно лишь, что запись вводится в память (становится доступной) или выводится, не оказывая заметного влияния на скорость работы центрального процессора.

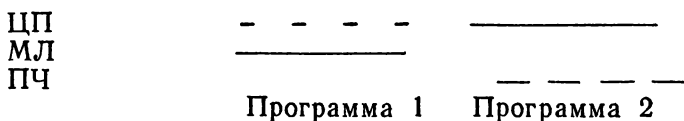
Очевиден один простой способ извлечь выгоду из этого — вводить следующую запись (или блок записей, если в целях экономии они сгруппированы в блоки), пока текущая запись обрабатывается, а предыдущая выводится. Но это даст идеальный результат только, если на обработку затрачивается такое же время, как и на ввод и вывод каждого блока, что невероятно. В действительности оказывается, что в некотором классе программ время на обработку всегда меньше времени на ввод или вывод записей; в другом — оно всегда больше. Существует третий класс программ, которые фактически переходят из первого класса во второй (или обратно) в процессе работы, читая пакет данных, затем ведя длинную обработку и,

наконец, печатая пакет результатов. Третий класс может быть сведен к одному из двух других либо путем деления программы на части, либо путем выполнения ввода-вывода автономно (off-lining), т. е. вне связи с основными вычислениями, что будет описано ниже. Дальнейшую экономию мы можем получить при помощи «мультипрограммирования».

2.4. Мультипрограммирование

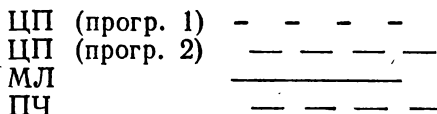
Под мультипрограммированием, как было объяснено в 1.2, мы понимаем выполнение «сразу» двух или большего числа программ. Это не означает, что обе программы выполняются в данную микросекунду, но центральный процессор переключается с одной на другую и обратно, выполняя все работы вместе, а не доводя одну программу до конца. Цель мультипрограммирования — держать процессор загруженным, не давая ему простаивать в ожидании завершения пересылки информации с периферийного устройства для одной программы.

Рассмотрим два крайних случая наших программ, упомянутых в предыдущем разделе: одну программу, обновляющую данные в файле, с небольшими вычислениями, но множеством чтений и записей на магнитную ленту, и другую, скажем выполняющую вычисления и только изредка печатающую результаты. Если мы выполняем одну программу вслед за другой, даже с заблаговременным чтением, то картина использования машины выглядит следующим образом:



где ЦП — центральный процессор, МЛ — магнитная лента, ПЧ — печать.

Теперь предположим, что мы смогли бы обеспечить выполнение вычислений для второй программы в промежутке между вычислениями для первой:



Мы выполняем первую программу с такой же скоростью, как и раньше, и вторую, скажем, с 9/10 ее нормальной скорости. Если для выполнения каждой работы требовался один час времени, то при совмещении общее время для выполнения двух работ было бы равно одному часу и шести минутам вме-

сто двух часов. Мы можем делать это, держа обе программы в памяти и выполняя первую до тех пор, пока есть, что считать. Когда же она останавливается, в ожидании завершения ленточной операции, мы переключаемся на вторую, а когда ожидавшаяся операция закончена, мы снова переключаемся на первую и т. д.

Конечно, мы взяли для совместного выполнения две идеальные программы. На практике же никакая установка не имеет в точности нужного баланса между программами, лимитируемыми процессором (счетными программами), и программами, лимитируемыми периферийными устройствами (пересылочными программами). Однако если мы выполняем вместе две пересылочные программы, чередуя обработку их записей, то при условии, что они не пользуются одним и тем же периферийным устройством, мы также достигаем экономии. Примером могут служить две программы, из которых одна копирует карты на магнитную ленту, а другая читает другую магнитную ленту и печатает ее содержимое. Заметим, что в этом примере мы можем пользоваться ленточной системой, которая не допускает одновременное чтение и запись. При условии, что время чтения с карт и печати строки много больше (как это обычно и бывает), чем время чтения или записи на ленту, мы будем, вдобавок к использованию центрального процессора для счета сразу по двум программам, выполнять сразу две работы и в нашей ленточной системе. Мы могли бы добавить еще больше пересылочных программ. Если и после этого центральный процессор все еще терял время, а мы имели бы счетную программу, то с таким же успехом мы могли бы добавить и ее и пропускать, когда ни одна из пересылочных программ не выполняется. В результате в машине находилось бы четыре, пять и более программ, выполняемых в одно и то же время. Конечно, эти действия целесообразны, только если наш процессор достаточно быстр с точки зрения обработки записей, но в настоящее время это условие практически выполняется даже на малых машинах. Заметим, между прочим, что для первого примера более выгодный способ выполнить ту же работу примерно в половинное время состоит в том, чтобы приобрести в два раза более медленный процессор и по-прежнему выполнять в два раза больше пересылочных программ!

Как дорого обходится реализация мультипрограммирования? Ну, во-первых, для хранения дополнительной программы нам нужна дополнительная внутренняя память. В действительности это может приводить, а может и не приводить к увеличению стоимости, так как редко все программы используют всю память, емкость которой определяется наибольшим программным сегментом и округляется до размеров стандартных

конструктивных блоков. Но даже если это приводит к удорожанию, то, скажем, удвоение производительности машины обычно оправдало бы покупку небольшого дополнительного куба памяти. Практически для того, чтобы сделать мультипрограммную работу эффективной, нужно добавить не только память, но и внешние устройства, поскольку почти все программы требуют нескольких вводных и нескольких выводных устройств. Правда, как будет объяснено ниже, в значительной степени этого добавления можно избежать, если использовать off-lining (автономный ввод-вывод). Но тем не менее даже и без этого, при помощи относительно небольшого увеличения аппаратуры, может быть достигнуто увеличение пропускной способности машины в два или более раз.

Мультипрограммирование описанного здесь типа обычно выполняется управляющей программой, которая определяет, когда и какую задачу ввести в счет. Эта программа является одной из частей супервизора.

p

2.5. Автономный ввод-вывод

Автономный ввод-вывод (off-lining) по-настоящему является предметом одной из следующих глав. Мы коснемся этого вопроса здесь лишь с той целью, чтобы показать его существенную связь с мультипрограммированием.

В предыдущем разделе мы упомянули о двух проблемах, в которых в сущности ставилась задача сделать более эффективным использование наших вводных и выводных периферийных устройств (т. е. устройства ввода перфокарт и печатающего устройства), а не центрального процессора. Пусть у нас есть программа, требующая серии вводов, за которой позже последует серия выводов; кроме того, имеются программы, нуждающиеся только во вводе или эпизодически только в печати, но которые мы не можем выполнять параллельно с другими программами, обращающимися к тем же устройствам. Если система имеет диск (или барабан), то мы решаем проблему совмещения, читая вводимый на диск материал заранее, вызывая его с диска, когда он потребуется в процессе счета, помещая результаты на диск, и затем позже, печатая пакет результатов. Таким образом, мы вводим карты и печатаем результаты в режиме off-line, т. е. автономно, вне основного процесса — работы главной программы. Конечно, дисковые пересылки должны выполняться блоками и занимать незначительное время по сравнению с «медленными» периферийными пересылками. Таким способом освобождаются два устройства во время основного вычисления, поэтому они могут быть использованы для других программ. С другой стороны,

таким путем мы имеем возможность выполнить все наши чтения с карт и печать результатов, если постоянно держим в памяти две программы, одну — для чтения карточного файла на диск, другую — для вывода файлов с диска на печать. Тогда в идеальном случае мы можем сразу же использовать оба наши устройства, одно — для чтения данных с карт для следующей программы, другое — для печати результатов предыдущей программы параллельно с выполнением текущей программы, данные для которой уже находятся на диске и которая помещает туда же свои результаты. Эти программы образуют удачный набор для мультипрограммирования главным образом в силу того, что переписывающие программы достаточно малы и занимают небольшую часть внутренней памяти. Более глубокое совмещение выполнения основных программ может быть достигнуто за счет заполнения времени ожидания диска или других устройств для хранения файлов, таких, как магнитная лента.

2.6. Подзадачи или подпроцессы

Особой формой мультипрограммирования является одновременное выполнение частей одной программы. Исходная программа определяет внутри себя элементы, которые можно выполнять независимо и тем самым получить выгоду от мультипрограммирования в пределах одной программы. С этим методом связано несколько названий, среди которых «подзадачи» и «подпроцессы».

Использование этого метода программирования может быть проиллюстрировано двумя примерами; вот первый из них. Рассмотрим программу, которая производит серию вводов, затем считает, а затем выполняет серию выводов. Альтернативой режима off-line — реализации ввода и вывода с помощью отдельных программ — является выполнение того же внутри одной работающей программы. Этот метод оказывается наиболее удобным, если общий объем вывода большой, а количество результатов, полученных после очередного интервала вычислений, достаточно мало, что позволяет разместить их в оперативной памяти (или, может быть, результаты получаются достаточно непрерывно в процессе вычислений, но с непостоянной скоростью). Идея метода состоит в том, что подлежащий выводу материал основной программы ставится в очередь выводимых данных (предположим, в памяти) и элемент основной программы, управляющий выводом, печатает данные, начиная с головы этой очереди. Другой элемент, управляющий вводом, действует аналогично, наполняя очередь вводимых данных. В итоге три элемента (один из них —

основная программа) выполняются системой в мультипрограммном режиме в точности так же, как если бы они были независимыми программами. Конечно, эти элементы должны содержать внутри себя блокировки, чтобы разбираться в таких ситуациях, когда очереди становятся пустыми или переполненными.

Второй иллюстрацией может служить прием данных от устройства, которое по своей природе дает неравномерную скорость пересылок, например получение сообщений от ряда телетайпов, за которыми работают люди. В этом случае, управляя приемом информации с помощью некоторого элемента программы, мы можем не только сгладить поток сообщений для основной программы и организовать очереди для каждого телетайпа, но, более того, этот элемент может давать немедленные ответы, требующиеся телетайпам, как только пересылка закончена. (Этот метод станет более ясным, после того как будут рассмотрены прерывания от устройств ввода-вывода.)

2.7. Мультидоступ

Другой формой параллельной работы является упомянутый в 1.2 мультидоступ. Это служит главной темой одной из последующих глав, и поэтому здесь это понятие только вводится. Мультидоступ своим появлением обязан желанию иметь вычислительную машину «под рукой», т. е. готовой к немедленному использованию человеком. Для многих целей, включающих и отладку программы, отдельный пользователь хотел бы запустить свою программу, дать ей проработать в течение некоторого времени, затем посидеть и подумать, ввести с телетайпа дополнительные данные или исправления, пустить программу несколько дальше и т. д. Но привязка машины таким способом к одному пользователю является расточительством (хотя оно и делается довольно часто) даже на малых машинах и недопустимо дорого на больших, поскольку человеческие задержки заставят процессор простаивать большую часть времени. Решение этой проблемы состоит в мультипрограммном пропуске программ большого числа пользователей. Мы даем каждому пользователю свой «пульт» (скажем, телетайп) и выполняем в режиме off-line любые вводы и выводы, не относящиеся к этим пультам. Но теперь мы обнаруживаем, что у нас так много программ, что им недостаточно внутренней памяти. Решение этой проблемы — не держать все программы в памяти одновременно, а выбрасывать их и подкачивать, используя диски или барабаны. На этом пути также достигается мультипрограммный эффект, правда на более грубой временной сетке, чем та, которая реализуется при переключении.

чении программ, находящихся в оперативной памяти. Но на практике этого бывает достаточно, чтобы обеспечить пользователю достаточно быстрое обслуживание.

Этот вид параллельной работы называется мультидоступом, и он отличается от мультипрограммирования, рассмотренного ранее, техникой подкачки программ «в» и «из» памяти, а также своими функциями; в отличие от мультипрограммирования перед мультидоступом не ставится цель увеличить пропускную способность системы, однако эта цель не ставится лишь до тех пор, пока мы не рассматриваем всю систему в целом, включая в нее и пользователя-человека.

2.8. Прерывания от периферийных устройств

До сих пор мы рассматривали выгоды от параллельных операций, не обсуждая способов управления ими. Мы хотим, чтобы периферийные устройства работали параллельно с вычислениями, и хотим уметь переключаться с текущей программы, когда она останавливается. Одной из причин переключения может быть желание продолжить ту программу, которая ожидала пересылки, для того чтобы эта программа смогла обработать следующую запись и, таким образом, поддерживать внешнее устройство в активном состоянии. Но даже когда выполняется одна программа, может оказаться, что некоторое действие должно быть выполнено срочно, при окончании пересылки. Например, для внешнего устройства, которое мы не можем мгновенно остановить, у нас может возникнуть необходимость немедленно начать следующую пересылку, которая была приготовлена заранее. Аналогично для устройств, допускающих цепочки приказов, нужно успеть быстро выдать очередной приказ.

Один способ выполнения такого сорта переключений состоит в том, что любая текущая (выполняемая) программа периодически, в моменты, удобные для нее самой, входит в стандартную программу, которая проверяет, произошли ли некоторые события. Этот способ может давать удовлетворительное решение, если программа тесно связана с событиями, как это бывает при работе в режиме реального времени, обсуждаемого в гл. 9. Но в обычной ситуации это в лучшем случае обременительно, а в худшем — совершенно неприемлемо. И вообще, при мультипрограммной работе, как одна программа может определить, насколько часто она должна просматривать события другой программы? Значительно более удобный способ состоит в следующем. Устройство ввода-вывода посылает процессору аппаратный сигнал об окончании пересылки, и по получении этого сигнала аппаратно осуществ-

ляется прерывание текущей программы, запоминание всех регистров, которые необходимо сохранить, и переключение на специальную программу, которая определяет причину прерывания и решает, что делать дальше. Как выбор нужных действий, так и сами действия достаточно сложны и зависят от типа устройства и операции. Поэтому аппаратная реализация этих действий была бы дорогой, потребовала бы введения в центральный процессор оборудования, зависящего от устройства ввода-вывода, и была бы крайне негибкой. Следовательно, несмотря на то что встроенные аппаратные возможности в больших машинах существенно упрощают дело, тем не менее, как уже говорилось, существует специальная программа, предназначенная для выполнения рассматриваемых функций. Эта программа является ядром супервизора.

СВОЙСТВА СУПЕРВИЗОРА

Брайен Миллис

Во второй главе Брайен Миллис ввел понятие супервизора и рассмотрел его роль в мультипрограммировании. Здесь он продолжает изложение, рассматривая приоритеты в мультипрограммировании, распределение оперативной памяти, абонирование программами пользователя внешних устройств и связь с оператором. В заключение после некоторого обсуждения больших операционных систем он подводит итоги, перечисляя основные функции супервизора.

3.1. Развитие супервизоров

Мы показали, что для обработки сигналов прерывания от устройств ввода-вывода и для организации мультипрограммирования нужна управляющая программа. Исторически именно эти задачи привели к существованию супервизоров, и их дальнейшее развитие шло, во-первых, по линии дальнейшего совершенствования управления мультипрограммной работой, во-вторых, по линии добавления новых возможностей, которые было удобнее реализовать в центральной управляющей программе, и, в-третьих, по линии применения выдвинутых ранее идей в области программирования.

К первой линии мы можем отнести приоритеты в мультипрограммировании, загрузку программ, распределение памяти, защиту программ пользователя, распределение внешних устройств и управление ими, осуществление связи с оператором и восстановление системы при сбоях процессора. Ко второй линии отнесем восстановление при ошибках внешних устройств, цепочки программ, регистрацию работы системы, вспомогательные средства, обеспечивающие «посмертное вскрытие» программ, и средства, организующие off-lining. Применение идей в области программирования оказывает влияние на то, каким способом и в какой степени реализуются все эти вещи.

3.2. Приоритеты в мультипрограммировании

В первом примере, данном для иллюстрации мультипрограммирования, были две программы. Одна из них зависела от темпа работы внешнего устройства (пересылочная программа), другая — от скорости процессорной обработки (счетная программа). Мультипрограммирование состояло в том,

что вторая программа использовала для счета интервалы времени, которые первая программа тратила на ожидание своего внешнего устройства. Это достигалось благодаря тому, что всякий раз, когда еще не был получен от устройства ввода (или не принят устройством вывода) очередной блок информации, первая программа сообщала супервизору, что она не может продолжить счет. Супервизор переводил программу в режим ожидания, запоминал причину и входил во вторую программу. Когда внешнее устройство заканчивало свою работу, оно выдавало сигнал, который вызывал прерывание второй программы и уход на супервизор. Последний, выяснив причину данного прерывания, мог снять состояние ожидания для первой программы и снова войти в нее. При этом, конечно, запоминалось состояние второй программы, включая номер ее текущей команды, для того, чтобы можно было возобновить счет в том месте, на котором он был прерван. Супервизор таким образом, всегда уходил в первую программу, как только возникала возможность ее продолжения. Это и означает, что первая программа имеет более высокий приоритет в использовании центрального процессора.

Если у нас больше пересылочных программ, ясно, что мы должны дать им всем больший приоритет, чем счетной программе. Но если в данный момент существует не одна пересылочная программа, готовая к работе, то возникает вопрос: какую из них выбрать для счета? Если существуют устройства, требующие быстрого обслуживания, и при этом супервизор не может сам обеспечить такое обслуживание, выдавая, например, цепочку заранее приготовленных приказов, то мы должны программе, имеющей дело с подобным устройством, дать более высокий приоритет, чем другой пересылочной программе, связанной с более «терпеливым» устройством. Также если какая-то пересылочная программа иногда становится счетной, очевидно, имело бы смысл дать ей приоритет ниже, чем другим пересылочным программам, и выше, чем чисто счетной программе. Более тонкие различия в присвоении приоритетов часто не имеют большого значения, особенно в том случае, когда передача данных осуществляется заранее и передаваемая информация запоминается супервизором, так как при этом не важно, в какой момент между пересылками блоков программа получит свой отрезок процессорного времени, лишь бы она его получила.

Таким образом, один из реальных способов организации мультипрограммирования заключается в следующем. Каждой программе дается приоритет, который приблизительно пропорционален тому, насколько «безотлагательно» программе следует отвечать на запросы своих внешних устройств, и

обратно пропорционален количеству используемого ею процессорного времени. Супервизор всегда передает управление на программу с высшим приоритетом из тех, которые в данный момент готовы продолжать работу. Мы можем заметить здесь, что если программе необходимо иметь значительное количество процессорного времени и в то же время нужно быстро реагировать на запросы некоторых своих внешних устройств, то, вероятно, это можно организовать, разбив программу на подзадачи, как описано в 2.6. При этом каждая подзадача участвует в мультипрограммировании как отдельная задача со своим собственным приоритетом, так что «нетерпеливые» внешние устройства могут обслуживаться подзадачами с высоким приоритетом, а остальной счет может выполняться подзадачами с низким приоритетом.

Следует подчеркнуть, что приоритетная система является только инструментом для распределения времени центрального процессора. Приоритеты должны присваиваться программам так, чтобы они были хорошим исходным материалом для планирования системой своей работы и улучшения пропускной способности, а не соответствовали внешней важности или срочности работы. Приоритет программы может устанавливаться при компиляции присвоением ей подходящего числа из заданного диапазона приоритетов, однако обычно оператору или операционной системе предоставляется возможность изменять приоритет для повышения эффективности работы в отдельных случаях. Программа может также пожелать изменить свой приоритет, если, например, она перешла от счета к пересылочной фазе. Сложный супервизор мог бы присваивать приоритеты, используя обучающийся алгоритм; такой супервизор мог бы учитывать и внешние приоритеты, определяющие срочность программы, но это следовало бы делать вместе с распределением ресурсов системы.

Существуют более простые способы распределения процессорного времени. Мы могли бы считать каждую программу по очереди до момента, когда она остановится в ожидании своего внешнего устройства. Такой способ мультипрограммирования для нескольких пересылочных программ с «терпеливыми» внешними устройствами вполне достаточен, а что касается счетных программ, то их можно подключить, выдавая чистые «фиктивные» заказы на внешние устройства. Другой способ заключается в том, что каждой программе выделяется фиксированный квант времени (time slice) всякий раз, когда она готова к выполнению. Этот метод предполагает наличие в аппаратуре «часов», которые вызывают прерывание текущей программы, израсходовавшей свой квант времени. Если квант сделать достаточно малым, то этот метод подойдет для многих

работ, хотя и не позволит очень быстро откликаться на обслуживание внешних устройств. Однако возникает проблема — как сделать квант времени малым и при этом не обременить себя чрезмерными накладными расходами от частой смены программ. В предельном случае по этой схеме можно переключаться на другую программу после выполнения каждой команды при условии, что у каждой программы имеется свой набор машинных регистров. Такая схема была реализована на одной системе, но аппаратное переключение с программы на программу и необходимость дублирования аппаратуры повлекли, с одной стороны, потерю гибкости системы, и с другой — удорожание установки.

Выделение фиксированных квантов времени каждой программе, в отличие от метода с приоритетами, позволяет одновременно выполнять несколько счетных программ. Но для пакетной обработки делать это бессмысленно, так как суммарная пропускная способность при этом не увеличится по сравнению с последовательным выполнением программ. Работа с относительно большими квантами времени используется в системах мультидоступа, но, как уже говорили выше, в таких системах не ставится цель повысить пропускную способность системы.

На практике, следовательно, ни счет «до вынужденной остановки», ни «выделение фиксированных квантов времени» не дают лучшего алгоритма для универсальных систем мультипрограммирования, чем системы с приоритетами; и практическое применение каждого из этих способов в общем случае не так просто, как могло бы показаться с первого взгляда. Однако они приемлемы во многих случаях и могут быть также реализованы в системах без прерываний от устройств ввода-вывода.

3.3. Загрузка программ, распределение оперативной памяти и т. д.

Коль скоро мы имеем систему, организующую с помощью супервизора мультипрограммный режим работы, необходимо рассмотреть, что происходит, когда одна программа закончила работу и нужно запустить другую. Поскольку важно не нарушить работу ранее запущенных программ, супервизор должен или содержать в себе или уметь вызывать загрузчик программ. Но так или иначе нужно, чтобы несколько программ могли находиться в памяти одновременно. Можно было бы добиться этого простейшим способом, разбив память на несколько областей и считая, что каждая программа рассчитана на работу в данной области и ни в какой другой. Но этот вариант нельзя считать удовлетворительным, поскольку мы как мини-

мум должны иметь возможность выполнять программу в разных местах памяти. Поэтому написанные в программе адреса мы должны уметь превращать в «абсолютные» адреса в соответствии с фактическим расположением программы в памяти. Самый прямой способ состоит в следующем. Каждой программе выделяется связанная область памяти; при написании программы предполагается, что она начинается с адреса 0 и затем ко всем адресам памяти в программе прибавляется «база», т. е. расстояние от начала памяти (от абсолютного адреса 0) до того места, куда попадает адрес 0 нашей программы. Прибавления можно делать с помощью загрузчика, для чего ему должны быть сообщены значения абсолютных адресов, или же можно динамически корректировать адреса в процессе работы программы с помощью аппаратуры.

Последний вариант, хотя и требует специальных добавлений в аппаратуре, имеет ряд преимуществ. Во-первых, в обоих способах корректироваться (другими словами, базироваться) должны только адреса, а не литералы¹⁾, поэтому в первом варианте нужно уметь различать адреса во время загрузки. Во-вторых, если существуют команды, в которых адресное поле ограниченных размеров (адрес, например, содержится в индекс-регистре), то для прибавления базы в адресном поле может просто не хватить разрядов. В машинах с такой адресацией настройка адресов по первому способу невозможна. Но самое важное, что при втором варианте программы содержатся в памяти в неизменном и «перемещаемом» виде, следовательно, любые преобразования программных адресов памяти (умышленные или произвольные) не изменят результата, и в любое время, если в программе не происходит ничего особенного с точки зрения супервизора, ее можно перемещать в памяти. (Под «особенным» понимается, главным образом, передача данных между внешним устройством и областью программы.)

Если программы нельзя передвигать в памяти без соответствующей коррекции адресов («неперемещаемые» программы), то, как правило, очередную программу удобно загружать точно в освобождающееся место. Поэтому на практике для некоторого этапа работ на машине память разбивается на несколько фиксированных частей. Но если программы «перемещаемы», то все свободные участки памяти можно сливать в один связный блок и изменять длину частей, если потребуются. При этом естественно позаботиться о том, чтобы

¹⁾ Адресное поле в разных командах может использоваться в разном смысле: как собственно адрес (ссылка в оперативную память) или как «литерал» (буквальная величина), например величина сдвига, маска, константа и т. п., т. е. буквально воспринимаемое значение. — *Прим. перев.*

программы были расширяемы и могли воспользоваться преимуществами от увеличения доступной памяти.

Существуют другие решения проблемы распределения памяти, не требующие расположения программ в связных областях. Одно из решений состоит в создании такой архитектуры машины, когда адреса в программе задаются с помощью смещений относительно базового значения, хранящегося в таблице, которой заведует супервизор. Это позволяет разным частям программы располагаться на несмежных кусках памяти, но так как они имеют разную длину, то все же остается необходимость объединения свободных областей и, следовательно, перемещения частей программы в памяти. Другой способ заключается в том, что память разбивается на блоки, например, по 1024 слова в каждом, и каждой программе отводят целое число блоков, не заботясь об их связанности. И затем аппаратно во время работы часть адреса, заданного программистом, подменяется номером фактического блока, который берется из таблицы, сформированной при загрузке. Чтобы это делать с разумной скоростью, естественно, требуется значительная аппаратура и, следовательно, этот метод применим только на больших машинах. Однако раз уж в этом методе существует процедура подмены адресов, то возникает возможность вылавливать адреса, которым не соответствует ни один блок в памяти. Этот метод можно использовать для адресации в блоки, расположенные во внешней памяти, которые вынуждены перекрывать друг друга в оперативной памяти¹⁾. Здесь мы еще раз коснулись большой темы, называемой «страничным сегментированием» (paging), обсуждать которую более глубоко мы не будем.

Но при любом методе необходимо выполнение следующего важного условия. Ни одна программа не должна случайно портить супервизор или другую программу, участвующую в мультипрограммной работе. Теоретически защита от ошибок программы не является абсолютным требованием. Это можно аргументировать тем, что риск в использовании тщательно отлаженных программ не больше, чем, например, вероятность получить сбой при работе аппаратуры, или же тем, что при использовании подходящих языков высокого уровня и компиляции риск может стать незначительным или вовсе исчезнет. Но полагаться на «отлаженность» программы неразумно, и, кроме того, исчезает очень полезная возможность отлаживать новые программы параллельно с работой уже отлаженных программ. Второй аргумент, конечно, имеет смысл, если мы

¹⁾ Здесь автор имеет в виду возможность по возникающему при такой «адресации» прерыванию уйти на программу, организующую автоматическое чтение блока из внешней памяти. — *Прим. перев.*

уверены, что все программы написаны на языках достаточно высокого уровня, а машина и компиляторы спроектированы так, чтобы исключить какое-либо вредное влияние одних программ на другие. Это один из способов осуществить защиту, и он используется по крайней мере в одной вычислительной системе. Однако ситуация, когда работа одной программы вызывает таинственные ошибки в другой, потенциально так опасна (очень трудно разобраться в случившемся), что та или иная форма защиты считается необходимой почти во всех проектах машин с мультипрограммированием.

Защиту можно осуществить, выделив двоичный разряд защиты для каждого блока памяти и меняя его значение при переключении с программы на программу. При выборе размеров блока приходится идти на компромисс, поскольку желательно иметь отводимый блок как можно меньших размеров, а с увеличением числа разрядов защиты растут накладные расходы. В других вариантах блоки памяти помечаются специальными программными номерами, и при обращении к памяти этот номер сравнивается с номером текущей программы. В одной из систем помечается даже каждое слово памяти, имеющее для этой цели дополнительно четыре разряда.

Если программы размещаются в связанных областях памяти, то нижняя и верхняя границы каждой области могут храниться в некоторых регистрах и все адреса перед обращением к памяти могут аппаратно сравниваться с ними. В тех случаях, когда осуществляется аппаратное базирование, т. е. сложение адресов с нижней границей, оказывается достаточным добавить сравнение на «предел». Добавляя пары «база — предел», можно за счет увеличения числа регистров и сравнений получить область памяти из нескольких фрагментов, однако уже примерно при четырех фрагментах этот метод оказывается неконкурентоспособным по сравнению с другими из-за высокой стоимости и потерь времени на сравнениях. Подводя итог, отметим, что наиболее естественным с точки зрения защиты являются страничная память и метод подмены адресов, при котором адреса недопустимых блоков обнаруживаются в момент подмены.

Чтобы предотвратить нежелательные изменения чужой программы, достаточно блокировать только запись в чужие области памяти, что позволит программам иметь общие подпрограммы, использовать общие таблицы и организовывать связь друг с другом. Но все не так просто, как может показаться. Осложнения возникают из-за изменчивости адресов и перемещаемости программ, а также из-за недостаточной их секретности, что в некоторых применениях может быть существенным. И наконец, хотя это не так важно, случайные чтения

из мест, расположенных вне области программы, могут привести к неопределенной информации, что затруднит диагностику ошибок в программе. Часто поэтому осуществляется абсолютная защита, т. е. запрещается доступ любого вида в чужие области. В более развитых системах, где желательно воспользоваться преимуществами обоих методов, встраивается управление разными типами доступа. Для этого, например, на каждый блок отводятся по три разряда защиты, заведующих соответственно выборкой операнда, записью и выборкой команды. В схеме со страничной памятью проблему адресации во внешнюю область программы можно решить подменой (возможно, и разных) адресов блоков разных программ на один и тот же абсолютный адрес; так, например, мы могли бы иметь одну программу, формирующую некоторую таблицу в ее собственном блоке с адресом 3, и одновременно другую программу, имеющую доступ только для чтения из этой таблицы в ее собственном блоке с адресом 7.

3.4. Управление внешними устройствами и их распределение, связь с оператором

По тем же причинам, по которым супервизор занимается распределением оперативной памяти и процессорного времени между программами пользователя, он должен вести распределением внешних устройств. Если программа, например, использует четыре ленты на установке с восемью лентопротяжками, желательно иметь возможность пользоваться любыми четырьмя лентопротяжками из восьми. При этом также хотелось бы застраховаться от вмешательства в чужую программу при случайном использовании ее внешних устройств. Самое простое для этого — разрешить в программе использовать свои «программные» (или «логические») номера, скажем 0, 1, 2, 3, а супервизору поручить заменить их на физические адреса лентопротяжек, которые были для данного прохождения программы сопоставлены с соответствующими программными номерами. (Эту замену адресов можно не делать аппаратно, так как большая скорость здесь не нужна.)

Для супервизора самый простой способ сопоставления физического устройства с логическим номером, видимо, состоит в фиксации первого упоминания данного номера в программе и привязке его в этот момент к любому свободному устройству требуемого вида. Это один из приемлемых способов привязки для устройств чтения карт, печатающих устройств и т. д., которых на установке не более двух-трех каждого вида. Однако даже в этом случае лучше иметь в программе отдельный приказ — заказ на устройство, выдаваемый

супервизору перед первым обращением к этому устройству. Это облегчает осуществление привязки в подготовительной фазе программы, в которой при необходимости супервизор может сообщить программе пользователя, что устройства требуемого вида нет, или сообщить о каких-либо особенностях выделенного устройства. Защита также облегчается, так как любую ссылку на незаказанный программный номер можно считать ошибкой. Кроме того, любые специальные требования (например, требование о выделении печатающего устройства со 120 позициями в строке при наличии устройств разного типа) могут быть перенесены в заказ и не указываться в обычном обращении к устройству для пересылки информации. Можно пойти дальше и в обычном обращении не задавать никакой информации о типе устройства, кроме его программного номера, а в заказе будет говориться о том, что это за устройство — устройство чтения карт или что-то другое.

Еще один способ спецификации устройств с помощью заказа в программе заключается в том, чтобы требования на устройства задавать дополнительно в момент загрузки программы. Преимущество этого способа состоит в том, что программа, запросы которой на внешние устройства в данный момент удовлетворить нельзя, не загружается. К сожалению, иногда этот способ оказывается недостаточно гибким. В некоторых случаях программы способны после определенной перестройки работать без какого-либо устройства (например, без второго устройства печати) или же допускают замену устройств. Также нежелательно, например, абонировать выводной перфоратор в начале работы длинной программы, если вывод на него потребуется только в конце. По этим причинам привязку к устройствам в момент загрузки обычно используют как еще одну возможность в дополнение к привязке устройств с помощью заказа в программе.

Однако способ случайного выбора из свободных устройств требуемого вида неудовлетворителен для лент. Здесь очень существенно иметь возможность устанавливать бобины заранее до привязки к программным номерам, особенно если лента уже участвовала в предыдущей работе. Для этого в начале каждой ленты (в первом блоке) имеется «головная метка» в некотором стандартном виде, содержащая имя ленты (конечно, имя желательно иметь в любом случае, хотя бы для контроля). В заказе на ленту указывается ее имя, так что, получив его, супервизор будет просматривать имена на непризнанных лентах в поисках той, имя которой совпадает с именем в заказе. Если нашлась такая лента, то она будет привязана; в противном случае будет выдано сообщение программе об отсутствии требуемой ленты или программа будет приостановлена.

новлена, а от оператора потребуют установить нужную ленту. Метка, по которой ленту можно опознать, нужна даже в том случае, когда лента предназначена только для вывода результатов. Для своего вывода программа может заказать конкретную ленту или в некоторых системах запросить любую из набора доступных лент, которые, например, можно было бы идентифицировать по содержащемуся в метке сроку хранения, и использовать для вывода любую ленту, срок хранения которой уже истек. Программа (в последнем случае, по крайней мере) должна иметь возможность вписать свою собственную метку в отданную ей ленту, и было бы удобно, если бы супервизор мог это делать вместе с привязкой к ленте. Эта процедура привязки с присвоением имени требуемому файлу называется «открытием» файла.

Метод привязки лент можно также применять к другим вводным устройствам. Следует заметить, однако, что, если у супервизора нет возможности повторить чтение головной метки, то он должен эту метку помнить. Запоминание метки желательно также и для сокращения числа повторных чтений начального блока, но тогда очень существенно, чтобы супервизору надлежащим прерыванием сообщалось о смене ленты на лентопротяжке.

Во всех случаях, когда привязка осуществляется без использования опознавания документа (файла) по его метке, должен существовать какой-то способ, позволяющий оператору принудительно отдавать конкретные устройства программе пользователя. Это также верно и для устройств вывода. Вообще говоря, по отношению к печатающему устройству всегда можно определить «хозяина» выдачи, отпечатав перед выдачей некоторую «титულную» страницу. Однако если печатать нужно на заранее заготовленном типографском бланке, то привязка к произвольному устройству будет сопряжена как минимум с потерей времени на установку соответствующей бумаги.

При рассмотрении устройств прямого доступа мы встречаем другие требования. Необходимо, очевидно, позволить нескольким программам пользоваться одними и теми же дисками или барабанами. В системах со сменными дисками пакет дисков можно рассматривать как ленту и отдавать его целиком в распоряжение одной программы. Однако и в этом случае иногда желательно, учитывая свойства устройств прямого доступа, позволить нескольким совместно работающим программам хранить небольшие файлы и рабочие области на одном общем пакете. Если это допустить, то надо выяснить, когда можно заменять пакеты дисков, поэтому обычно работа с совместно используемыми пакетами ведется так же, как с

несъемными дисками. Совместное использование может распространяться на программы, имеющие доступ к общим частям на дисках или барабанах (это особенно полезно в отношении системных данных и библиотечных подпрограмм), и может также служить средством связи между программами. Эта общедоступная область во многом сходна с общей областью оперативной памяти, особенно когда она используется в качестве рабочей области: при этом возникают те же самые проблемы распределения и защиты. В некоторых системах супервизор управляет этой рабочей областью аналогичным способом, осуществляя поблочное распределение, так что фактически ее можно рассматривать как расширение оперативной памяти, и здесь мы снова затрагиваем «страничное сегментирование».

Но что касается постоянно находящихся в системе данных, их естественно представить в виде файлов с прямым методом доступа (например, с произвольной или индексируемой выборкой). В этом случае супервизор, управляя доступом, должен заданный в программе адрес блока или дорожки файла превратить в абсолютный адрес на диске. Если такой механизм существует, его можно использовать и в программе для ссылок в разные области одного и того же диска под разными именами файлов, что очень удобно и теоретически, и практически, так как помогает избежать проблем, связанных с перемещениями файлов внутри диска или «растаскиванием» их по разным сменным пакетам. Описанные возможности вместе с методикой поиска лент по их меткам и преобразованиями программных номеров и имен файлов в физические адреса устройств, дорожек и т. п. оказываются очень полезными и в тех системах, которые выполняют одновременно только одну программу.

В некоторых системах файл может занимать только связную область на диске; в этом случае процесс получения физического адреса состоит просто в добавлении к программному адресу базы файла. Более развитые системы допускают дополнительные области, лежащие вне основного массива. Размещение файлов может находиться полностью под контролем супервизора, и тогда он в любой момент может переставить одни файлы, чтобы освободить место для других. При ином варианте расположение файлов может задаваться в момент их образования. Для больших файлов, расположенных на устройствах сдвигающимися головками, больше подходит последний вариант, так как в этом случае в целях минимизации движений головок расположение файла должно тщательно согласовываться с характером его использования в программе.

Теперь перейдем к связям с оператором. Оператор и его пульт представляют собой как бы некоторое устройство ввода-вывода, работа которого распределяется между различными программами. Одно из решений состоит в том, чтобы иметь несколько пультов — по одному на решаемую задачу, и привязывать их к программам как обычные устройства ввода-вывода с той лишь разницей, что пульт определяет программу, подлежащую загрузке, а не программа привязывает себя к пульту. Но в этом нет нужды, и, если пультом является телетайп (а это практически везде так), достаточно печатать имя программы перед выводом сообщения, и оператор также должен указывать имя программы в передаваемом им сообщении. Этого вполне достаточно для управления тремя или четырьмя потоками программ. При большем числе потоков могли бы возникнуть трудности, однако в этом последнем случае нужно рассчитывать на помощь мощной операционной системы. Супервизор должен уметь определять программу, на которую ссылаются идущие от телетайпа сообщения. И добавлять имя программы в сообщения, которые выдаются на телетайп из программы. Кроме того, супервизор генерирует свои сообщения, которые зависят от самой системы, но могут, например, включать в себя и индикацию программных ошибок или сообщения о том, какие устройства привязаны к каким программам. Иногда оператору нужно передать сообщение супервизору, например, о том, какую программу загружать следующей. Таким образом, супервизор включает в себя существенную часть средств, связанных с управлением при помощи операторского телетайпа и с интерпретацией сообщений.

3.5. Супервизоры и операционные системы

Чтобы полностью обрисовать супервизор, мы должны обсудить, что общего и в чем отличие супервизора и операционной системы. Мы видели, как стремление выполнять операции параллельно приводит нас к центральной управляющей программе, супервизору. Мы упоминали, что другое направление дальнейшего увеличения эффективности — это сокращение вмешательств оператора. Следовательно, система должна получать заранее инструкции о том, что надо делать, и иметь управляющую программу, которая соответствующим образом реагировала бы на происходящие события. Естественное развитие приводит к тому, что на эту программу возлагаются функции распределения ресурсов и ввода информации, нужной той или иной работе заранее до начала выполнения этой работы (off-lining). Эта программа и образует ядро того, что принято называть операционной системой.

Исторически сложилось так, что при создании центральной управляющей программы разные разработчики обращали внимание на различные стороны вопроса. Но, несмотря на различие подходов, главные управляющие программы были созданы и получили такие названия: исполнительная, главная, супервизорная программа или операционная система. Термин «операционная система» шире других, он обычно включает в себя служебные программы, например программы ввода-вывода в режиме off-line, и может включать в себя компиляторы, действующие совместно с системой.

Внутри системы иногда полезно различать две системы: внутреннюю — «супервизор» и внешнюю — «операционную систему». В системе среднего или большого размера обязательно существует сложная управляющая программа, которая на практике как-то разбита на части. Одно из разумных разбиений состоит в выделении относительно быстрого «микроуровня», управляющего параллельно выполняемыми операциями, программы которого сменяют друг друга с частотой примерно раз в миллисекунду и поэтому должны постоянно находиться в памяти, и относительно медленного «макроуровня», определяющего, какая из программ должна вызываться в машину следующей. «Микроуровень» может образовать внутреннюю часть, которая обращается к внешней части при возникновении в программах определенных ситуаций. Мы можем назвать эту внутреннюю часть супервизором. Он будет поддерживать мультипрограммную работу программ, находящихся в оперативной памяти, заведовать обменом с внешними устройствами и управлять прерываниями. А такие функции, как связь с оператором, распределение внешних устройств, фиксация ошибок в программе или определение момента, когда программа исчерпала свое время, отнесем во внешнюю часть.

Это разделение возникает только в связи с реализацией. Нет никакого априорного деления функций между супервизором и внешней операционной системой, хотя вторая часть всегда больше тяготеет к управлению функциями «макроуровня» в операционной системе. Истинное положение границы между частями варьируется от системы к системе и должно выбираться только исходя из соображений эффективности и простоты организации. Это обстоятельство вместе с «растяжимостью» границы (или размеров) операционных систем (включение компиляторов, например) требует соблюдения осторожности при сравнении супервизоров и операционных систем на разных машинах.

Более глубокую специфику можно обнаружить в системах с двумя или большим числом уровней управления. В предыдущем разделе мы приводили разные соображения в пользу

таких систем, где базовый супервизор управляет всей системой в целом, а одна из программ, работающая под его управлением, сама является управляющей программой для своих объектных программ. Как было сказано, это делается для обеспечения в операционной системе дополнительных или специальных функций. На этом пути мы получаем самостоятельную операционную систему, «отделимую» от супервизора.

Следовательно, мы различаем два подхода к центральной управляющей программе — «операционный» подход и «супервизорный», а в частных системах мы различаем «супервизор» и «операционную систему». Но деление, если оно существует, делается по чисто практическим соображениям и в большинстве случаев будет приблизительно соответствовать функциональным различиям. В малых системах почти наверняка эффективна единая центральная управляющая программа, и любой самостоятельный супервизор, способный сам управлять своими объектными программами, должен содержать в себе некоторые типичные свойства операционной системы.

3.6. Основные функции супервизора

Чтобы проиллюстрировать сказанное до сих пор, опишем основные функции супервизора. Супервизор, выполняющий все перечисляемые ниже функции, можно было бы отнести к системе, которая несколько больше минимальной среди способных поддерживать мультипрограммную работу. Такая система, очевидно, должна уметь работать с языком управления работами (заданиями), который рассматривается в гл. 4. Здесь же он только упоминается в связи с влиянием на другие выполняемые функции. На такой системе при наличии соответствующих устройств в качестве пультов можно также реализовать, но с некоторыми ограничениями, подкачку программ, необходимую для режима мультидоступа.

1. Внутренние функции

Обеспечение функций центрального процессора, не реализованных аппаратно. Осуществляется с помощью экстракодов. Обеспечение таких функций может понадобиться при создании ряда совместимых машин.

2. Управление вводом-выводом

Проверка корректности поступающих от объектной программы команд на ввод-вывод. Замена «программных» номеров и адресов устройства на «физические». Управление устройствами ввода-вывода (обеспечение правильного порядка и своевременности выдачи управляющих

команд). Анализ прерываний от устройств ввода-вывода, получение сведений о состоянии устройств, анализ и восстановление при ошибках; выдача результатов объектной программе и перевод в нужных случаях программы из состояния ожидания в состояние готовности работать. Управление цепочками связанных команд, оптимизация движений головок на дисках и т. д. Обеспечение макрокоманд для работы с устройствами (например, независимые с точки зрения устройства операции перемещения головок и передачи данных могут задаваться одной командой в объектной программе). Возможно принятие мер, обеспечивающих совместимость для объектных программ периферийных устройств одного класса, но с разными характеристиками.

3. Мультипрограммирование

Распределение времени процессора между объектными программами. По схеме, основанной на приоритетах, перевод при определенных условиях программ в состояние ожидания (ожидание ввода-вывода, ожидание действий оператора, действий, выполняемых другой программой или подпрограммой, и т. п.) и обратный перевод программ в состояние готовности; выбор программы с высшим приоритетом среди готовых к счету программ. Обеспечение мультипрограммного режима работы для частей одной программы. Подкачка программ в режиме мультидоступа. Совмещение во времени системных операций с работой программ пользователя. Предоставление возможности сменить приоритеты для перераспределения процессорного времени. Предоставление возможности писать программы независимо друг от друга. Защита от нежелательного влияния одних программ на другие и на супервизор как с помощью аппаратных средств защиты, так и с помощью программных проверок границ передаваемых массивов. Обеспечение допустимых межпрограммных связей. Распределение ресурсов.

4. Управление оперативной памятью

Распределение оперативной памяти между программами. Обеспечение относительности адресации (или возможности переработки адресов) совместно с аппаратурой и, следовательно, независимости программ от места расположения в памяти. Перемещение программ в памяти в случае необходимости. Управление страничным сегментированием, если оно есть в машине.

5. *Распределение устройств ввода-вывода*

Привязка устройств ввода-вывода к программам (сопоставление программных и физических устройств) в соответствии с типом устройств и (или) метками. Организация совместного использования некоторых внешних устройств несколькими программами, а также распределение областей внешней памяти прямого доступа. Работа с метками и оглавлениями файлов.

6. *Off-lining*

«Переключение» потоков информации для устройств, работающих в режиме off-line. Организация совмещения ввода-вывода в режиме off-line с другими работами. Более подробно о режиме off-line см. в 4-й и 5-й гл.

7. *Выявление ошибок*

Выявление некорректных команд в объектных программах и выполнение соответствующих действий (может быть с учетом указаний, заданных на языке управления работой). Фиксация некоторых сбоев аппаратуры, например сбоев по четности в оперативной памяти, накопление информации для инженеров, обслуживающих аппаратуру, обеспечение дальнейшей работы не затронутых сбоем программ. (Об ошибках ввода-вывода см. также п. 2, Управление вводом-выводом.)

8. *Журнал работы системы*

Выдача на телетайп или запись в файл на диске и т. п. всей информации о работе системы. Эта информация служит для сбора статистики (включая частоту отказов в работе устройств) и (или) для расчетов с пользователями.

9. *Загрузка программ*

Загрузка программ. Поиск нужной программы в файле программ. Возможность загружать программу по указаниям, заданным в программе, или в потоке управляющих сообщений, или в описании работы.

10. *Связь с оператором*

Управление связью от объектной программы к оператору и обратно; обеспечение возможности отменять или заменять действия, которые выполнялись бы в соответствии с описанием работы. Предоставление оператору информации о состоянии системы и о действиях, тре-

буемых от него (установка ленты, например). Получение от оператора директив о том, какую программу загрузить следующей, какое устройство выделить программе и т. д.

11. Разное

Предоставление оператору возможности, например, распечатать содержимое части оперативной памяти, в частности содержимое памяти, отведенной программе. Возможность информировать программу например, о времени и дате, о доступных средствах системы и о собственных характеристиках (например, о размере выделенной для нее оперативной памяти или о физических номерах ее внешних устройств).

ОПЕРАЦИОННЫЕ СИСТЕМЫ, ВВЕДЕНИЕ В GEORGE 1 И GEORGE 2

Питер Беркиншоу

Считая известными основные функции супервизора, Питер Беркиншоу идет дальше и высказывает некоторые идеи, касающиеся того, как сократить вмешательство человека-оператора в работу системы. Он более детально рассматривает некоторые цели и исторические предпосылки операционных систем, упомянутые в гл. 1, и излагает наиболее важные принципы языка описания работ, иллюстрируя изложение примерами из системы GEORGE 1. Затем он обсуждает понятие макрокоманд и более глубоко рассматривает off-lining. В заключение описываются возможности off-lining в системе GEORGE 2.

4.1. Введение

За последние годы большинство пользователей вычислительных машин стало считать операционные системы необходимой частью того окружения, в котором им приходится работать; особенно это касается больших и сложных машин. Операционные системы представляют собой компромисс между двумя иногда противоречащими друг другу целями.

1. *Повышение эффективности использования ресурсов машины путем:*

- (а) минимизации времени, затрачиваемого на переключение с работы на работу;
- (б) максимизации загрузки внешних устройств; при этом, например, гарантируется высокая плотность информации на операцию пересылки (в инженерной терминологии «увеличение отношения уровня сигнала к уровню шумов»), а также приближение рабочей скорости устройств, насколько это возможно, к максимальной;
- (с) максимизации количества арифметических операций, выполняемых в единицу времени, за счет, например, буферизации внешних пересылок или даже за счет полной автономизации (отсоединения) устройств ввода-вывода от программ во время их выполнения;
- (d) минимизации времени пребывания в машине «коротких» работ по отношению к времени пребывания «длинных» работ;

- (е) разрешения возможно большему числу пользователей работать одновременно в режиме коллективного пользования;
- (f) сведения до минимума требований квалификации, необходимой для работы с системой.

2. *Обеспечение «безопасности» использования машины в смысле повышения уверенности в том, что задача будет решена правильно (а не в смысле степени секретности, связанной с задачей) при помощи:*

- (а) хороших интерфейсов;
- (b) хорошей управляющей информации;
- (с) диагностической информации;
- (d) соблюдения стандартов;
- (е) введения новых стандартов там, где это нужно.

Чтобы задача была решена правильно, она, естественно, должна быть как-то описана. Это уже предполагает наличие интерфейса между человеком, желающим выполнить работу, и человеком, сидящим за пультом и «нажимающим кнопки». Интерфейс мог бы быть реализован и вне контекста операционной системы, но только ценой больших потерь как по производительности, так и по надежности, что многие сочли бы нерациональным. Если же мы имеем описание работы в некоторой закодированной форме, такой, что оно может быть воспринято и обработано операционной системой, то мы говорим о *языке описания работ*, или *языке управления работами*. Если при выполнении работы возникает какая-нибудь ошибка и она обнаруживается операционной системой, то в интерфейсе должно появиться сообщение об этом. Для увеличения эффективности иногда бывает полезно уменьшить свободу программиста или пользователя в некоторых направлениях. На деле это является не таким большим ограничением, как кажется, ибо наложение стандартов может быть очень благотворным. Стандарты могут дать в руки набор подручных инструментов, от которого все будут получать пользу и который в противном случае многие вынуждены были бы делать сами. Но необходимо также помнить, что налагаемые стандарты должны учитывать разнообразие работ, предназначенных для выполнения, и что гибкая операционная система должна многое разрешать.

4.2. История развития стандартов операционных систем

Язык описания работ является в действительности самой последней ступенью в процессе стандартизации операционных процедур, который продолжался в течение долгого времени.

Чтобы оценить пройденный путь, проследим за изменением обстановки в машинном зале, для чего мысленно перенесемся назад в дни, когда вычислительные машины были только что изобретены и никакого программного обеспечения еще не существовало. У нас есть машина с перфокарточным вводом, выводом и построчной печатью и центральный процессор, состоящий из пульта управления, арифметического устройства, устройства управления и памяти. Давайте попробуем перечислить проблемы, с которыми мы можем столкнуться при использовании этого оборудования; подумаем, как решать их, и что может случиться, если эти проблемы оставить без внимания.

Первос, что мы должны сделать, если вопрос касается схемных управляющих панелей или коммутационных досок, — это установить форматы ввода-вывода. Например, мы должны определить формат вводимых карт, содержащих команды программы. Если мы этого не сделаем, то каждый изобретет для себя собственный формат и нам понадобится большое число коммутационных досок — от одной на каждого программиста до одной на каждую программу, в зависимости от того, как часто каждый программист будет менять свой формат, прежде чем выработает удовлетворительный для себя стандарт. Это крайне нежелательно, так как каждый программист, фактически дублируя работу других, потратит много времени и усилий; машинный зал превратится в комнату, захлавленную коммутационными досками; машина будет все больше и больше простаивать, пока оператор найдет нужную доску, и будет расти число работ, запускаемых с чужими коммутационными досками.

Поэтому мы устанавливаем стандартный формат для программы на перфокартах и стандартизуем коммутационную доску, обеспечивающую их чтение. Идя по этому пути, мы стандартизуем также форматы вводимых данных на картах и выводной формат для печати. Такая стандартизация является важной частью любой операционной системы.

Следующий вопрос, к которому мы обратимся, — это время загрузки (ввода) программы. Если на каждой карте находится только одна машинная команда, то для этого может потребоваться несколько минут. Пока программы содержали лишь сотни команд, такой формат был вполне приемлем. Но теперь, когда программы состоят из тысяч команд, время, затрачиваемое на их загрузку, становится чрезмерно большим. До сих пор необходимые указания о загрузке (приказы загрузки) располагались на тех же картах, что и сами загружаемые команды, за исключением приказа чтения самой первой карты, задаваемого с пульта. Было бы удобнее на вре-

мя загрузки иметь в памяти программу, которая могла бы обрабатывать программные карты, содержащие более чем одну команду. Поэтому мы пишем программу-загрузчик, которая на протяжении всего процесса загрузки находится в зарезервированной области памяти. Это означает, конечно, что ни одна программа, использующая этот стандартный загрузчик, не может содержать команд в отведенной загрузчику области памяти, но эта область может быть использована уже загруженными программами, как память для переменных, вырабатываемых в процессе выполнения этих программ.

Мы ускорили процесс загрузки, теперь мы должны сделать его более эффективным. Все знают, что первая или последняя карты программы обладают досаднейшей привычкой теряться или, хуже того, неделей позже внезапно появляться в качестве неверного конца другой программы, делая неработоспособными две программы сразу. Поэтому, раз уж мы все равно имеем загрузчик в памяти, можно поручить ему при загрузке «проверять» поля карты, характеризующие принадлежность карты к той или иной программе, порядок следования карт и общее число карт. Поручая загрузчику выполнять перечисленные функции, мы решаем некоторые задачи, о которых раньше говорилось как о задачах операционных систем.

Следующее, что мы обнаруживаем к своему изумлению, это то, что наши программы не работают с первого раза, как бы коротки они ни были. В итоге мы вынуждены тратить часы своего и машинного времени, сличая слово за словом содержимое памяти с тем, что по нашему мнению должно там находиться. Нам приходится это делать, если в нашем распоряжении нет программы печати или перфорации содержимого памяти, т. е. программы *дампинга*, которая бы дала нам возможность осуществить «посмертное вскрытие» отлаживаемой программы за нашим столом. Как правило, программа дампинга занимает примерно тот же самый объем памяти, что и стандартный загрузчик. Так как они никогда не нужны одновременно, то они могут занимать одно и то же место в памяти и загружаться по мере надобности. Из этого следует, что теперь мы вообще не можем использовать место, отведенное загрузчику, но оно и раньше могло использоваться только с некоторыми ограничениями, так что потеря невелика.

Поначалу мы сами пропускали наши программы на машине. Это вполне естественно, пока идет отладка, но становится скучным, после того как программа полностью отлажена и кто-то другой пожелал ею воспользоваться. К сожалению, только программист, который написал программу, знает, как с ней работать. Действительно, как кто-то другой узнает,

что остановка программы на какой-то конкретной команде (с какой-то индикацией на пульте) означает, что необходимо загрузить другой набор данных в вводное устройство и продолжить работу с некоторой другой точки? Поэтому следующая задача, которую мы должны решить, — это стандартизировать индикацию так, чтобы ее мог распознать *оператор*. Оператор может не уметь писать программы, так что прежде всего возникает проблема общения. Например, оператор не знает, куда передать управление в программу после загрузки. С этой целью мы увеличиваем загрузчик, чтобы он мог понимать карту другого сорта, которая говорит: «Не *загружай* меня, но *сделай* это». Далее на карту, которая является последней картой программы, мы помещаем команду: «Передай управление на точку входа». Другая вещь, которую оператор может не распознать, — это индикация, означающая: «Пожалуйста, загрузите колоду карт, имеющую в столбцах 47—53 алфавитный эквивалент двух цифр, которые набраны на пульте в двоичном коде XXXXXXXX». Если, однако, мы приготовим стандартную головную карту для данных — так называемый *заголовок документа* — и поместим ее перед пакетом данных, то программа загрузки-входа сможет проверить по этой первой карте соответствие подложенных данных и предотвратит неправильную работу программы. Теперь наша стандартная программа загрузки-входа выполняет множество функций, обеспечивающих правильное и эффективное выполнение программ. Фактически она является частью нашей операционной системы.

Продвинемся на несколько лет вперед и снова пересмотрим ситуацию. Оборудование теперь включает несколько магнитных лент, барабаны или диски, большую и более быструю память. Что касается операционных аспектов, то программы теперь загружаются с диска, барабана или с магнитной ленты столь же успешно, как с карт или бумажной перфоленты. Наша примитивная программа-загрузчик заменена постоянно находящимся в памяти супервизором, который включает в себя наш первоначальный стандартный загрузчик, общается с оператором, посылает сообщения пользователю на печатающее устройство и выдает сообщения для различных других служб: для людей, ответственных за библиотеки на лентах, для людей, ведающих финансовыми расчетами за работу машины, для инженеров обслуживания и т. д. В супервизоре воплощены стандарты, сложившиеся на протяжении многих лет, — он выполняет указания, информирует, командует, допуская к выполнению работы, подготовленные в соответствии с правилами системы, и отвергая работы, нарушающие эти правила.

4.3. Описания работ

Работа — это попросту все то, что пользователь желает сделать за один выход на машину, например протранслировать свою исходную программу, загрузить и запустить программу, полученную в результате трансляции, разрешить ей прочесть некоторую совокупность данных и, когда она проработает, отпечатать содержимое некоторой части памяти. Ясно, насколько существенно описать те операции, которые нужно сделать, если они не выполняются самим пользователем. Раньше оператор либо знал, как выполнить данную работу, либо его инструктировали устно в случае короткой работы, либо он получал письменную инструкцию для более длинной и более сложной работы; наконец, если это было необходимо, ему предоставлялся специальный бланк с пояснениями и замечаниями, заполненный пользователем.

В операционной системе эта информация передается при помощи последовательности предложений, подготовленных в соответствии с правилами *языка описания работ*, изложенными в соответствующем руководстве. Эта последовательность предложений называется «описанием работы» и обычно пробивается на перфокартах или перфоленге. Она вводится в вычислительную машину, и ее команды выполняются операционной системой. По существу это есть некая программа для машины, совокупность команд которой, вместо обычного набора арифметических операций, состоит из операций такого вида: «Загрузи программу X в память», «Передай управление в ячейку Y только что загруженной программы», «Выдай состояние памяти загруженной программы на устройство Z» и т. п.

Приведем пример описания работы, состоящей из трансляции программы, написанной на языке FORTRAN, и счета по оттранслированной программе.

Описание работы	Пояснения
JOB FORTRAN, 77101, PBR	Имя работы, учетный номер, имя пользователя;
IN MT (PROGRAM TAPE)	открыть вводную библиотечную ленту;
LOAD # XFAM	загрузить программу #XFAM (компилятор с языка FORTRAN) с этой ленты;
IN/0 (SOURCE)	вводимая программа на языке FORTRAN находится на том же носителе информации — картах, что и описание работы;
ENTER 1	точка входа в компилятор;

Описание работы

Пояснения

AT DELETED LO, GO TO 8	если компиляция успешна, то перейти к метке 8 в описании работы; в противном случае — конец работы, перейти к следующей работе;
END	загрузить готовую программу; вводимые данные для программы также находятся на картах;
8 LOAD # IN/0 (DATA)	войти в стандартную точку входа готовой программы; конец работы, если программа дошла до конца или остановилась в точке EP;
ENTER	в случаях других остановок точка 9 является точкой повторного входа/восстановления.
AT HALTED EP, END } AT DELETED 00, END }	При всех прочих неправильностях в работе программы напечатать ячейки с 0 по 3000.
AT OTHER HALTED, ENTER 9	Признак конца описания работы.
PRINT 0 TO 3000	

4.4. GEORGE 1

Описание работы, приведенное в этом примере, написано для системы GEORGE 1, которая в основном предназначалась для замены вручную управляемого супервизора на более крупных и быстрых машинах — средних из серии 1900. При разработке программ для машин 1905 и 1904 довольно быстро было установлено, что работы достаточно короткой продолжительности, какими является большинство пусков программ в процессе отладки, крайне плохо используют машину даже при очень квалифицированных операторах. Другая проблема состояла в том, чтобы добиться корректного выполнения сложных работ, так как при управляемом вручную супервизоре пользователь должен был либо сам запускать свои программы, либо написать инструкцию для человека-оператора, которую тот как-то проинтерпретирует. И чем сложнее была работа, тем более чувствительной она была к ошибкам интерпретации.

Операционная система, известная теперь как GEORGE 1, была разработана для решения этих проблем, но, кроме того, она должна была уметь выполнять все существовавшие программы и работы, выполнявшиеся со старым супервизором. В процессе ее разработки стало ясно, что некоторые возмож-

ности супервизора 1900, сделанного еще на заре машин этой серии, устарели. Особенно это касалось таких вопросов, как связь с оператором и защита магнитных лент при мультипрограммной работе. Кроме того, возникали трудности из-за отсутствия возможностей, которые позволили бы организовать динамическое распределение периферийных устройств. Нужно было создать операционную систему, которая могла бы стыковаться с супервизором, дополняя его теми операциями, которые позволили бы перейти от ручного управления работами к автоматическому. Необходимо было также сократить поток сообщений, выдаваемых на телетайп, и заменить некоторые функции супервизора, не соответствующие современному подходу к операционным системам. Поскольку GEORGE 1 предназначалась для замены ручного супервизора, многие возможности системы должны были стать необязательными, т. е. такими, которые было бы легко обходить в тех случаях, когда ими неудобно пользоваться. При этом важно, чтобы программа вела себя так, как будто она выполняется под управлением старого супервизора. По этой причине в GEORGE 1 все описания данных являются необязательными и не производится проверки заголовков документов или меток, если это не задано специально.

Особое внимание было уделено той части управления работами, которая позволяет предвидеть разные необычные ситуации, возникающие либо при отладке программ, либо в отлаженных программах. Предшествующие системы, разработанные главным образом в эру «управляющих карт», были признаны недостаточно гибкими. Нужен был богатый набор условных переходов, дающих возможность передавать управление назад и вперед в пределах описания работы. Все это было реализовано в GEORGE 1 в виде списка предложений АТ, в котором любое событие может быть выделено для специального внимания (точно так же, как человек-оператор может быть нацелен на то, чтобы следить за некоторым событием), но с тем преимуществом, что на языке описания работ не существует ограничений на число уровней вложенных условий и предписываемых действий.

Итак, главные черты GEORGE 1 состоят в том, что

- 1) написанные пользователем команды автоматически поступают непосредственно в систему без всякой промежуточной интерпретации человеком;
- 2) имеется возможность, используя заголовки документов и метки лент, проверить, что периферийные устройства программы загружены правильными данными, относящимися к этой программе;

- 3) проблема, связанная с возможностью совпадения меток у разных лент, может быть решена за счет введения порядковых номеров лент, которые, являясь внешними идентификаторами лент, доступны для визуального контроля со стороны оператора в отличие от имен файлов, которые оператор проверить не может;
- 4) можно с помощью деклараций в описании работы заменить имена файлов, используемых в управляющих частях программы, и соответственно информировать об этом программу;
- 5) лента может выделяться просто при помощи предложения ALLOT в описании работы;
- 6) можно указать действия, которые должны быть приняты при любом событии, если оно произойдет в процессе выполнения программы;
- 7) вмешательства оператора ограничены физическими действиями, такими, как загрузка периферийных устройств;
- 8) обеспечивается «непрерывность» выделяемых программе периферийных устройств в пределах границ, отведенных программе.

Область памяти, постоянно требуемая для GEORGE 1 (около 2000 24-разрядных слов), такова, что система может работать в малых машинах с 16 000 слов оперативной памяти. Поскольку система не обеспечивает максимальной эффективности использования периферийных устройств, если не говорить о ее большей надежности, то было сочтено целесообразным оставить пользователю возможность доступа к программным каналам (slots)¹⁾, имевшуюся в мультипрограммном супервизоре, за исключением канала (или каналов), используемого самой GEORGE 1. В 4.6 и 4.7 мы исследуем другие системы, разработанные для того, чтобы преодолеть низкую эффективность, вызванную дисбалансом в использовании периферийных устройств.

4.5. Макро

Одной из сильных, а одновременно и слабых, сторон человека-оператора является его способность обучаться, приспособляться и применять свои знания в новых ситуациях, аналогичных встречавшимся ранее. Конечно, это очень удобно, если мы можем сказать оператору, что перед ним ситуация X, которую он может распознать и осуществить сложную

¹⁾ «Программный канал» (английский термин slot — щель) — это совокупность средств в операционной системе, относящихся к одной из нескольких программ, выполняемых одновременно в мультипрограммном режиме. — *Прим. ред.*

совокупность действий, нужных в данной ситуации. Однако, с другой стороны, опасность того, что оператор подумает о некоторой ситуации как о ситуации X, хотя ему об этом не будет сказано явно, быстро растет по мере того, как увеличивается число случаев, в которых он сталкивается с ситуациями, аналогичными X, но не являющимися таковыми. Трудность состоит в том, что мы не можем управлять маской, которую применяет оператор в своем подсознательном процессе распознавания ситуации X. Для оператора также трудно бывает изменить своим привычкам, когда, несмотря на то что он находится в ситуации X, от него требуется другая реакция. Многие программисты испытали на себе опасности разработки новых вариантов для установившихся и часто применяемых пакетов программ, работающих в операционной системе с большим объемом ручного манипулирования. В тот момент, когда оператор видит знакомую программу, он говорит: «О, это программа Y, а я знаю все о программе Y, поскольку я имею с ней дело очень часто». Но, к сожалению, ситуация в этот момент другая, и, совершенно не отдавая себе в этом отчета, он делает «правильные» вещи, которые оказываются неверными в этом случае. Те действия, которые выполнил оператор, увидев программу Y, есть по существу *макропроцедура*, или просто *макро*.

Большинство современных операционных систем позволяют пользователю построить библиотеку макро с помощью программ, которые входят в систему как ее часть и служат для создания и обновления файлов таких макро. Файлы макро обычно хранятся на тех же устройствах, что и сама операционная система. Макро операционной системы обычно превращаются в набор элементарных команд машины и написаны они так, что в момент обращения к макро поля параметрических переменных замещаются строками (цепочками) символов, заданными в обращении к макро. Обеспечение макровозможностями явно выгодно с точки зрения экономии усилий, затрачиваемых при пользовании системой. Менее явное преимущество макро состоит в том, что, хотя все могут получать удовольствие от использования макробιβλιοгеки, никто не покушается на право каждого не пользоваться ею; и, конечно, остается возможность обычным образом запрограммировать все, что не может быть выражено с помощью библиотечных макро. Это, очевидно, дает бóльшую гибкость, чем ручные системы, в которых бывает трудно «обойти» сложившиеся операционные приемы. Макровозможности составляют одну из характеристик расширения системы GEORGE 1, известного под именем GEORGE 2.

На следующей странице приводится пример макро в системе GEORGE 2 для компиляции и выполнения задачи на языке FORTRAN. Макрообращение для этого макро FORTRUN задается в виде

FORTRUN %A, %B, %C, %D, %E, %F, %G

Когда в программе встретится такое обращение к макро, система GEORGE заменит его хранящимся макроопределением, произведя подстановку параметров от %A до %G:

- %A — описание документа, содержащего исходную программу
- %B — входная точка в компилятор #XFAM
- %C — метка перехода, если имеются недостающие сегменты
- %D — метка перехода, если компиляция не закончилась успешно
- %E — имя скомпилированной программы
- %F — вводные данные для скомпилированной программы
- %G — точка входа в скомпилированную программу

Так, например, предполагая, что описание работы находится на картах, для компиляции программы #TRAN, заданной на картах, и для выполнения ее, начиная с входной точки 7, после того как ее данные будут прочитаны с магнитной ленты, необходимо задать следующий набор параметров:

- %A = /0 (SOURCE) — ввод с карт для #XFAM
- %B = 1 — точка входа в #XFAM
- %C = 6 — метка в описании работы
- %D = 7 — метка в описании работы
- %E = TRAN — имя скомпилированной программы
- %F = TRO(DATA) — лента с вводными данными для #TRAN
- %G = 7 — точка входа в #TRAN

Итак, все, что должно быть написано в качестве описания работы, выглядит следующим образом:

```

JOB COMPILETRAN, 77777, THE USER
FORTRUN /0(SOURCE), 1, 6, 7, TRAN, TRO(DATA), 7
AT HALTED OK, END
AT FAIL, GO TO 8
6 DISPLAY 'SEGMENTS MISSING'
  ENDJOB
7 DISPLAY 'COMPILATION FAILURE'
  ENDJOB
8 PRINT 0 (100)
* * * *
```

Макроопределение

Пояснения

MACDEF FORTRUN

IN MT (PROGRAM TAPE)

LOAD #XFAM

IN %A

ENTER %B

```
9FR1 AT HALTED EC, GO TO 9FR2 }
AT DELETED LO, GO TO 9FR4      }
AT HALTED SM, GO TO %C         }
AT HALTED CE, ENTER 6          }
AT HALTED ST, GO TO 9FR3       }
AT HALTED LD, RESUME           }
AT FAIL, GO TO %D              }
```

```
9FR3 HALT 'NEEDS CORE'
RESUME, 9FR1
```

9FR2 IN MT (PROGRAM %E)

```
LOAD #%E
GO TO 9FR5
9FR4 LOAD #
```

9FR5 IN %F

ENTER %G

ENDMAC FORTRUN

Далее следует определение макро с именем FORTRUN; открыть библиотечную ленту на ввод; загрузить программу #XFAM (компилятор с языка FORTRAN) с этой ленты; ввести исходную программу — указывается имя и носитель; точка входа в компилятор;

метка 9FR1: список предложений «AT», задающих действия, которые нужно предпринять при соответствующих событиях, например при остановках в процессе компиляции или в конце компиляции;

метка 9FR3: возобновить компиляцию с метки 9FR1 после того, как оператор выделит добавочную память;

метка 9FR2: готовую программу под именем, заданным параметром %E, поместить на магнитную ленту;

загрузить готовую программу; перейти к метке 9FR5;

метка 9FR4: загрузить готовую программу, имя и носитель которой указывались в последнем сообщении DELETED;

метка 9FR5: ввести данные для готовой программы (имя и носитель);

точка входа в готовую программу;

конец макроопределения.

4.6. Неэффективность внешних устройств

Все обычные традиционные программы вводят данные с перфокарт или с перфоленты и выдают результаты либо на те же носители, либо на печать, или же сразу на те и другие устройства. Они могут работать также с файлами, хранящимися на магнитных дисках, лентах и т. п. Все такие программы образуют некоторый спектр по степени, в которой выполнение программ лимитируется потребностями ввода-вывода.

Они варьируются от традиционных программ «обработки коммерческих данных» до «научных», или «счетных», программ.

Программы первого типа проводят большую часть времени в ожидании выполнения команд ввода-вывода, и при этом мощность процессора фактически не используется. Программы второго типа большую часть времени, затрачивают на арифметические вычисления и только время от времени нуждаются во вводе данных или печати результатов. Ясно, что никакого абсолютного деления на два типа не существует, так как данная работа предстанет перед нами в различном качестве в зависимости от конфигурации машины, на которой она выполняется. Так, с неограниченно быстрым процессором все работы, очевидно, окажутся «коммерческими», тогда как с неограниченно быстрыми периферийными устройствами все работы окажутся «счетными».

Для данной конфигурации машины оптимальные характеристики по производительности могут достигаться только для одного сорта работ, и таковыми оказываются работы, которые выполняют свои операции ввода-вывода с точно такой же частотой, которая является номинальным максимумом для оборудования. Даже при наиболее тщательном, учитывающем специфику машины программировании вероятность того, что встретится хотя бы одна такая совершенная (оптимальная) программа, очень мала. Горькая правда состоит в том, что, когда мы рассматриваем загруженность установки в целом, мы обнаруживаем, что процессор простаивает в ожидании, пока работают основные устройства ввода-вывода, примерно столько же времени, сколько в других ситуациях простаивают устройства ввода-вывода. Величина потерь зависит от отклонения работ от оптимума. Чем больше разнообразие работ, тем большее время будет неизбежно потеряно.

Действительно ли это неизбежно? Частичным решением проблемы является применение методов мультипрограммирования, позволяющих компенсировать потери времени за счет одновременного выполнения работ, которые находятся на противоположных концах спектра задач и их потребности в периферийных устройствах не пересекаются. Крайне маловероятно, что пользователи будут гореть желанием купить больше печатающих устройств или устройств для ввода-вывода карт, чем им теоретически необходимо, хотя большинство пользователей вычислительных машин хотят, чтобы их результаты были напечатаны независимо от объема. Это противоречие может приводить к тому, что от программистов будут требовать сокращения объема выдачи на печать или

отказа от печати полностью и замены ее выдачей, например, на телетайп. В некоторых случаях, если составление программы находится еще в начальной стадии, удается убедить автора программы поступить таким образом. Но, к сожалению, даже программы, которые в конце концов обходятся небольшими выдачами, должны пройти фазы компиляции и отладки, которые неизбежно требуют больших объемов печати. Следовательно, в начале жизни любая программа имеет «коммерческий» характер.

4.7. Off-lining

Существует другое решение проблемы простоев внешних устройств и процессора, если для этой цели выделить достаточные дополнительные ресурсы вспомогательной и внутренней памяти. Эта альтернатива и есть выполнение ввода и вывода в режиме *off-line*, т. е. автономно. Под этим подразумевается передача вводных данных работы во вспомогательную память до начала выполнения работы и пересылка выводных результатов из вспомогательной памяти на соответствующее устройство после окончания работы. В процессе же выполнения работы, в зависимости от конструкции системы, программы даже могут не знать, читают ли они вводную информацию и пишут ли они свои результаты на устройства, фактически имевшиеся в виду, или нет. Реальное же чтение, печатание или перфорирование происходят совершенно независимо от основного процесса счета под управлением отдельных программ, которые даже могут выполняться на других машинах. В этом состоят основные принципы работы в режиме *off-line*.

Перечислим главные преимущества этого метода.

1. Он сглаживает колебания в запросах на работу периферийных устройств таким способом, который недостижим только при одном мультипрограммировании.
2. Поскольку вспомогательная память, используемая в качестве посредника, обычно имеет магнитный, а не механический носитель, то время пересылки от основной программы существенно сокращается, а в течение реальной фазы ввода или вывода периферийные устройства работают с максимальной скоростью.
3. Метод дает большие преимущества с точки зрения планирования загрузки внутренней памяти, потому что при этом основные программы, которые обычно много больше программ пересылки, могут работать значительно быстрее.

Имеются другие, менее ощутимые преимущества, касающиеся надежности и других аспектов системы.

4. Внешние устройства имеют тенденцию работать лучше, если они работают более равномерно.
5. Работы естественно разбиваются на три фазы: ввод, обработка и вывод. Сбой на любой из фаз делает необходимым повторение работы, начиная только с этой фазы. Подобным образом, если нужны дополнительные копии выводного материала, то они могут быть легко получены при условии, что запрос на это поступил в ограниченный период после того, как работа завершена.

Реализация перечисленных преимуществ связана со следующими непосредственными и побочными издержками:

1. Программы off-line требуют внутренней памяти и устройств для хранения файлов, таких, как магнитные ленты или области на дисках, на той же машине или на машине-сателлите.
2. Если программы off-line выполняются на той же машине, то они могут заметно усложнить проблемы планирования работы каналов ввода-вывода, которые используются и с устройствами off-line.
3. Нарушение порядка следования и другие ошибки ввода могут оказаться не выловленными до начала собственной обработки. Если это достаточно серьезно, то может возникнуть необходимость в написании специальных программ переписи для важных файлов больших объемов, производящих специальные проверки, обычно отсутствующие в стандартных пакетах.
4. Усложнение системы приводит к неясностям при возникновении ошибок, если на каждой фазе не разработаны специальные проверки, гарантирующие высокую достоверность данных.
5. Некоторые виды работ невыполнимы в режиме off-line. Примером может служить любая работа, требующая двух синхронизованных входных потоков, как, например, сравнение двух бумажных перфолент. Другие работы, такие, как диагностическая печать магнитной ленты, могут оказаться бессмысленными в этом режиме.
6. В режиме off-line не может работать программа, если она пользуется возможностями управлять устройством ввода динамически. Такие возможности иногда предоставляются аппаратурой, например динамическое изменение режима чтения с перфоленты. Аналогично не может работать программа, если она использует не-

стандартным образом управляющую информацию, поступающую от устройства, например информацию о конце страницы или конце бумаги на печатающем устройстве.

4.8. GEORGE 2

Операционная система GEORGE 2 машин серии 1900 была разработана как более совершенная версия GEORGE 1 с макровозможностями, описанными в 4.5, и с работой периферийных устройств в режиме off-line. Возможности off-line могут быть использованы одним из следующих способов:

- (а) ввод и вывод выполняются в режиме off-line;
- (б) только вывод выполняется в режиме off-line;
- (с) ни ввод, ни вывод не используют режима off-line, т. е. как в GEORGE 1.

Имеются два варианта систем GEORGE 2. В первом система располагается на магнитной ленте и использует в режиме off-line магнитную ленту, во втором система располагается на диске (или барабане) и в режиме off-line может работать как с магнитной лентой, так и с диском.

Операционная система GEORGE 2 в действительности состоит из трех различных программ: программы пересылки данных в режиме off-line с карт или бумажной перфоленты на магнитный носитель, программы off-line перфорации карт или бумажной перфоленты или вывода на широкую печать с магнитного носителя, а также центрального модуля. Самым эффективным способом использования системы является использование всех трех частей одновременно. В этом случае пакет А будет обрабатываться выводной программой, пакет В находится в стадии счета, а пакет С создается (формируется) во вспомогательной памяти программой ввода. При этом должно существовать до четырех программных каналов в зависимости от супервизора и варианта системы, находящегося в эксплуатации.

Из-за наличия дополнительных возможностей GEORGE 2 использует несколько большую внутреннюю память, чем GEORGE 1: от трех до шести тысяч 24-разрядных слов. Поэтому эта система доступна только для машин с внутренней памятью не меньшей, чем 32 000 слов. Кроме того, она требует так много другого оборудования и времени работы каналов или устройств, что на большинстве машин среднего размера становится нерациональным или даже невозможным какое-либо мультипрограммирование помимо того, что делается самой системой. Тем не менее если режим off-line использован правильно и в соответствующих условиях, то он

может дать больший эффект, чем мультипрограммирование, так как в нем достигается более регулярное и организованное совмещение операций. Следовательно, для машин среднего размера режим off-line нужно рассматривать как метод организации мультипрограммирования, а не как нечто, используемое в дополнение или в сочетании с ним.

На больших машинах под контролем супервизора в принципе могут работать различные мультипрограммные системы. Однако для лучшего использования мощности машины имеет смысл не организовывать несколько независимых потоков off-line, а объединить ресурсы машины, слить их в единый фонд и использовать для организации единого архива файлов, как это сделано в системе GEORGE 3.

КОНЦЕПЦИИ GEORGE 3

Тим Голдингем

После GEORGE 1 и GEORGE 2 Тим Голдингем рассматривает GEORGE 3 для того, чтобы проиллюстрировать возможности более совершенных операционных систем. Он коротко останавливается на архиве для хранения файлов и показывает, как его можно использовать для более гибкой работы в режиме off-line и для операций учета. Затем он развивает идею мультидоступа и его использования для разработки и отладки программ, а также знакомит читателей с идеей разговорных компиляторов. В заключение он рассматривает важные вопросы планирования работы системы.

5.1. Введение

Операционная система GEORGE 3 для машин серии ICL 1900 представляет собой мощную систему, объединяющую возможности и пакетной обработки, и мультидоступа. Поэтому она гораздо сложнее систем GEORGE 1 и 2, рассмотренных в предыдущей главе.

Чтобы понять идеи, положенные в основу системы, целесообразно рассмотреть трудности, с которыми сталкивается человек-оператор при работе на машине. От него требуется:

- (а) интерпретировать инструкции программиста;
- (б) выдавать соответствующие указания машине с помощью телетайпа;
- (с) устанавливать в рабочее состояние периферийные устройства — читающее устройство с карт, широкую печать и т. д.;
- (д) следить за тем, чтобы эти устройства постоянно были обеспечены картами, бумагой и т. д.;
- (е) устанавливать магнитные ленты и диски;
- (ф) фиксировать в журнале все, что происходит, для того чтобы в дальнейшем можно было произвести необходимые расчеты с пользователями;
- (г) в оставшееся время решить, какую подготовить смесь работ, чтобы наилучшим образом загрузить и центральный процессор, и внешние устройства.

Чтобы выдержать такую нагрузку, нужны сверхчеловеческие усилия, физические и умственные. Однако большинство из перечисленных задач по существу являются шаблонными (стандартными) операциями, т. е. как раз такими, с которыми, как говорят, хорошо справляются машины. Следовательно, целесообразно переложить на машину как можно больше таких операций.

5.2. Архив файлов

В главе 4 мы уже видели, как GEORGE 1 и 2 помогают решить некоторые из названных задач. Язык описания работ решает проблему интерпретации инструкций программиста, а использование устройств ввода-вывода в режиме off-line существенно упрощает работу оператора с этими устройствами. Что может быть еще сделано в этом направлении?

Ответ на вопрос дает *архив файлов*. Основная цель архива файлов — обеспечить постоянную доступность всей нужной системе информации. Первоначально он располагается на устройствах с прямым доступом (на дисках, барабанах и т. п.), но может переходить и на магнитные ленты. Архив файлов имеет иерархическую структуру. Файлы в архиве содержат, как правило, один из трех типов информации. Первый тип — это данные, организованные любым способом, который устраивает пользователя; для некоторых работ подойдет файл с последовательной организацией, тогда как для других может потребоваться файл с прямым доступом. Второй тип — это программы на исходном языке или протранслированные программы. Третий тип информации связан с организацией самого архива файлов, поскольку должны существовать файлы, содержащие каталоги (или оглавления) файлов, расположенных ниже их в иерархии.

Детальному рассмотрению архива файлов посвящена отдельная глава, поэтому здесь мы не будем развивать эту тему дальше; однако это понятие является фундаментальным для понимания основных концепций GEORGE 3, поскольку из него вытекают многие свойства и возможности системы.

5.3. Ввод-вывод

Рассмотрим, например, работу с устройствами ввода-вывода. В GEORGE 3 она может осуществляться двумя путями. Наиболее простым является непосредственное подключение периферийных устройств к программе пользователя (режим on-line). Это, однако, требует непрерывного внимания со стороны оператора, от чего мы стремимся избавиться, и поэтому такой способ используется только в особых случаях, например при работе с данными, имеющими нестандартную кодировку.

Более распространенной является одна из двух форм работы в режиме off-line.

В главе 4 было показано, как это сделано в GEORGE 2, где последовательность обработки совпадает с вводом. Наличие архива файлов в GEORGE 3 позволяет применить более

гибкую тактику. Вводные пакеты записываются в архив, и обращения к ним осуществляются по мере надобности. При этом может оказаться, что данные для той или иной конкретной работы, будучи введены утром, окажутся обработанными лишь во второй половине дня.

Существуют две различные реализации этого метода. Первая — это образовать файл с именем, который будет содержать вводной пакет. В этом случае в описании работы нужно задать команду

INPUT имя устройства, имя файла

и вводимые данные будут записаны в этот файл. Затем понадобится связать этот файл с соответствующей программой пользователя. Это делается при помощи команды ASSIGN. В описании работы для выполнения программы программист, например, пишет

ASSIGN * CR0, INPUTDATA¹⁾

И это значит, что всякий раз, когда в программе встретится команда чтения карты, она будет перехвачена системой GEORGE и превращена в команду извлечения очередной записи из файла с именем INPUTDATA.

Аналогично команда

ASSIGN * LP0, OUTPUT²⁾

приведет к тому, что подлежащий печати материал не распечатается сразу на широкой печати, а запомнится в файле с именем OUTPUT и будет отпечатан позже.

Вторая реализация метода работы в режиме off-line отличается от первой лишь тем, что вместо постоянного применяется временный файл. В этом случае достаточно задать команду OFFLINE и временный файл будет автоматически создан, а затем уничтожен после того, как его содержимое будет использовано.

5.4. Учет

Другой возможностью системы, связанной с архивом, является *учет*. В универсальной операционной системе, используемой разными пользователями в различных целях, должна

¹⁾ CR0 (CARD READER 0) — нулевое устройство ввода карт. —
Прим. ред.

²⁾ LP0 (LINE PRINTER 0) — нулевое печатающее устройство. —
Прим. ред.

быть предусмотрена методика расчетов с каждым пользователем с учетом стоимости тех средств (устройств), которые он использовал. Для этой цели в GEORGE 3 заводится мониторинный файл на каждую работу, которая пропускается на машине. В этот файл фактически попадает вся та информация о ходе выполнения работы, которая в менее автоматизированных системах может быть выдана на операторский телетайп: занимаемая оперативная память, процессорное время, используемые периферийные устройства и т. д. При завершении работы вызывается специальная *программа расчета*, которая анализирует накопленную информацию и сообщает пользователю стоимость работы, проделанной им. В дополнение к этому, мониторинный файл, если нужно, может быть сохранен для последующего сводного анализа.

Такая система учета уже содержит в себе некоторые из средств контроля, которые нужны директору вычислительного центра. Однако мало учитывать стоимость после того, как работа уже проделана. Нужна еще и *бюджетная система*. Известно старое правило: нужно «закрывать дверь конюшни, пока вы уверены, что лошадь еще внутри». И снова организация бюджетной системы становится возможной только благодаря архиву с его иерархической структурой, в котором хранятся детальные сведения обо всех пользователях системы.

Всякий раз, когда пользователь образует «подчиненного» пользователя в архиве, он должен выделить ему долю своего бюджета. Бюджет задается для трех ресурсов: *места, времени и денежных средств*.

Распоряжаются этими ресурсами по-разному. Ограничения по месту и времени накладываются только в том случае, когда на эти ресурсы претендуют другие пользователи системы. Предположим, например, что пользователю выделено в качестве его бюджета места некоторая область в архиве. Если он хочет воспользоваться большей областью, то его никто не ограничивает, пока эта область свободна; но как только поступят запросы на эту область от других пользователей, он будет ограничен той областью, которая ему выделена по бюджету. То же самое относится и к бюджету времени: пользователю будет позволено использовать центральный процессор в той мере, в какой ему это нужно (в зависимости от его денежного бюджета), если при этом он не мешает другим пользователям.

Денежный бюджет является барьером, который никогда не может быть превышен. Это, конечно, верно с точностью до тех единиц, в которых ведет расчеты специальная программа анализа и которые устанавливаются директором вычислительного центра. Например, процессорное время опла-

чивается по числу израсходованных минут. Кроме того, устанавливается цена за использование каждого вида периферийных устройств и некоторая изначальная плата. Занимаемая память, как оперативная, так и в архиве, оплачивается в расчете на тысячу слов и на число минут. Меняя соответствующим образом цены, директор вычислительного центра может оказывать влияние на пользователей, например, в зависимости от спроса, поощряя использование устройств определенного типа. Возможны дальнейшие уточнения, например установление дифференцированной оплаты для работ, выполняемых в дневную, вечернюю или ночную смену.

Бюджет потребителя проверяется в начале каждой работы независимо от того, ведется ли она в режиме мультидоступа или является фоновой работой: если бюджет превышен, то работа не может продолжаться до тех пор, пока начальство не выделит пользователю дополнительные ресурсы.

Распределение ресурсов производится периодически (обычно раз в месяц). Для пользователя устанавливается некоторый месячный ресурс и, если он исчерпал его в данный месяц, то ему придется для запуска своей работы ждать начала следующего месяца. Пределы, в которых пользователю разрешается переносить свой бюджет из одного месяца в следующий, находятся под контролем директора вычислительного центра. Каждому пользователю назначается *норма расхода* — сумма денег, выделенная для него *в месяц*. Фактически сумма, которую он может израсходовать в текущем месяце, т. е. его *расходный фонд*, складывается из нормы расхода плюс какая-то часть от неизрасходованной суммы предыдущего месяца, которую ему разрешили перенести на следующий месяц. Коэффициент переноса может быть установлен, например, равным $\frac{1}{2}$. Следовательно, если месячная норма пользователя равнялась 100 фунтам, и в первый месяц пользования системой он израсходовал только 80, то его расходный фонд на второй месяц будет равен $110 = 100 + \frac{1}{2}(100 - 80)$ фунтам.

Подобные бюджетные системы особенно важны на машинах с *мультидоступом*, где большое число пользователей работают с машиной при помощи выносных терминалов и где контроль за потоком работ не может производиться в машинном зале.

5.5. Мультидоступ

В главе 4 в качестве одной из возможных задач операционной системы предлагалась максимизация числа одновременно работающих пользователей. Некоторые операционные системы разработаны специально для этой цели, тогда как

другие предназначены только для пакетной обработки. При разработке GEORGE 3 одной из главных целей было создание системы, которая была бы одинаково хороша для обоих названных видов работы и в которой для обоих этих видов применялся бы по возможности один и тот же язык управляющих команд. Поэтому мы должны четко определить, что понимается под термином «мультидоступ».

Одной из основных проблем эффективного использования вычислительной машины является фантастическое несоответствие между скоростями электронного центрального процессора и механических периферийных устройств. Именно это несоответствие вызвало к жизни идею мультипрограммирования, поскольку процессор вполне может управлять одновременно работой нескольких периферийных устройств. Однако даже и в этом случае процессор, как правило, полностью не загружается. Таким образом, остаются дополнительные возможности повышения производительности машины, нужно только найти способ их использования.

Это трудно сделать в условиях обычной пакетной обработки: большинство работ такого сорта требуют нескольких периферийных устройств, а обеспечивать машину большим количеством устройств неэкономично. И вообще, является ли пакетная обработка наилучшим способом использования машины?

Причина, по которой мы обрабатываем данные пакетным способом, опять-таки заключена в уже упомянутом несоответствии скоростей. Полагали, что для того чтобы максимально загрузить процессор, нужно вводить в него данные с наибольшей возможной скоростью, загрузив их предварительно в большом количестве на очень быстрое периферийное устройство. Это идеально для многих коммерческих применений, но далеко не всегда так хотелось бы работать на машине. Может случиться, что мы хотим провести расчеты, располагая лишь небольшим количеством данных, и, более того, начиная расчеты, мы можем еще не вполне представлять их ход. Другими словами, мы хотим разрабатывать наши программы в процессе работы, просматривая промежуточные результаты вычислений и решая, что делать дальше в зависимости от этих результатов.

Оборудование, которое требуется нам в таком случае, — это не устройство чтения с карт или бумажной перфоленты, а телетайп, т. е. что-то более соответствующее нормальной человеческой скорости выполнения операций. Но как мы можем позволить себе использовать со скоростью черепахи наш быстрый процессор? Это будет экономически оправданным, если только мы подключим большое число телетайпов.

В этом случае мы действительно загрузим центральный процессор и в то же самое время удовлетворим достаточно большое количество пользователей, каждый из которых будет работать за своим пультом, находясь под впечатлением того, что он имеет в своем распоряжении всю машину, — точно так же, как кинозритель, забывающий тот факт, что большую часть времени он смотрит на темный экран. В этом состоит идея мультидоступа.

Мы представили мультидоступ и пакетную обработку как диаметрально противоположные вещи, но их вполне можно скомбинировать в одной системе, и такая комбинация является целью системы GEORGE 3. Эта система спроектирована так, чтобы обеспечить максимальное единообразие в выполнении обоих типов работ, используя один и тот же язык управляющих команд.

Представим себе пользователя, сидящего у своего телетайпа и, надеемся, не слишком растерянного, и проследим, как он работает.

Первое, что он должен сделать, — это подключиться к системе, т. е. установить свою связь с машиной и свои полномочия использовать ее. Поэтому он печатает слово LOGIN, а вслед за ним текущее *имя работы* и *имя пользователя* и ждет ответа машины.

Ответ будет зависеть от количества других пользователей: установлен предел на общее число работ, обрабатываемых одновременно, поэтому пользователь может получить сообщение (распечатку), что машина уже переполнена. Если это не так, то система GEORGE производит анализ имени пользователя.

Полностью смысл этого имени будет ясен после описания архива файлов в гл. 7. В данный момент достаточно сказать, что каждое имя пользователя запоминается, когда он впервые обращается к системе. Следовательно, когда он пытается подключиться к системе с данного телетайпа, нужно проверить по соответствующему словарю архива, известен ли такой пользователь системе.

Этого, однако, недостаточно, чтобы предотвратить какое-либо неправомерное вмешательство другого человека, выступающего под чужим именем. Поэтому, найдя в словаре архива имя пользователя, машина печатает на телетайпе слова TYPE PASSWORD (НАПЕЧАТАЙ ПАРОЛЬ), после чего пользователь должен ответить, напечатав пароль, известный только ему одному и системе GEORGE.

Точные алгоритмы управления этой проверкой отличаются для различных систем с мультидоступом. Установки, которые особенно заинтересованы в предотвращении неправомерного

доступа, могут иметь телетайпы, на которых печать может блокироваться, и поэтому пароль нельзя прочитать; или же телетайп может делать возврат на одну позицию после каждого знака, вследствие чего на бумаге будет напечатан только черный прямоугольник. Возможны также и чисто механические методы защиты, например запираание клавиатуры на ключ. Но они имеют тот недостаток, что пользователь может лишиться возможности перейти на другой телетайп, если его собственный поломан. Все эти методы, конечно, увеличивают стоимость оборудования и ими не снабжены стандартные модели серии 1900.

Итак, личность нашего пользователя установлена. Можно ли теперь ему предоставить неограниченный доступ к машине? К сожалению, есть еще одно, к тому же наиболее важное ограничение — это бюджет. Как было объяснено выше, бюджет на все ресурсы, выделяемые пользователю, устанавливается директором вычислительного центра, и он проверяется каждый раз, когда пользователь запрашивает машину.

После того как пользователь прошел все проверки, он может делать со своего пульта фактически все, что он мог бы сделать при подготовке работы для ее выполнения оператором. Он может, например, напечатать на своем телетайпе описание работы, используя стандартный язык описания работ. Он может создать новую программу, вводя команды строчка за строчкой с телетайпа, а затем выполнить ее или записать ее в архив. Равным образом он может вызвать существующую программу, хранящуюся в архиве, напечатав имя соответствующего файла. Можно вносить изменения в программу на исходном языке при помощи Контекстного Редактора, который найдет любую требующуюся команду и произведет над ней нужные изменения.

Когда дело доходит до счета по программе, интегральный подход, положенный в основу GEORGE 3 и объединяющий пакетную обработку с мультидоступом, открывает перед программистом возможности использовать разнообразные и гибкие приемы для отладки и совершенствования программы. Он может, например, выдать инструкции, чтобы некоторые вводные устройства моделировались (были заменены) его телетайпом. Если, например, пользователь дает команду `ONLINE * CR0`, то для GEORGE 3 это будет указанием, что в выполняемой программе устройство чтения с перфокарт должно быть заменено телетайпом. В результате каждый раз, когда его программа выдает команду чтения перфокарты, вместо того, чтобы запустить устройство чтения карт, система будет печатать на телетайпе сообщение, в ответ на которое пользователь должен ответить набором на клавиатуре телетайпа тех

80 символов, которые должны были бы вводиться с перфокарты. Или же пользователь может выдать с телетайпа команду INPUT, по которой вся последующая, набираемая на телетайпе информация будет записываться в файл до тех пор, пока не будет набран признак конца файла (обычно в GEORGE 3 это ****). Этот файл может быть впоследствии подсоединен к программе с помощью команды ASSIGN, снова выданной с телетайпа после того, как программа загружена.

Вывод точно так же может быть направлен на телетайп с помощью команды ONLINE*LP0 (или *CR0). При этом необходимо иметь возможность прервать такой вывод, иначе при заиклировании в программе можно «затопить» телетайп потоком бессмысленных сообщений, для анализа которых нужно совсем выйти из программы. Поэтому пользователь пульта обеспечивается прерывающим ключом, который позволяет ему возобновить контакт с операционной системой, а не с программой. Эту возможность можно использовать для того, чтобы проверить программу, внести в нее некоторые изменения и затем ее продолжить. Эта же возможность полезна в том случае, когда пользователь приходит к заключению, что он достиг в диалоговом режиме всего, что его интересовало, и хотел бы в дальнейшем выполнить эту работу как фоновую, а сам перейти к другой работе. В таком случае он может запустить данную программу и затем дать команду DISCONNECT, в которой при желании может указать имя новой работы; первая работа теперь выводится из-под контроля телетайпа, а новая работа заменяет ее.

Когда сеанс работы с телетайпом закончен, пользователь должен сообщить о конце работы, отключившись от системы. При этом может случиться, что пользователь не хочет терять свою программу, находящуюся в памяти. Если это так, то он может дать команду SAVE, которая заставит систему записать программу как сохраняемый файл: потом пользователь сможет восстановить программу снова с помощью команды RESUME. Когда он в конце концов печатает LOGOUT (конец сеанса), его работа вычеркивается из списка активных работ системы, просматривается мониторный файл и на телетайпе печатается необходимая пользователю информация об употреблении системы в процессе работы, расходы, которые он понес, и остаток бюджета.

Дальнейшее развитие системы GEORGE 3 расширяет возможности мультидоступа. Вместо телетайпов в качестве терминальных пультов могут подключаться визуальные дисплеи, а также увеличивается количество информации, доступной

пользователю за терминалом. Например, один пользователь может запомнить в системе сообщение, предназначенное для другого пользователя, и это сообщение будет выдано на пульт, как только этот другой пользователь начнет сеанс работы с машиной. Но самое важное усовершенствование относится к области программного обеспечения для мультидоступа — в частности к разговорным компиляторам.

5.6. Разговорные компиляторы

Пользователь терминального пульта может использовать стандартный компилятор, чтобы протранслировать программу на исходном языке, получить готовую программу и впоследствии ее выполнить. Ясно, однако, что какое-либо тестирование при этом невозможно до тех пор, пока его программа не закончена целиком: более того, количество диагностической информации, которую он может получить, ограничено синтаксическим анализом каждой строки по мере ее ввода. Для диалоговой работы более удобно интерпретировать каждую исходную команду, как только она вводится, и выполнять ее немедленно. Затем пользователь может тщательно исследовать результат, прежде чем выдать следующую команду. Такими возможностями обеспечивают пользователя несколько «разговорных» компиляторов с языков, часто напоминающих язык JOSS, созданный в Rand Corporation. Вариант такого языка, разработанный фирмой ICL для машин серии 1900 под именем JEAN, описан в гл. 8.

Будущие компиляторы, видимо, будут изготавливаться в нескольких различных вариантах. Один вариант обеспечит быструю компиляцию за счет эффективности готовой программы. Это даст возможность пользователю за терминалом очень быстро отладить программу и, когда программа отлажена, перекомпилировать ее, используя другой компилятор, предназначенный для того, чтобы выдать наиболее эффективную готовую программу, не считаясь со временем трансляции. Эффективность может быть достигнута при помощи промежуточного языка между исходной и готовой программой, который мог бы выполняться в режиме интерпретации. Развитие этой идеи состоит в использовании для компиляции в режиме мультидоступа машины-спутника, работающей с такими промежуточными интерпретируемыми программами, и затем в пересылке их по линиям связи в мощный центральный процессор, который производит вторую стадию трансляции в оптимальную готовую программу.

Несомненно, что мультидоступ предлагает захватывающие возможности расширения связи человек — машина, и придет время, когда в каждом доме будет стоять свой терминал, как сегодня в каждом доме стоит свой телефон.

5.7. Планирование

Нам осталось рассмотреть вопросы планирования. Мы видели, что операционная система, как правило, выполняет большое количество работ, предложенных ей пользователями, и что если она обладает возможностями мультидоступа, то она должна быть в состоянии приняться за некоторые работы без промедления, чтобы дать пользователям терминала ответ в короткое время. Следовательно, в системе должен каким-то образом решаться вопрос о том, какие работы выполнять в любой заданный момент. Этот вопрос решается с помощью *планировщика*.

В планировании нужно рассматривать два уровня: стратегическое и тактическое планирование. Под последним подразумевается управление машиной каждую миллисекунду. Понимание этого требует детального рассмотрения внутренней структуры системы, которая будет изучена в следующей главе. Под стратегией подразумевается общий отбор работ, какой мог бы быть выполнен человеком-оператором, планирующим свою работу на предстоящую смену. В действительности в системе GEORGE 3 стратегией и тактикой управляют два различных модуля — Планировщик Верхнего Уровня (Стратегический Планировщик) и Планировщик Нижнего Уровня (Тактический Планировщик).

Факторы, принимаемые на рассмотрение в Планировщике Верхнего Уровня, в свою очередь могут быть разделены на две категории: те, которые относятся к конфигурации машины, и те, которые относятся к выполняемым программам. Первые включают такие факторы, как:

- (а) внутренняя память машины, доступная программам;
- (б) соотношение между фоновой работой и работой в режиме мультидоступа;
- (с) максимальное число работ, выполняемых системой одновременно;
- (д) скорость пересылки между оперативной и вспомогательной памятью;
- (е) периферийные устройства, имеющиеся в наличии.

Некоторые из этих параметров должны быть установлены директором вычислительного центра, который должен осуществлять балансирование между обеспечением быстрого

сервиса для пользователей терминалов и гарантированием того, что фоновые работы не будут чрезмерно задержаны.

Когда речь идет об отдельных (индивидуальных) программах, факторы, которые должны быть приняты во внимание, включают:

- (а) срочность работы, обозначаемую буквами А—Z для GEORGE 3;
- (b) время, к которому работа должна быть закончена («крайний срок»);
- (с) ориентировочное время работы программы;
- (d) наличие необходимых файлов и внешних устройств, работающих в режиме on-line;
- (е) размер программы;
- (f) характер программы (счетная или коммерческая).

Некоторые программы, работающие в реальном времени, должны постоянно находиться во внутренней памяти, т. е. они никогда не должны вытесняться во вспомогательную память.

В GEORGE 3 существуют команды, с помощью которых информация по всем перечисленным пунктам может быть сообщена операционной системе, причем некоторые параметры устанавливаются заранее и являются общими, а часть их указывается в описании работы. Точный алгоритм выбора программ по заданным параметрам зачастую изменяется от одной установки к другой: машины, обеспечивающие главным образом обсчет разнообразных работ для инженерной лаборатории, будут иметь потребности, отличающиеся от потребностей коммерческой установки, предназначенной преимущественно для пакетной обработки более однородных работ. Поэтому внутри системы GEORGE 3 для пользователей предусмотрены возможности написать свой собственный Планировщик Верхнего Уровня и вставить его в систему вместо стандартного модуля.

Независимо от того, написан ли Планировщик Верхнего Уровня пользователем или поставлен фирмой, конечным результатом его работы для каждой программы в системе GEORGE 3 будет некоторый *индекс*. Это — число, которое указывает относительные размеры следующего отрезка времени (time slice), который разрешено занимать данной программе. Этот расчет выполняется приблизительно каждую минуту, и полученный в результате список программ со значениями их индексов передается Планировщику Нижнего Уровня. Последний работает более часто (примерно один раз в секунду) и назначает каждой программе *временной квант* (time slot) в зависимости от размера программы и общего числа программ, находящихся в активном состоянии. Этот модуль вычисляет также *идеальное время ожидания*, основываясь

на значении индекса и продолжительности последнего интервала работы программы и учитывая временной квант и количество фоновых работ, находящихся в состоянии ожидания. В любой заданный момент времени следующей будет выполняться та программа, которая имеет наибольшее несоответствие между идеальным временем ожидания и тем временем, в течение которого она фактически находится в состоянии ожидания.

В конце концов программы для выполнения «подключаются» к супервизору, так что на самом нижнем уровне управления используется обычное мультипрограммирование. В следующей главе вопрос выполнения программ мы рассмотрим более детально.

ВНУТРЕННЕЕ ФУНКЦИОНИРОВАНИЕ СИСТЕМЫ GEORGE 3

Тим Голдингом

Описав основные новые возможности системы GEORGE 3, Тим Голдингом теперь рассматривает ее работу более подробно. Он описывает организацию внутренней памяти и метод цепочек (ссылок), с помощью которого достигается исключительная гибкость системы. Далее он рассматривает одну из наиболее важных цепочек, которая содержит информацию о действиях, и тем самым подводит нас к пониманию способа, с помощью которого осуществляется общее управление различными процессами в операционной системе. В заключение в качестве иллюстрации он подробно рассматривает одну реальную ситуацию.

6.1. Основные принципы

Предыдущая глава была посвящена описанию системы GEORGE 3 с точки зрения пользователя. Теперь мы рассмотрим внутреннее функционирование системы.

Можно выделить некоторые основные принципы. Во-первых, определение местоположения интерфейса. Мы уже видели, что система GEORGE является независимой, самостоятельной частью математического обеспечения в таком же смысле, в каком самостоятельны компиляторы и пакеты прикладных программ. Они не встроены в операционную систему 1900, как это сделано в некоторых других системах, а написаны для того же интерфейса, что и программы пользователей.

С другой стороны, интерфейс занимает промежуточное положение между операционной системой и супервизором. Поэтому модули, необходимые для манипулирования с любым возможным набором внешних устройств, могут быть включены в супервизор, с тем чтобы один и тот же вариант GEORGE 3 мог сам работать на различных моделях серии 1900.

Во-вторых, на ранних этапах разработки было принято решение о том, что GEORGE 3 не должен прерывать своей работы, а это значит, что, когда поступает внешний запрос, GEORGE не обслуживает его сразу, а лишь регистрирует, обрабатывая его в следующий подходящий момент времени. Далее мы увидим, что эти два принципа требуют наличия эффективной связи между супервизором и системой GEORGE, а также механизма выделения программам временных квантов сравнительно короткой продолжительности, чтобы сигналы от внешних устройств обрабатывались без задержки.

Третья задача, поставленная при проектировании, состояла в том, чтобы создать систему управления внутренней памятью, гибкую настолько, чтобы не было потерь эффективности при выделении части памяти для специальных целей.

Наконец, было принято, что система GEORGE должна быть запрограммирована в виде набора простых процедур, а это означает, что программы не должны изменяться во время их исполнения, так чтобы одни и те же программы можно было использовать более чем в одном процессе.

6.2. Организация внутренней памяти

Рисунок 1 иллюстрирует распределение внутренней памяти процессора в системе GEORGE 3.

Как обычно, в машинах серии 1900 ячейки памяти с наименьшими номерами отводятся под супервизор, но в данном

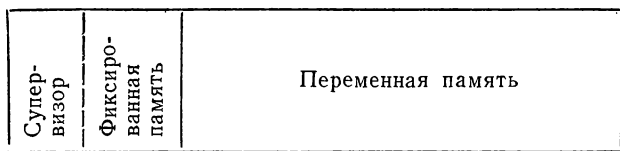


Рис. 1.

примере граница супервизора обозначена пунктирной линией, поскольку существует связь между супервизором и операционной системой и поэтому разделяющая их линия менее четкая, чем между супервизором и программой пользователя. Супервизор для GEORGE 3 написан специально для использования его с данной операционной системой и не может быть употреблен в другой ситуации. И наоборот, GEORGE 3 не может работать ни с каким другим супервизором.

Область, помеченная как «фиксированная память», содержит ту часть GEORGE 3, которая должна постоянно находиться во внутренней памяти. Таким образом, супервизор и фиксированная память, вместе взятые, соответствуют стандартному супервизору, и фактически требуемый для них объем памяти (примерно 12 000 слов) не превышает объема стандартного супервизора для большой конфигурации машины.

Остальная часть памяти, обозначенная на рис. 1 как «переменная», занимается программами пользователей и системой GEORGE по мере необходимости. Сама система GEORGE 3 состоит примерно из 100 000 слов в командах машины: она поделена на главы различной величины (как

правило, менее 1000 слов). В любой заданный момент времени в переменной части будет стоять некоторое число таких глав, содержащих программы, требующиеся для выполнения текущих операций, а также несколько программ пользователей. Таким образом, GEORGE можно представить себе как перекрывающийся сверх-супервизор (overlaid super-executive). Замена глав в памяти осуществляется Сменщиком Глав, который всегда находится в фиксированной памяти.

6.3. Цепочки

Базовая система мультипрограммирования для машин серии 1900 предусматривает работу с перемещаемыми программами. Так что всякий раз, когда какая-нибудь программа уничтожается, остающиеся программы сдвигаются в памяти таким образом, чтобы все свободное место представляло единый массив. Это показано на рис. 2, где вторая

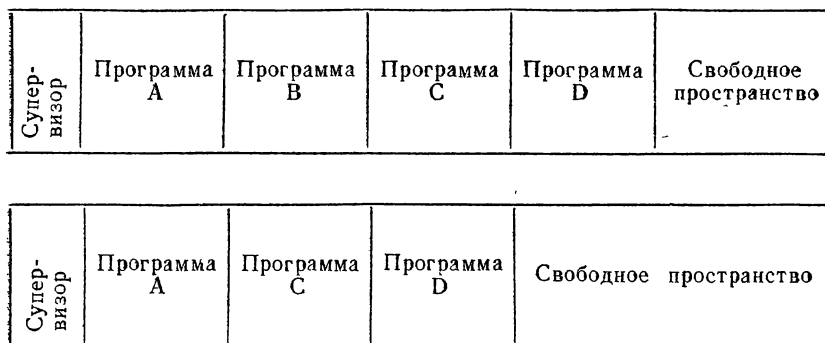


Рис. 2.

диаграмма иллюстрирует распределение памяти после того, как программа В была уничтожена.

Такая система идеальна для машин с пакетной обработкой, с ограниченным объемом внутренней памяти, но она была бы неприемлемой для машин с большой памятью, работающих в режиме мультидоступа, где программы сменяются очень часто и где поэтому потребовалось бы множество перемещений. Более того, было бы нецелесообразно сдвигать таким же способом главы GEORGE. Поэтому принятой здесь системой управления памятью являются *цепочки*, как показано на рис. 3.

Блоки могут находиться в любом месте памяти¹⁾; каждый блок имеет в своих первых двух ячейках указатели *вперед* — на следующий блок в цепочке и *назад* — на предыдущий блок

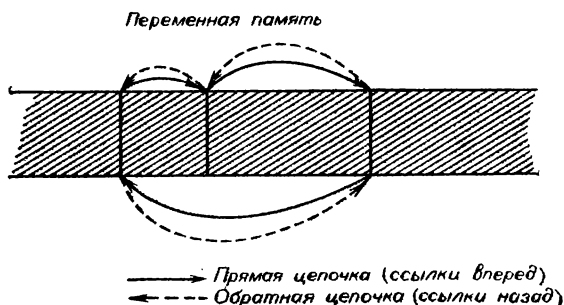


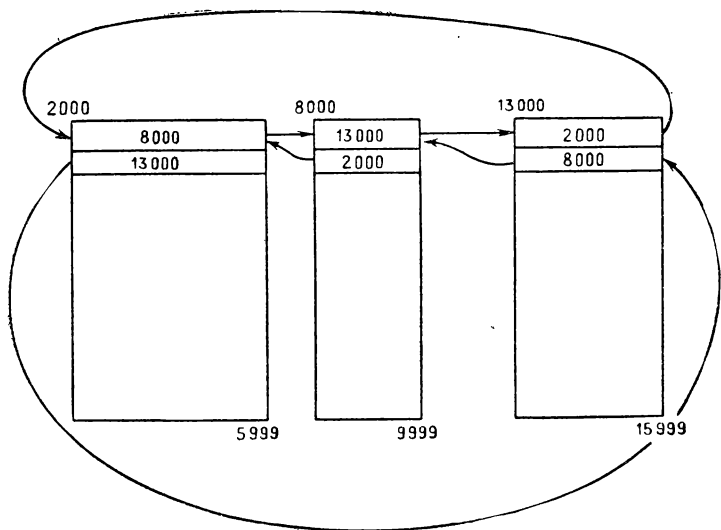
Рис. 3.

(последний блок ссылается на первый и наоборот, для того чтобы замкнуть цепочку). В GEORGE 3 имеется большое число таких цепочек; все вместе (включая и цепочку свободной памяти, которая соединяет все незанятые блоки) они охватывают всю внутреннюю память. Рис. 4 иллюстрирует только что изложенный принцип в терминах адресов; рис. 5 демонстрирует ту простоту, с которой в цепочку может быть добавлен новый блок: эта операция не требует никакого передвижения в памяти, а сводится только к изменению пары указателей.

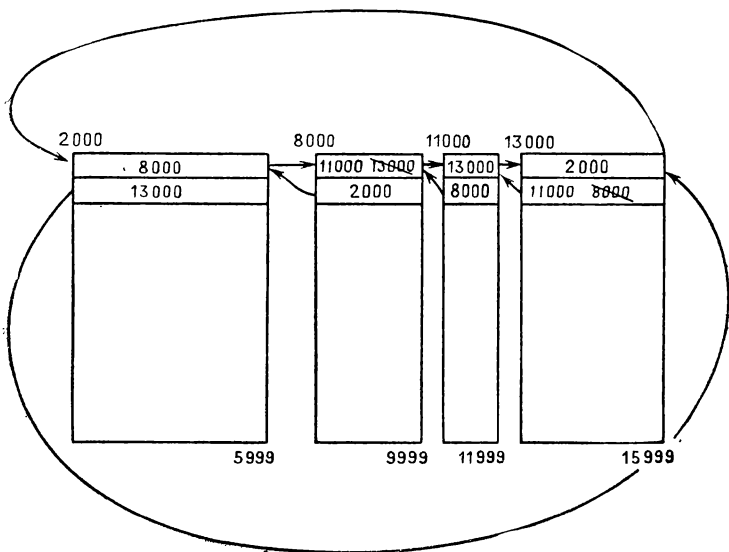
Одной из частей GEORGE 3, которая хранится в фиксированной памяти, является Система Распределения Памяти, которая исследует цепочку свободной памяти всякий раз, когда требуется память либо для главы GEORGE, либо для программы пользователя. В фиксированной памяти, кроме того, расположены базовые пары всех цепочек, содержащие начальные указатели вперед и назад, которые обеспечивают вход в цепочки.

Поняв технику построения цепочек, мы можем теперь рассмотреть одну из наиболее важных цепочек системы.

¹⁾ Диаграмма не совсем точна, так как всякая память одномерна и один блок должен занимать, конечно, последовательные номера ячеек.



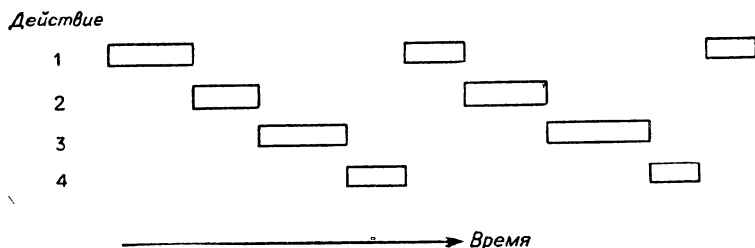
Р и с. 4.



Р и с. 5.

6.4. Цепочка действий

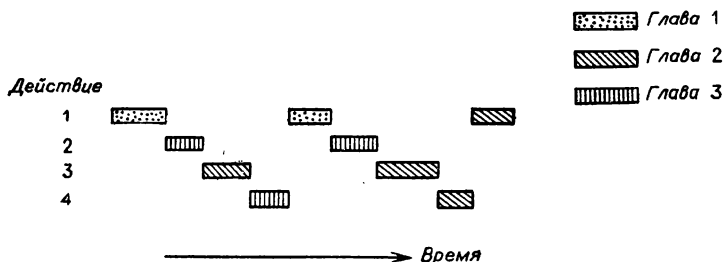
От системы GEORGE 3 требуется выполнение многих функций: загрузки программ, управление внешними пересылками и т. д. Каждая функция может быть названа *действием* (activity). Возможно, что в некоторый момент времени GEORGE будет выполнять несколько таких действий, и так



Р и с. 6.

как действия часто задерживаются, например в ожидании завершения пересылки, то они могут обычным способом выполняться в мультипрограммном режиме. Это проиллюстрировано на рис. 6.

Фактические команды (программы) для выполнения любого такого действия содержатся в главах GEORGE: некото-



Р и с. 7.

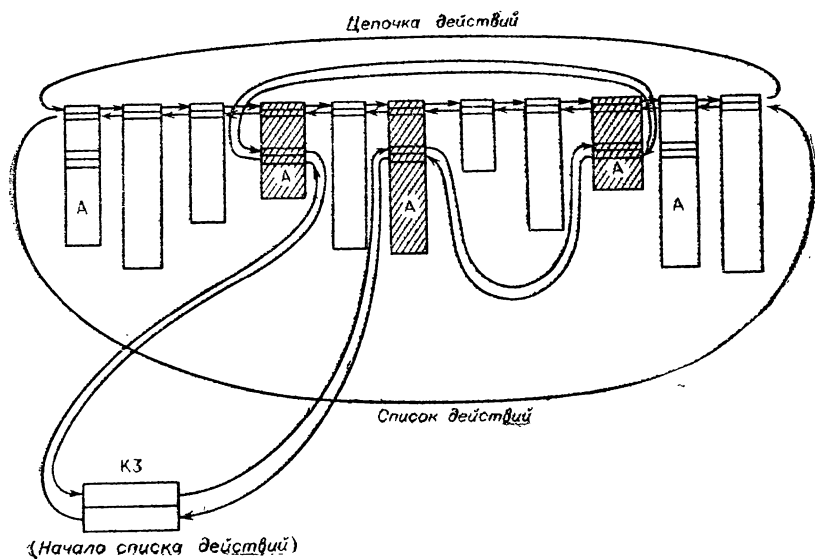
рым действиям может требоваться несколько глав. Но, как отмечалось ранее, эти главы *просты* в том смысле, что каждая глава может использоваться несколькими действиями одновременно.

Таким образом, мы имеем ситуацию, изображенную на рис. 7, которая показывает, что одна и та же глава может быть использована несколькими действиями и в то же время каждое действие может использовать несколько глав.

Хотя главы сами по себе являются «простыми» (неизменяемыми), конечно, должна иметься информация, связанная

Указатели цепочки действий		Вперед Назад	FPTR BPTR
Размер			
Тип			
Указатели списка действий		Вперед Назад	CHAINADD BACKCHAIN
Адрес соответствующей главы на барабане			ALINK 1
Длина	Точка входа		ALINK 2

Р и с. 8.



Р и с. 9.

с каждым действием. Она содержится в *блоке действия*; все блоки действий связаны вместе в цепочку, называемую *цепочкой действий*. Таким образом, действие — это абстрактное понятие, обозначающее определенный процесс. Переменная информация, связанная с ним, хранится в блоке действия, содержащемся в цепочке действий, а команды, выполняющие это действие, — в одной или нескольких главах. Блок действия в свою очередь может иметь несколько связанных с ним блоков данных.

Схема блока действия показана на рис. 8, из которого видно, что блок содержит ссылку на связанную с ним главу и, обратно, первое слово в каждой главе хранит текущее обслуживаемое действие.

Мы видели, как цепочка действий связывает во внутренней памяти все блоки действий в любой момент времени. Теперь мы должны рассмотреть механизм планирования, который обеспечивает связь с супервизором. В дополнение к цепочке действий имеется также *список действий*. По существу это цепочка со ссылками в обоих направлениях, аналогичная цепочке действий, но отличающаяся от нее тем, что она содержит только те блоки действий, которые требуют обслуживания. Всякий раз, когда к супервизору приходит сигнал прерывания, например от периферийного устройства, супервизор идентифицирует блок действия, который может обработать это прерывание, и добавляет его в начало списка действий. Мы, таким образом, имеем ситуацию, представленную на рис. 9, где *цепочка действий* содержит все блоки действий, находящиеся в данный момент в памяти, а *список действий* содержит подмножество блоков действий, расположенных в порядке их выполнения.

6.5. Координатор

Механизм, который отбирает действия из списка для их выполнения, называется Координатором. Его упрощенная блок-схема показана на рис. 10. Ранее мы говорили, что система GEORGE не должна прерывать своей работы, но, конечно, существенно, чтобы внешние события получали ответы своевременно, и поэтому вход в Координатор происходит приблизительно через каждые 100 команд. Таким образом, система GEORGE не прерывается извне, но через частые интервалы просматривают ситуацию для того, чтобы выяснить, что нужно обслужить в первую очередь. На упрощенной схеме показано, как берется верхний элемент из списка действий, а текущий, выполняемый элемент помещается в конец списка.

Этим способом достигается работа в режиме, изображенном на рис. 6.

Вход 1 («Обычный») соответствует процедуре, которая будет выполнена после того, как некоторое действие, использовав до конца свой временной квант, прервалось, но еще не закончилось. Адрес очередной команды, которая должна была выполняться, помещается в слово ALINK 2, указывающее на точку входа (рис. 8), так что в свое время, когда через

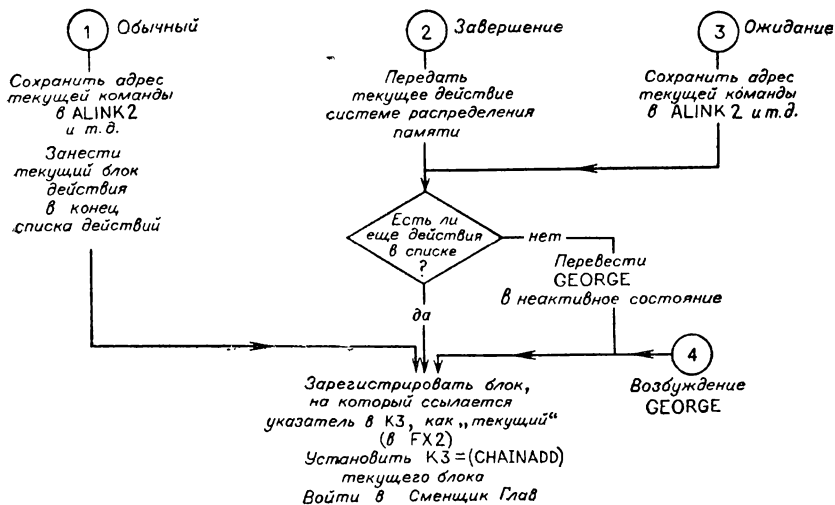


Рис. 10.

круг снова подойдет очередь, выполнение данного действия начнется с этой команды.

Блок, на который указывает начальная ссылка списка действий, теперь становится текущим действием, а в начальную ссылку списка действий заносится содержимое ячейки CHAINADD, т. е. указание на блок, который был вторым в списке. Таким образом, всякий раз, когда истекает временной квант некоторого действия, происходит вход в следующий блок списка.

Может случиться, однако, что действие в действительности закончено. В этом случае мы попадаем на вход 2 блок-схемы («Завершение») и отдаем место, занимавшееся этим действием, системе распределения памяти, так чтобы оно могло быть использовано для будущих действий. Затем мы проверяем, есть ли в списке другие действия, ожидающие выполнения. Если есть, то входим в них, как и прежде; если же

нет ни одного действия, то система GEORGE может быть переведена в неактивное состояние.

Третий вход («Ожидание») применяется в тех случаях, когда действие еще не выполнило всех своих работ и еще не истек его временной квант, но оно не может быть продолжено, потому что ожидает окончания некоторого действия, в большинстве случаев — пересылки информации либо на периферийное устройство, либо с устройства в память машины. В таком случае это действие помещается в конец очереди и вход в него будет сделан Координатором, когда пересылка завершится.

6.6. Пример

Теперь, после того как изложены основные принципы работы системы GEORGE, рассмотрим некоторую реальную ситуацию, а именно обработку простого описания работы для загрузки программы и передачи на нее управления (входа в нее). Это иллюстрируется рис. 11.

Оператор закладывает в устройство ввода карт карты, содержащие описание работы, вслед за которым идут программа и данные, и нажимает кнопку ENGAGE (ЗАНЯТО). В результате возникает прерывание, которое обнаруживается системным Начальным Действием, постоянно присутствующим в памяти. Это действие внутри своего блока содержит область из трех слов под названием Область Событий с Периферийными Устройствами, в которой на каждое периферийное устройство, подключенное к процессору, отведен один двоичный разряд. Таким образом, когда на читающем устройстве нажимается кнопка ENGAGE, возбуждается разряд, соответствующий этому конкретному устройству. Начальное Действие определяет, что прерывание, из-за которого было возбуждено это действие, порождено периферийным устройством, и вызывает главу системы GEORGE, ответственную за обработку событий с периферийными устройствами. Эта глава в свою очередь анализирует Область Событий с Периферийными Устройствами, чтобы определить, какое же из устройств вызвало прерывание. Установив, что это было устройство ввода карт, она возбуждает следующее действие — действие Обработки Управляющих Команд.

Это действие вызывает соответствующую главу из вспомогательной памяти для того, чтобы осуществить чтение образов карт. Для этой цели нужна буферная область. Глава Обработки Управляющих Команд запрашивает ее через систему распределения оперативной памяти и помещает адрес полученной области в блок Обработки Управляющих Команд.

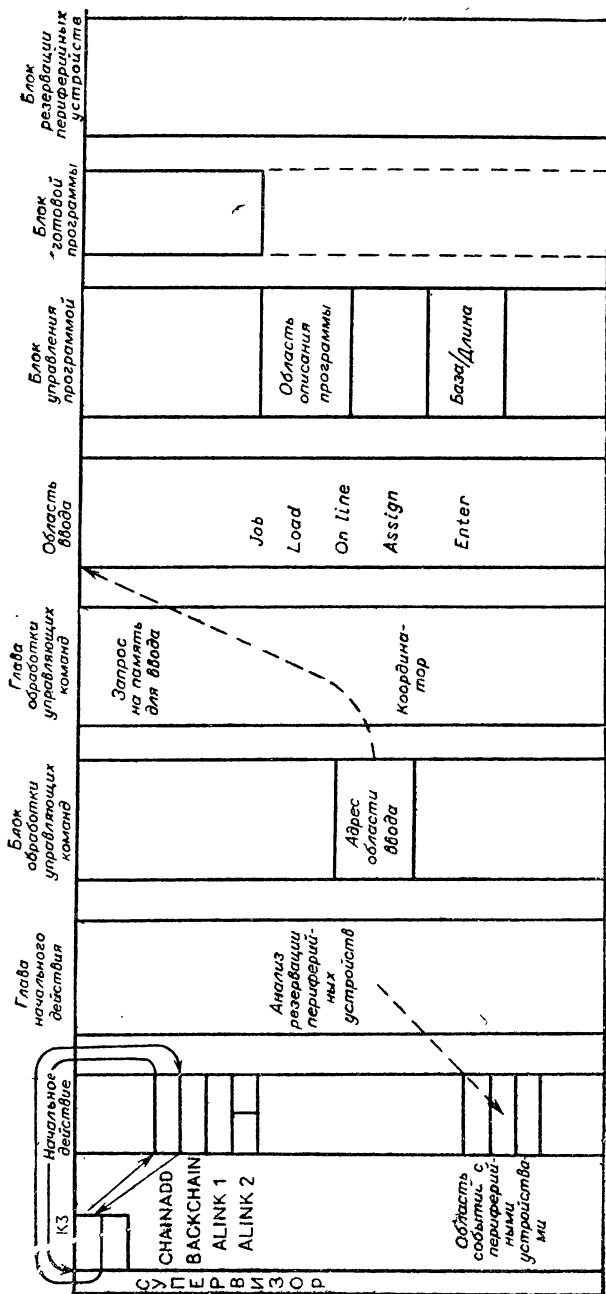


Рис. 11.

После этого глава может выдать команду ввода, ссылаясь на полученный адрес.

Пока выполняется команда ввода, действие находится в состоянии ожидания. По этой причине для каждого периферийного устройства заводится свое действие. На первый взгляд может показаться, что это необоснованное усложнение, но оно оказывается необходимым, если периферийные устройства используются в режиме разделения времени. Карты поступают в 80-символьную буферную область, откуда передаются в *файл описания работы*, образованный в архиве файлов. Этот файл впоследствии анализируется действием Обработки Управляющих Команд.

Очевидно, первой управляющей командой в нашем примере будет команда LOAD (Загрузить программу). Для того чтобы ее выполнить, нужны еще блоки следующих действий: Блок Управления Программой, Блок Готовой Программы, содержащий саму выполняемую программу, и Блок Резервации Периферийных Устройств.

Блок Управления Программой содержит информацию о программе, в частности ее базу и длину, которые определяют для супервизора область памяти, занимаемую этой программой. Тем самым супервизор получает информацию, необходимую для выполнения программы.

Блок Готовой Программы первоначально имеет такой размер, чтобы вместить только около 100 слов программы, которые загружаются командой LOAD. Вслед за этой командой обычно идут команды, которые описывают нужные программе периферийные устройства. Последние могут подключаться непосредственно к программе, и в этом случае задается команда ONLINE, или же они могут моделироваться при помощи архива файлов таким образом, что в момент выдачи программой команды чтения карты эта команда перехватывается системой GEORGE и превращается в команду чтения образа карты из файла, который был создан в архиве заранее. В этом случае задается управляющая команда ASSIGN.

Информация о распределении устройств записывается в двух ячейках в Блоке Резервации Периферийных Устройств, а для устройств, используемых в режиме off-line, в них содержатся указатели на Блок Действия с Устройством off-line.

Теперь можно приступить к выполнению программы. Для этого должна быть сделана ссылка на эту программу в списке работ программы, известной под названием Планировщик Нижнего Уровня.

Вход в Планировщик Нижнего Уровня осуществляется через фиксированные интервалы времени. Располагая всей информацией о доступной памяти, он исследует свойства про-

грамм, содержащихся в списке, и решает, какая из работ должна выполняться. Детальная информация об этой работе в виде адресов Блоков Управления Программой передается в супервизор тем же путем, что и при обычном мультипрограммном супервизоре.

Мы рассмотрели основные принципы взаимодействия системы GEORGE 3 с супервизором и выполнения в режиме разделения времени большого числа действий, обслуживаемых по существу циклически. Один вопрос, который был затронут в связи с вводом описания работы, требует дальнейшего уточнения, а именно методика взаимодействия с архивом файлов.

Было сказано, что карты читаются в 80-символьный буфер. Возможно использование большего буфера длиной в 512 слов, в который передаются полученные записи. Если происходит переполнение этого буфера, то его содержание переписывается во вспомогательную память.

С каждым действием связан Блок Управления Файлами, который содержит информацию об этих файлах. Этот блок включает имя каждого файла вместе со списком адресов во вспомогательной памяти каждого из блоков по 512 слов, которые, собственно, и образуют этот файл. Таким образом, когда буфер заполнится, он может быть записан в любое свободное место вспомогательной памяти.

Однако, касаясь деталей архива, мы забегаем вперед, поскольку архив является предметом следующей главы.

Как отмечалось в гл. 5, одним из наиболее важных аспектов системы GEORGE 3 является наличие архива файлов. Ангус Битти рассматривает его более подробно. Он перечисляет различные типы файлов, хранящихся в архиве, раскрывает иерархичность структуры архива и связь иерархичности с проблемой учета. Затем он обсуждает понятие секретности, очень важное при работе в режиме мультидоступа, и, наконец, описывает возможности дампинга, решающего как проблему переполнения файлов, так и проблему восстановления архива после сбоев системы.

7.1. Общие соображения

Как мы видели в гл. 6, имеются два метода выполнения программ под контролем операционной системы GEORGE 3: выполнение под непосредственным управлением с терминального пульта и выполнение программы как фоновой работы. Это деление определяет способы инициирования команд программы и контроля за их выполнением. Необходимо также различать два способа использования данных программой любого типа. Имеются две разновидности файлов: *on-line файлы* и *архивные файлы*. Наиболее явное внешнее различие между ними состоит в том, что пользователям *on-line* файлов кажется (и это в действительности так и есть), что в момент обращения к файлу при выполнении готовой программы файл располагается на конкретном периферийном устройстве, например на перфоленке или диске. С другой стороны, данные из архивного файла, хотя для готовой программы они и кажутся расположенными во вспомогательной памяти, в момент выполнения команды обращения к этим данным из программы оказываются частично в оперативной памяти, а частично во вспомогательной памяти с прямым доступом. С некоторыми ограничениями, которые мы сформулируем позже в этой главе, пользователь обращается к такому файлу точно так же, как он обращался бы к *on-line* файлу.

Специфика реализации состоит в том, что всякий раз, когда программа пользователя открывает файл или обращается к файлу любого типа, супервизор делает соответствующее сообщение операционной системе. Далее уже сама операционная система имеет дело со ссылками на архивный файл. Например, если готовая программа заносит блок данных для перфоленки в файл архива, то операционная система

пересылает эти выводные данные из буфера программы пользователя в другой (системный) буфер в памяти, а затем в удобные для нее моменты времени она переписывает блоки данных из системного буфера во вспомогательную память с прямым доступом, используемую для хранения архивных файлов.

Преимущества использования файлов этого типа во всех допустимых случаях очевидны. Система от этого только выигрывает, так как она не страдает от традиционного узкого места мультипрограммных систем — отсутствия достаточного количества периферийных устройств, подсоединенных к данной установке. Даже на больших установках, скажем с двумя устройствами ввода карт и двумя печатающими устройствами, было невозможно выполнение в мультипрограммном режиме более чем двух программ, если каждая из них нуждалась и в устройстве ввода с карт и в печатающем устройстве. Таким образом, такие системы без архива страдали либо из-за неполного использования центрального процессора, либо из-за необходимости видоизменять естественные методы отладки и счета по программе, для того чтобы минимизировать использование периферийных устройств. Но при эксплуатации архива программист получает возможность наилучшим образом использовать периферийные устройства, оставив операционной системе решать все проблемы разделения времени, планирования (диспетчеризации) и организации хранения файлов данных.

Для простоты в оставшейся части этой главы словом «файл» будем обозначать «файл из архива GEORGE 3».

Нужно заметить, что, кроме уже описанных типов файлов, в архиве хранятся файлы, содержащие программы, к которым обращается операционная система или система программного обеспечения, например загрузчик программ. Это удобно и операционной системе, и системе программного обеспечения, а благодаря принятой системе имен (см. ниже) системный программист не должен волноваться о том, что данные его программы используют архив операционной системы совместно с другими файлами (такими, как библиотечный файл), о которых он может ничего не знать.

Другой пример системных файлов, также занимающих архив, представляют собой временные файлы, которые создает операционная система в том случае, когда пользователь в описании работы пишет команду OFFLINE (см. гл. 5). При этом для вводного файла операционная система вводит соответствующий файл целиком и записывает его во временный архивный файл, который в свою очередь готов выдать одну запись по каждому запросу готовой программы. Область в архиве, занятая таким файлом, может быть перераспределена

операционной системой для других целей после того, как программа закрыла этот файл или сама была вычеркнута из списка активных программ.

Операционная система может также перемещать файл в пределах архива, если это вызвано требованиями перераспределения памяти, внутренней или вспомогательной, поступившими от других программ или от самой операционной системы.

7.2. Типы файлов

7.2.1. Последовательный файл

Последовательным называется файл, подобный off-line файлу на перфоленоте или перфокартах. Пользователь может считать *следующую* запись из последовательного вводного файла или записать *следующую* запись в последовательный выводной файл. Формат последовательного файла фактически совпадает либо с форматом перфокарточного, либо с форматом перфоленточного файла, и для того чтобы обратиться к файлу, основная программа должна выполнить последовательность команд чтения или записи в файл.

С последовательным файлом можно обращаться так же, как с магнитной лентой, при условии, что над ним производятся только следующие операции: открыть, прочитать, записать и закрыть.

7.2.2. Файл с прямым доступом

Файл этого типа аналогичен дисковому файлу. Это означает, что порядок обращения к данным в файле может быть совершенно произвольным: характерно, что последовательные команды обращения могут записывать и считывать произвольное количество слов по любому адресу в файле.

7.2.3 Файл связи

Любой файл с прямым доступом может быть открыт несколькими программами одновременно только для чтения. Однако файл связи — это такой последовательный файл, который открывается одной программой для чтения и в то же самое время другой — для записи. Типичным случаем использования этого файла является тот случай, когда одна программа должна добавлять данные в конец файла, пока другая программа с каким-то запаздыванием читает записи этого файла. Если при этом читающая программа забежит вперед

и попытается прочесть из файла несуществующие данные, то операционная система поставит запросы на чтение в очередь до тех пор, пока записывающая программа не добавит новые данные. Когда на практике встречаются такие случаи, поступающие новые данные передаются операционной системой сразу во внутреннюю память для второй программы.

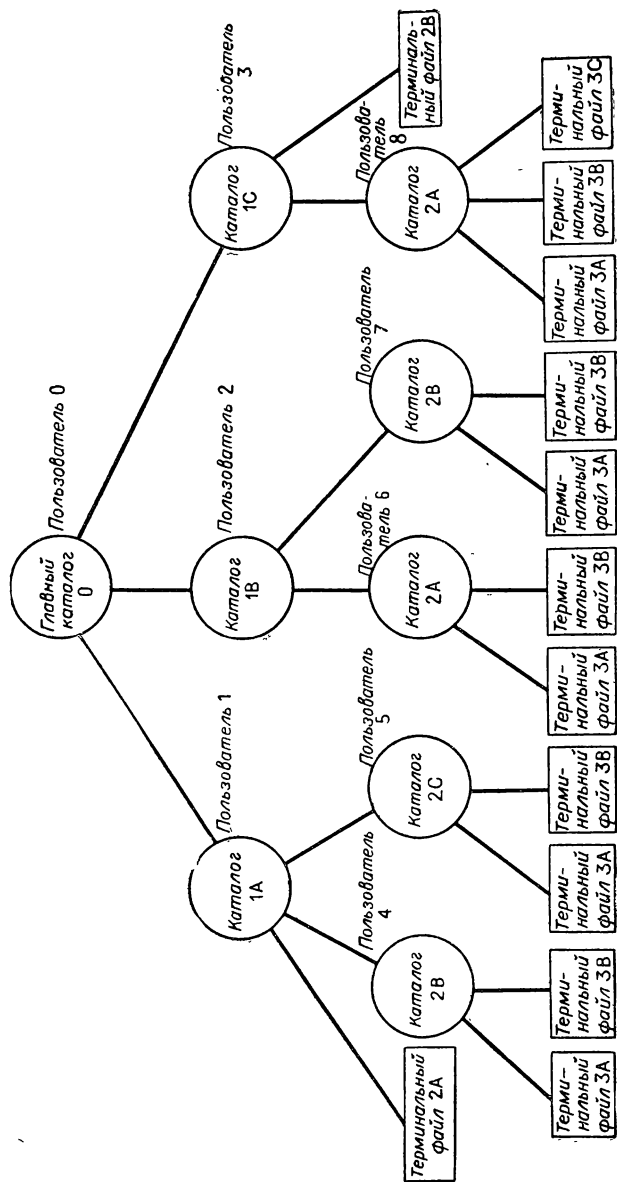
Если же пользователь записывает данные в последовательный файл, не являющийся файлом связи, то операционная система всегда собирает пакет записей и отправляет их во вспомогательную память. Когда позже пользователь читает такой файл, он фактически заставляет операционную систему выполнить серию команд чтения из вспомогательной памяти. По существу это же происходит в программах пользователя, работающих с файлом связи, если процесс чтения отстает от процесса записи. Однако в том случае, когда считывающая программа воспринимает данные сразу, как только они поступили от программы записи, операции чтения из вспомогательной памяти оказываются ненужными. Следовательно, используя этот метод, можно сэкономить время на внешних пересылках.

Файлы связи дают возможность пользователю создавать свою программную систему не из подпрограмм, жестко связанных друг с другом, а из автономных, независимо скомпилированных программ.

7.3. Структура архива

Архив, как он реализован в системе GEORGE 3, представляет собой единый логический комплекс, содержащий, как упоминалось выше, и системные и пользовательские файлы. Файлы образуют древовидную иерархическую структуру. Узлы дерева могут быть либо *узлами каталогов* (административными), либо узлами файлов с действительными данными (*оконечными*, или *терминальными*, *узлами*). Пользователь может не знать, какова структура дерева выше того уровня, на котором расположены его текущие файлы, если, конечно, этого не требует специфика его задачи: когда пишется программа, которой не нужно работать с иерархическими структурами, программист может использовать только часть возможностей, описанных ниже. Пример структуры архива файлов дан на рис. 12.

В описании работы для GEORGE 3 имеется команда DIRECTORY (КАТАЛОГ), которая позволяет пользователю указать, какой именно каталог он употребляет в качестве базовой точки для опознавания файлов, используемых в данной программе. Если такая команда отсутствует, то в качестве



Р и с. 12.

текущего каталога берется *собственный каталог* пользователя — один из совокупности системных административных каталогов. В системе имеется по одному такому каталогу на каждого пользователя системы. Имя каждого собственного каталога пользователя и другая информация о нём, такая, как бюджет, хранится в системном файле, называемом *каталогом пользователей*.

На файл можно сослаться одним из двух способов:

1. По локальному имени

Если в описании работы пользователя указано одиночное имя файла, то оно относится к вводу или выводу файлу с этим именем, который находится в иерархии архива непосредственно под текущим каталогом пользователя.

2. По абсолютному имени

Это имя файла, к которому добавлено имя пользователя. Используя абсолютное имя, каждый пользователь может ссылаться на файл, находящийся непосредственно под собственным каталогом другого пользователя системы, который заранее информировал операционную систему о том, что такое обращение разрешается.

7.3.1. Команда LINK

Команда LINK (СВЯЗАТЬ) позволяет пользователю под частным именем в текущей программе ссылаться на любой файл в архиве, к которому ему дано право доступа. Например, команда

LINK SALARY: PERSONNEL.WAGES

заставит операционную систему обращаться к файлу с именем WAGES, принадлежащему пользователю PERSONNEL, всюду, где в текущей программе пользователя встретится имя файла SALARY. По команде LINK операционная система снабдит строку в собственном каталоге пользователя, относящуюся к имени SALARY, ссылкой на файл WAGES. Заметим, что это не означает дублирования данных, содержащихся в файле WAGES.

Команда LINK дает возможность пользователю писать программы, оперирующие с файлами, чьи имена, владельцы и положение в иерархии файлов подвергаются изменениям в течение жизни программы.

7.3.2. Команда STREAM

Команда

STREAM ACCOUNTS, DEBITS + CREDITS + BALANCE

это команда для операционной системы, в результате которой все обращения к файлу ACCOUNTS будут превращены в обращения к трем файлам DEBITS, CREDITS и BALANCE в данной логической последовательности. Команда STREAM может быть также использована для того, чтобы передать компилятору на трансляцию несколько сегментов исходной программы, записанных в виде самостоятельных кусков. Файлы, логически соединенные с помощью команды STREAM, представляются пользователю непрерывным потоком записей, так что он не замечает, в каком месте они стыкуются.

7.4. Учет и архив файлов

То обстоятельство, что собственные каталоги по одному на каждого пользователя физически представляют собой обычные узлы дерева архива, позволяет установить иерархию не только данных, но и пользователей. Директор вычислительного центра выделяет каждому пользователю ресурсы места, денег и времени центрального процессора в различных приоритетных группах. Связь этой системы с иерархической структурой архива такова, что каждый пользователь может поделить свои доли ассигнований между пользователями, которые в иерархии архива находятся под ним. Когда эти пользователи расходуют ресурсы, то это автоматически учитывается в ресурсах «главного» пользователя.

7.5. Секретность

Пользователю, однажды создавшему файл, разрешается доступ к файлу любым способом, какой ему нравится. Ни один другой пользователь в это время не имеет права обращаться к этому файлу. Однако в какой-то момент, когда файл уже существует, его владелец может передоверить его другим пользователям, т. е. разрешить им доступ к его файлу в одном или нескольких режимах. Например, он может разрешить только читать файл, только добавлять к нему записи, использовать его только как программу или, скажем, разрешить все эти режимы и, кроме того, записать в файл. Операционная система при этом не допустит никакого обращения к файлу, за исключением обращений от одного или группы пользователей, которым владелец файла его доверил, да и то только в

режимах, указанных владельцем. Владелец может использовать эту же систему как средство контроля за своей собственной программой, главным образом для защиты от ошибок.

Система GEORGE 3 реализует защиту в одном очень трудном случае, когда программа пользователя вводится в режиме off-line. Проблема здесь состоит в том, что очень легко подделать «удостоверение личности», пробиваемое на картах или перфоленте. Поэтому для пользователя введена возможность, позволяющая ввести свою программу в машину в режиме off-line как обычную работу, но объявить операционной системе о своем желании инициировать некоторую программу только после того, как он начнет сеанс работы в режиме on-line и тогда сообщит системе свой персональный пароль.

7.6. Дампинг

Дампинг, т. е. сохранение состояния архивных файлов, в системе GEORGE 3 делается по двум причинам. Это — переполнение архива и защита системы от сбоев аппаратуры. Дампинг и восстановление осуществляются тремя системными программами: Программой Пошагового Дампинга, Программой Восстановления Архива и Программой Обработки Сохраненной Информации.

7.6.1 Программа Пошагового Дампинга

Эта программа копирует на две магнитные ленты (на случай, если одна при чтении выдаст ошибку) текущее состояние всех on-line файлов архива, которые еще не копировались в их текущем состоянии. При каждом обращении к программе номер шага увеличивается на единицу и записывается в каталог каждого сохраняемого файла, что в дальнейшем облегчает процесс восстановления, если он потребуется. Система GEORGE 3 хранит информацию о местонахождении двух копий сохраненных файлов для каждого значения шага.

Операционная система обращается к Программе Пошагового Дампинга через интервалы времени, устанавливаемые директором вычислительного центра (скажем, каждые полчаса), или же когда это вызывается необходимостью из-за нехватки места, отведенного файлу пользователя, или из-за расширения запросов на вспомогательную память прямого доступа, используемую системой хранения файлов.

7.6.2. Программа Восстановления Архива

Эта программа выполняет действия, обратные тем, которые делает программа дампинга, а именно находит ранее сохраненные файлы или файл.

Операционная система обращается к этой программе после машинных сбоев или когда от программы пользователя поступает запрос на файл, который был сохранен программой дампинга из-за недостатка места в памяти.

Если после сбоя машины общая процедура восстановления в системе GEORGE 3 обнаруживает, что вышли из строя какие-то части дисковой памяти, то вызывается Программа Восстановления Архива, которая просматривает в обратном порядке ленты с сохраненной информацией и восстанавливает файлы, находившиеся в поврежденной части дисков. Как только хотя бы для одной активной программы будут восстановлены все нужные файлы, эта программа снова активизируется. Однако может произойти более длинная задержка, если потерян (затерт) главный каталог, поскольку, пока он не восстановлен, система не сможет запустить ни одной программы.

Другой причиной обращения к Программе Восстановления Архива может оказаться запрос некоторой программы на обращение к файлу, ранее запомненному на ленте из-за недостатка места на дисках операционной системы. Операционная система попытается накопить несколько запросов такого сорта, после чего Программа Восстановления Архива просмотрит нужные ленты и перепишет по очереди на диски все запрошенные файлы.

7.6.3. Программа Обработки Сохраненной Информации

Эта программа позволяет получить более рациональное расположение информации на лентах. Частота ее использования зависит от директора вычислительного центра. Программа извлекает самый последний вариант сохраненного файла, переносит его на ленту (как правило) с меньшими номерами шагов и соответственно корректирует индексы шагов.

ЯЗЫКИ ДЛЯ РАБОТЫ В НЕПОСРЕДСТВЕННОМ КОНТАКТЕ С МАШИНОЙ. РЕЖИМ ДИАЛОГА

Дэвид Фостер

В главе 5 при обсуждении систем с мультидоступом было введено понятие разговорных компиляторов. Дэвид Фостер развивает его более детально, начиная с описания проблемы общения с пользователем и необходимости иметь быстрый доступ к машине. Эти проблемы особенно важны при разработке программ, и Дэвид Фостер останавливается на них достаточно детально. Он делает обзор методов, использовавшихся в прошлом, чтобы показать, какие возможности хотелось бы иметь, и таким образом подводит нас к необходимости диалога с машиной. Затем он описывает развитие разговорных компиляторов, отвечающих требованиям диалоговой работы, и иллюстрирует ее на примере одного из доступных для GEORGE 3 языков — языка JEAN. В заключение он показывает, как преимущества таких компиляторов можно объединить с преимуществами обычных языков, приводя в качестве примера систему «разговорного Фортрана».

8.1. Проблема общения

Одним из главных факторов, от которого зависит эффективность вычислительной машины, является легкость, с которой пользователь может сообщить машине, что она должна для него сделать. Машина в основном может выполнять только элементарные операции, такие, как сложение двух чисел или проверка их равенства, поэтому, для того чтобы машина могла решить задачу, последняя должна быть выражена в терминах этих элементарных операций. Может оказаться, что задача требует огромного количества таких операций, и расчленение ее на эти операции становится трудной проблемой. С точки зрения пользователя это далеко не лучший способ описания задачи для решения ее на машине.

Это и послужило причиной развития языков программирования и компиляторов. Пользователю на одном из таких языков гораздо легче описать свою задачу. А специальная программа, названная компилятором, используется для перевода задачи с такого языка на язык элементарных операций машины. Иными словами, компилятор превращает программу, написанную на языке программирования в объектную программу, состоящую из машинных команд.

К языкам программирования, например, относится COBOL, позволяющий писать программы для коммерческих расчетов на ограниченном английском языке; языки для решения научных задач ALGOL и FORTRAN, основанные на обычной ма-

тематической символике; язык PL/I, спроектированный с учетом запросов как научных, так и коммерческих задач. Например, компилятору можно дать задание вычислить корень квадратного уравнения $Ax^2 + Bx + C = 0$ с помощью исходного предложения на Фортране:

$$\text{ROOT} = (\text{SQRT}(B * B - 4.0 * A * C) - B) / (2.0 * A)^1)$$

Очевидно, намного легче написать такое предложение, чем примерно 30 элементарных команд, которые нужны для этого вычисления.

8.2. Проблема доступа

Языки программирования существенно продвинули решение проблемы общения с вычислительной машиной. Однако существует другая серьезная проблема — проблема доступа.

Самым распространенным способом использования машины был способ пакетной обработки. Пользователь писал свою исходную программу на каком-нибудь языке, например на Алголе, Коболе или Фортране, отдавал ее перфорировать и затем передавал на машину. Программа транслировалась, полученная объектная программа выполнялась, и через какое-то время пользователь получал результаты. Программа вначале, как правило, оказывалась с ошибками. Результаты обычно содержали в себе некоторую диагностическую информацию, помогающую пользователю найти свои ошибки, но все же требовалось несколько пусков и некоторое время, пока программа начинала работать правильно.

Критическим параметром во всем этом является время от представления работы на машину до получения результата. Это время зависит от ряда вещей, таких, как расстояние от места работы пользователя до машины, загруженность машины и эффективность административных служб. Если загруженность на машине небольшая и пользователь находится достаточно близко от машины, то результат он мог бы получить в пределах часа. Но если путь от места службы до машины долгий и загруженность ее большая, то этот цикл, или, другими словами, время оборота, может вырасти до 24 или 48 часов.

Для того чтобы пользователь мог успешно продвигаться в своей работе, ему необходим немедленный доступ к машине. Этого можно достичь на пути, который принят в системе WATFOR — очень быстрой системе пакетной обработки с использованием Фортрана и с выполнением программы

¹⁾ Символ * обозначает операцию умножения. — *Прим. перев.*

сразу после трансляции, разработанной в университете г. Ватерлоо (Канада). Система обеспечивает обслуживание наподобие кафе самообслуживания и предоставляет студентам возможность транслировать и пропускать небольшие программы, написанные на Фортране. Пользователь стоит в очереди, чтобы передать свою отперфорированную на картах исходную программу оператору, который подложит ее к пакету других программ и загрузит в читающее устройство. Затем очередь выстраивается у печатающего устройства, где пользователь имеет возможность получить результаты сразу, как только они будут отпечатаны. После этого очередь возвращается к читающему устройству, где пользователь получает свои карты обратно. На практике могло быть около 10 человек, ожидающих своей очереди вручить карты оператору, и время от момента, когда пользователь становился в очередь, до момента, когда он получал свои карты обратно, равнялось приблизительно двум минутам.

Однако такая система ограничивается обработкой только небольших программ в строго определенных условиях. Потребность в немедленном доступе более широко обеспечивается возможностями оперативного общения в системе, подобной GEORGE 3. С помощью телетайпа, соединенного с машиной и расположенного на рабочем месте пользователя или вблизи него, пользователь может на расстоянии инициировать свои работы и управлять их ходом, вместо того чтобы посылать свои программы на машину.

На практике он может хранить программу, написанную на Фортране или Алголе, в виде файла в архиве. Пользуясь клавиатурой телетайпа, он в любой момент может попросить машину протранслировать и выполнить эту программу. Ему разрешено пользоваться любыми внешними устройствами на машине, но в процессе отладки своей программы он обычно заказывает печать результатов на своем телетайпе, чтобы не ждать, когда они придут к нему по другим каналам. Если в его программе обнаружилась ошибка, он может исправить свою исходную программу с телетайпа, снова протранслировать ее и пустить в счет. Таким образом, отладка программы может происходить непрерывно без мешающих работе задержек.

8.3. Необходимость диалога

Остается еще проблема, которая в особенности важна во время отладки программы. До сих пор во время трансляции или счета программы не было настоящего взаимодействия между пользователем и машиной. Сначала пользователь опре-

делял, что нужно сделать. Затем машина обрабатывала данные и выдавала результаты, после чего пользователь начинал планировать следующий этап работы. В общем этого достаточно, но пользователь мог бы работать намного быстрее, если бы был возможен некоторый диалог между машиной и человеком.

На заре эры вычислительных машин пользователь мог сидеть за пультом, монополизируя машину на время отладки своей программы, и работать в режиме диалога с ней. Он останавливался в некоторой точке, проверял содержимое ячеек памяти, исправлял программу, если это было необходимо, и продолжал счет с того места, на котором он прервал выполнение программы. Сейчас этого не делают, так как очень невыгодно отдавать целиком машину в пользование одного человека и, кроме того, разговор с машиной лучше вести в терминах Фортрана или Алгола, а не в терминах разрядов ячеек памяти. Но в некотором отношении возможность такого рода диалоговой работы была бы полезной, если бы при этом не терялась эффективность.

Например, были бы полезны такие возможности, если бы их можно было обеспечить с помощью телетайпа.

1. Если при вводе исходной программы с телетайпа в ней обнаруживается синтаксическая ошибка, то пользователю об этом немедленно сообщается и ему предоставляется возможность набрать предложение снова.
2. Осуществляются меры по всестороннему контролю за правильностью выполнения программы, и, если обнаружена ошибка, программа останавливается и об этом сообщается пользователю.
3. Во время таких или заранее запланированных остановок пользователь имеет возможность проверить содержимое памяти, попросив для этого напечатать переменные, которые он укажет.
4. Во время таких остановок пользователь может также изменить значения переменных или модифицировать программу, исправляя исходные предложения, прежде чем он продолжит счет.

Именно для реализации возможностей названного типа и созданы разговорные языки.

8.4. Системы с разговорными языками

Разговорные языки, подобно таким языкам, как Алгол или Фортран, разрабатываются для того, чтобы предоставить пользователю возможность естественным способом выражать свою задачу для вычислительной машины. Но, кроме

того, в них предусмотрены операции, связанные с работой на телетайпе и позволяющие включать такого сорта возможности, которые были перечислены выше.

Одним из первых разговорных языков был JOSS, разработанный Дж. Шоу, сотрудником RAND Corporation. Разговорный язык JEAN, разработанный фирмой ICL для машин серии 1900, основывался на языке JOSS и очень похож на него. Эти языки создавались для того, чтобы предоставить пользователю средства, нужные для отладки и выполнения в диалоговом режиме программ, предназначенных для решения математических задач. Они очень просты, и их легко освоить.

Здесь уместно задать вопрос — является ли диалоговый режим эффективным с точки зрения использования машины? На первый взгляд пользователь, сидящий за телетайпом и думающий о том, какие исправления или проверочные действия предпринять следующими, кажется, не способствует эффективности.

Однако эффективность в такой системе в основном зависит от распределения времени. Большое число пользователей одновременно могут пользоваться системой, и в то время, когда некоторые из пользователей сидят за телетайпом и думают или вводят новую информацию, машина может быть занята выполнением других программ. В большинстве случаев в то же время можно также вести обработку пакета фоновых задач.

Кроме того, для многих диалоговых языков, подобных JEAN, пишутся шаговые трансляторы. Это означает, что каждое предложение исходной программы обрабатывается независимо от других вплоть до момента ее выполнения. Если вставляется новое или исправляется старое предложение, то заново транслируется не вся программа или какая-либо ее часть, а только это предложение.

В JEAN и в некоторых других системах это достигается с помощью интерпретации. Когда вводятся исходные предложения, они не транслируются немедленно, а запоминаются в виде строчек символов, проходя очень небольшую обработку. Строка интерпретируется, когда выполняется предложение. Это означает медленное выполнение, но быстрое внесение исправлений и добавлений (редактирование), а в процессе создания программы время, затрачиваемое на редактирование, оказывается более существенным фактором.

В некоторых системах, сохраняя шаговый принцип трансляции, добиваются повышения эффективности за счет большей обработки предложений при вводе. Это увеличивает скорость выполнения, сохраняя простоту внесения изменений.

8.5. JEAN

Система JEAN, реализованная на машинах серии 1900, позволяет ряду пользователей (число которых зависит от размера машины) одновременно создавать и выполнять программы для математических расчетов, используя телетайпы, дистанционно связанные с машиной.

Поскольку в GEORGE 3 используются возможности одновременной работы в режиме on-line со многими пользователями, каждый пользователь должен ждать, когда система напечатает ему символ \leftarrow в качестве разрешения печатать ввод с телетайпа. Самое простое, что можно затем сделать, это послать команду выполнить какое-то арифметическое действие (или последовательность действий) и получить результат (результаты). Например,

```

 $\leftarrow$  TYPE (79.15 + 33.27 - 15.3)* 4
      (79.15 + 33.27 - 15.3)* 4 = 388.48
 $\leftarrow$  TYPE X*(X + 1) FOR X = 1 (1) 4
      X*(X + 1) = 2
      X*(X + 1) = 6
      X*(X + 1) = 12
      X*(X + 1) = 20

```

Вторая команда четыре раза выполняет вычисление выражения $X*(X + 1)$ со значениями $X = 1, 2, 3, 4$ соответственно.

Обычно с помощью JEAN получают программы для более длинных вычислений, состоящих из нескольких шагов. Для этого используются косвенные команды. Каждой такой команде предшествует номер в виде смешанного десятичного числа, который не имеет отношения к порядку ввода предложений, а указывает на порядок выполнения шагов (по возрастанию номеров).

Например, пользователь мог бы написать следующую программу для вычисления и вывода на печать корней квадратного уравнения $Ax^2 + Bx + C = 0$:

```

 $\leftarrow$  1.1 DEMAND A, B, C
 $\leftarrow$  1.2 X = (- B + SQRT (B  $\uparrow$  2 - 4*A*C))/(2*A)
 $\leftarrow$  1.3 Y = (- B - SQRT (B  $\uparrow$  2 - 4*A*C))/(2*A)
 $\leftarrow$  1.4 TYPE X, Y IN FORM 1
 $\leftarrow$  1.5 TO STEP 1.1
 $\leftarrow$  FORM 1:
      ROOTS = # # # . # # # , # # # . # # #

```

Во время выполнения этой программы шаг 1.1 пошлет запрос на ввод с телетайпа значений A, B и C. После того как пользователь напечатает эти значения на телетайпе, шаги 1.2

и 1.3 вычислят корни. Затем шаг 1.4 напечатает их в указанном формате, где символ $\#$ представляет одну десятичную цифру. Шаг 1.5 вызовет возврат к шагу 1.1, и процесс повторится.

Пользователь может ввести команду, останавливающую процесс повторений. Он печатает

```
← 1.1 DEMAND A
← 1.13 DONE IF A = 0
← 1.16 DEMAND B, C
```

Это вызовет замену старого шага 1.1 и помещение перед шагом 1.2 двух новых шагов. Программа будет остановлена, если в ответ на запрос значение A будет задано нулевым.

Для того чтобы выполнить программу, пользователь печатает

```
← DO PART 1
```

По этой команде система выполняет по порядку все шаги, номера которых имеют 1 в качестве целой части. Диалог мог бы проходить так:

```
← DO PART 1
A = ← 1
B = ← 1
C = ← - 12
ROOTS = 3.000, -4.000
A = ← 3
B = ← 11
C = ← 8
ROOTS = -1.000, -2.667
A = ← 0
```

```
←
```

В JEAN имеется ряд диалоговых свойств. Синтаксические ошибки обнаруживаются либо сразу после ввода предложения, либо в момент выполнения ошибочной команды. В любом случае печатается ЕН? (что?) и пользователь имеет возможность снова ввести исправленное предложение. Например,

```
← 3.1 TIPE X, Y
ЕН?
← 3.1 TYPE X, Y
```

Если пользователь попытается использовать переменную до того, как установит ей значение, JEAN сообщит об этом пользователю. Например, JEAN напечатает сообщение для переменной Z в таком виде:

```
INTERRUPTED: Z = ???
```

```
←
```

Пользователь может установить значение для Z и продолжить счет.

Другие семантические ошибки также вызывают печать сообщений на телетайпе и приостановку счета. При таких или заранее запланированных остановах пользователь может отпечатать значения переменных, используя команду TYPE. Он может также изменить свою программу, меняя старые или вставляя новые шаги, и продолжить счет с любого места. Например,

← 2.4 $Z = (Y - 1) * (Y - 2) - 1$

← 2.5 $X = X / (1 - Z \uparrow 2)$

⋮

← DO PART 2

ERROR AT STEP 2.5: I HAVE A ZERO DIVISOR¹⁾

← TYPE Y, Z

$Y = 3$

$Z = 1$

← 2.45 $Z = Z / 4$

← TO STEP 2.45

Здесь пользователь смог вставить пропущенную ранее команду и продолжить программу с этого места.

8.6. Положение общепринятых языков в системах программирования

Такая система, как JEAN, предоставляет пользователю несколько преимуществ. Ее главная сила — в диалоговых возможностях. Такие возможности позволяют пользователю легко и за очень короткое время создавать и отлаживать программы.

Реализация JEAN следует принципам шаговой трансляции и интерпретативного выполнения программы. Как мы говорим, это означает медленное выполнение, но быстрое внесение изменений, что является ценным качеством в работе по созданию программ.

Однако обычные компилирующие системы, такие, как Фортран, имеют свои преимущества перед JEAN. Они предоставляют пользователю более мощный язык, позволяющий ему более разумно конструировать большие и сложные программы. Фортран — устоявшийся международный язык.

¹⁾ Это сообщение в переводе означает:

ОШИБКА В ШАГЕ 2.5: У МЕНЯ НУЛЕВОЙ ДЕЛИТЕЛЬ

— Прим. перев.

Многие программы и подпрограммы на Фортране существуют и доступны для пользования. Кроме того, хотя отладка программы на Фортране дольше, чем подобной программы на JEAN, но уж если она отлажена, компилятор с Фортрана предоставит пользователю необычайно эффективную объектную программу. Если при этом потребуется много раз считать по отлаженной программе, то с точки зрения затрат машинного времени значительно выгоднее считать по программе, полученной компилятором с Фортрана, чем с помощью системы, подобной JEAN.

Естественно, что многие разработчики стали пытаться соединить преимущества и той и другой системы, создавая разговорные компиляторы для обычных языков.

Ярким примером такой диалоговой системы служит система RUSH, разработанная фирмой Allen-Babcock Computing, Inc. Эта система, созданная на базе языка PL/I, лежит в основе дистанционного доступа к службе вычислений. К моменту написания этой книги к системе было подключено около 150 терминалов для обслуживания пользователей, разбросанных по всей территории США.

Перечислим требования к диалоговой системе с языком типа Фортран.

1. Должна быть обеспечена возможность вести всю работу (ввод, трансляцию, отладку и выполнение программы) с одного дистанционного телетайпа с сохранением возможности использовать при необходимости другое периферийное оборудование.
2. Особое внимание должно быть уделено быстрой и легкой отладке. Должны быть предусмотрены разнообразные средства для всесторонней диагностики, обязательно должны быть предоставлены возможности, перечисленные в 8.3.
3. Внесение исправлений должно быть быстрым и легким и не должно требовать долгих перетрансляций. В момент приостановки счета должна быть возможность внести по крайней мере небольшие исправления и продолжить вычисления с того места, где они были прерваны, а не начинать все сначала.
4. Скорость выполнения программы пользователя должна быть по возможности быстрой, но только не в ущерб требованиям пунктов 2 и 3. Кроме того, должна существовать возможность программу, отлаженную в диалоговом режиме, протранслировать обычным способом и получить эффективную объектную программу.
5. Система должна допускать эффективную работу со многими пользователями одновременно.

Эти требования были главными при создании разговорной системы с Фортраном для ICL 1900, о некоторых возможностях которой рассказывает следующий раздел.

8.7. Диалоговая система с Фортраном для ICL 1900

Данная система позволяет пользователю с телетайпа создавать программы на Фортране. В основе ее работы лежит обработка отдельных предложений и сегментов Фортрана. Каждое предложение в сегменте имеет свой номер в виде смешанного десятичного числа.

Так же как в JEAN, пользователь должен подождать, пока система не выдаст ему символ \leftarrow , предлагающий пользователю начать печатать на телетайпе какое-то свое сообщение.

Для того чтобы начать строить сегмент, пользователь должен в ответ на это приглашение напечатать заглавное предложение сегмента. Система ответит на это, отпечатав имя сегмента, заключенное в круглые скобки, на следующей строке слева, чтобы указать, что данный сегмент стал для системы текущим. Затем в следующей строке появится 1 в качестве номера строки и стрелка \leftarrow , приглашающая пользователя напечатать первое предложение сегмента. Когда пользователь напечатает это предложение, ему будет дано приглашение печатать следующее и т. д. В конце сегмента он должен напечатать END. Например,

```

 $\leftarrow$  REAL FUNCTION AREA (A, B, C)
      (AREA)
        1  $\leftarrow$  S = (A + B + C)
        2  $\leftarrow$  AREA = SQRT(S*(S - A)*(S - B)*(S - C))
        3  $\leftarrow$  END
 $\leftarrow$ 

```

Чтобы заменить или вставить предложение в текущий сегмент, пользователь должен только напечатать номер изменяемого (или вставляемого) предложения и за ним само предложение. Например,

```

 $\leftarrow$  1 S = (A + B + C)/2.0
 $\leftarrow$  55.5 RETURN

```

Затем пользователь мог попросить отпечатать исправленный сегмент. Его команды и ответ на них выглядели бы следующим образом:

```

 $\leftarrow$  LIST AREA
      0 REAL FUNCTION AREA (A, B, C)
        1      S = (A + B + C)/2.0
        2      AREA = SQRT(S*(S - A)*(S - B)*(S - C))
      55.5      RETURN
      END

```

Несколько необычный выбор номера для предложения RETURN показывает, как можно расширить сегмент. Следует заметить, что если пользователь хочет начать построение сегмента с середины, а не с начала, то он может построить пустой сегмент, а затем использовать средства редактирования.

Имеются также средства для выбрасывания предложений, замены и вставки блоков предложений, копирования предложений из одного сегмента в другой.

Воспринимая очередную строку исходной программы, система проверяет ее синтаксис. В случае ошибки в следующей строке печатается EH? Пользователю при этом дается возможность ввести предложение заново.

Если в этом месте пользователь захочет получить дальнейшие разъяснения по поводу ошибки, он должен напечатать слово HELP. После получения разъяснений пользователю опять предоставляется возможность ввести предложение заново. Например,

```
33 ← A = (X + 1/(X + 1/(X + 1/(X + 1/X)))
EH?
33 ← HELP
MISSING PARENTHESIS 1)
33 ← A = (X + 1/(X + 1/(X + 1/(X + 1/X))))
34 ←
```

Различные средства слежения (трассирования программы) позволяют пользователю находить ошибки и во время выполнения своей программы. Он имеет возможность заказать диагностическую печать команд передачи управления или некоторых промежуточных результатов. Он также может указать места в своей программе, на которых он хотел бы остановиться.

Выполнение программы пользователя задается директивой

← RUN

Выполнение может быть прервано по нескольким причинам. Например, встретилась команда STOP, достигли точки, в которой пользователь просил остановиться, или обнаружена ошибка. В любом случае на телетайпе печатается сообщение, содержащее причину остановки и предложение программы, на котором он произошел.

Во время таких остановов пользователь может предпринять ряд действий. Он может напечатать часть своей програм-

¹⁾ Это сообщение в переводе означает:

ПРОПУЩЕНА СКОБКА

мы, исправить ее, изменить задание на диагностическую печать. Он может попросить напечатать какие-то переменные, элементы массива и т. п. Он может набрать на клавиатуре любое предложение Фортрана без предшествующего ему номера. В таком случае предложение не запоминается в памяти, но немедленно выполняется. А это означает для пользователя возможность, например, изменить значения переменных или передать управление на другое предложение. Затем в большинстве случаев он может напечатать команду

← GO

которая вызовет продолжение выполнения его программы с того места, где она была остановлена. Однако этого нельзя делать при такого рода исправлениях, как изменение типа переменной, после которых выполнение программы нужно обязательно начать сначала.

Пользователь может в любой момент прервать выполнение своей программы, если он подозревает, что она зациклилась. И в этом случае он может предпринять любые возможные действия, подобные тем, о которых только что говорилось.

Во время выполнения программы данные, естественно, выводятся на телетайп, а требуемый ввод запрашивается у пользователя по мере необходимости. Однако для пользователя имеется возможность создавать сегменты данных и, следовательно, осуществлять ввод заранее до выполнения программы. Сегменты из данных можно редактировать.

Система предназначена для работы с системами GEORGE 3 и GEORGE 4, так что при желании пользователь может вводную и выводную информацию иметь в архиве файлов. То же самое относится и к исходной программе и к получаемому после трансляции листингу.

Система работает на основе идеи пошаговой трансляции. Выполняемые предложения Фортрана компилируются безотносительно к декларативным (описательным) предложениям; обработка деклараций при трансляции сводится к заполнению соответствующих таблиц. В момент, когда по директиве RUN инициируется выполнение, полученная при трансляции программа просматривается и настраивается согласно декларациям.

Блоки команд, получаемые в результате компиляции, соединяются между собой ссылками, образуя списочную структуру. Отсюда и возникает возможность вставлять или выбрасывать предложения без лишних перетрансляций, а также возможность после исправления продолжить выполнение программы с того места, где оно было прервано. Если исправляется декларация, то необходим повторный просмотр

программы для настройки, но полной перетрансляции не требуется. Отсюда вытекает возможность быстрых исправлений.

Но такие конструктивные особенности объектной программы замедляют ее работу по сравнению с программой, полученной с помощью обычного транслятора, примерно в три раза. Это, однако, допустимо на этапе создания и отладки программы, а как только она заработает, ее можно будет пропустить через обычную трансляцию с тем, чтобы получить эффективную объектную программу. Кроме того, имеется возможность уже отлаженные подпрограммы транслировать обычным транслятором и подключать к программе, создаваемой в диалоговом режиме.

Поскольку система проектировалась для работы с GEORGE 3, она является операционной системой, которая обеспечивает одновременное обслуживание многих пользователей. Эта система будет эффективнее работать для машины со страничной памятью, такой, как 1906A, использующей страничную версию операционной системы — GEORGE 4. В этом случае доступна большая виртуальная память, и у большого количества пользователей появляется возможность работать с одной и той же копией диалоговой системы при условии, что каждый из них имеет свою рабочую область оперативной памяти. Если потребуется, некоторые страницы оперативной памяти будут пересылаться во внешнюю память и обратно, но сама разговорная система, вероятнее всего, будет оставаться в оперативной памяти.

РЕАЛЬНОЕ ВРЕМЯ. СПЕЦИАЛЬНЫЕ ТРЕБОВАНИЯ К УПРАВЛЯЮЩИМ ПРОГРАММАМ

Стюарт Дж. Миллер

На примере супервизора и систем типа GEORGE мы имели возможность проиллюстрировать большинство свойств операционных систем пакетной обработки и мультидоступа. Для иллюстрации свойств реального времени Стюарт Миллер выбирает другую систему — UNIVAC STARS. Сначала он описывает характерные черты систем реального времени и, в частности, объясняет, почему нужны новые концепции, отличные от концепций, положенных в основу пакетной обработки. Затем он рассматривает тип требуемой системы управления, методы управления сложной сетью каналов связи и специальные вопросы, относящиеся к запоминающим устройствам с произвольным доступом, в условиях работы в реальном времени. В заключение после рассмотрения вопросов управления программами и временем он обсуждает важность быстрого восстановления работоспособности системы после отказов аппаратуры.

9.1. Введение

Тем, кто имеет какое-то отношение к вычислительным машинам, хорошо известны многие черты общепринятой на сегодня пакетной обработки. На протяжении всех лет развития вычислительных машин всегда стояла цель — добиться для таких систем еще большей эффективности и производительности. Это привело сначала к убыстрению памяти и процессора вместе с некоторыми изменениями в архитектуре машин, а затем к появлению возможности решать одновременно несколько задач на одной машине. Последнее было достигнуто более тесной кооперацией разработчиков аппаратуры и программистов. В результате этих и других усовершенствований стало выгодно применять вычислительные установки в экономике, административном управлении, образовании и прикладных областях, что обратило на себя внимание пользователей, которые в свою очередь стали предъявлять к машинам все возрастающие требования.

Развитие вычислительных систем, а точнее, развитие диалогового способа общения с машиной, стремительно продолжается. И системы реального времени безусловно оказывают все возрастающее влияние на применения вычислительных машин.

Чтобы понять, что скрывается под словами *реальное время*, сначала нужно осознать, что лежит в основе любой организации. Любое действие организации состоит из некоторого

набора процедур независимо от того, что хочет она получить в результате: вал, выточенный на токарном станке, или результат денежных расчетов, или какое-то административное решение. Это одинаково применимо к коммерческим, административным, учебным и прикладным организациям. Степень человеческого участия варьируется в зависимости от содержания процедуры. Например, роль человека, присматривающего за работой автомата, производящего бутылки, можно считать скорее пассивной, в то время как роль кассира в банке, наоборот, активна. Любую процедуру можно разбить на элементы, которые мы назовем подмножеством главной процедуры. Такое деление можно производить бесконечно. Возвращаясь к исходному вопросу, мы утверждаем, что любая организация представляет собой иерархию диалоговых процедур.

Это утверждение не такое уж хитрое, как может показаться, и оно позволяет нам в общих терминах определить принципы работы в реальном времени. Итак, вычислительная система реального времени — это такая система, в которой машина и человек выполняют процедуры, являясь партнерами в диалоговой работе, причем *интерфейс между человеком и машиной определяется исключительно требованиями человека*. В обычной пакетной обработке как раз наоборот — интерфейс между человеком и машиной определяется требованиями машины.

Исторически сложились три основных типа систем реального времени:

1. Системы управления процессами

Здесь участие человека в процедурах скорее пассивное, чем активное, а требования к эффективности функционирования окружающего оборудования самые жесткие. Эти системы обычно применяются в управлении индустриальными процессами, уличным движением и космическими полетами.

2. Коммерческие системы реального времени

Здесь происходит самый интенсивный диалог между человеком и машиной, каждый из которых выполняет разные элементы процедуры, принимает решения и реагирует на принятое решение. Системы в основном применяются для резервирования билетов авиалиний, в медицине, в банковском деле, в информационных и других системах, работающих с централизованными файлами.

3. Системы мультидоступа

Здесь несколько независимых пользователей используют одну машину каждый для своей цели. Системы обычно применяются в учебной, научной и некоторых коммерческих областях.

В этой главе мы рассматриваем коммерческие системы реального времени. Их можно назвать системами для работы со многими пользователями в отличие от систем мультидоступа, которые можно назвать системами для многих работ пользователей. Вопросы, касающиеся систем управления процессами, очень специальные и не будут рассматриваться в этой книге.

На рис. 13 показана конфигурация типичной большой коммерческой системы реального времени. К аппаратуре такой системы предъявляются следующие наиболее важные требования:

1. Особые требования к коммуникационной аппаратуре

Хотя полной сети каналов связи не показано, ясно, что она должна быть сложной из-за большого количества пользователей и их территориальной разбросанности. Возможно многоступенчатое подключение мультиплексоров, и обычно имеются дистанционные буферные устройства управления.

2. Дублирование оборудования

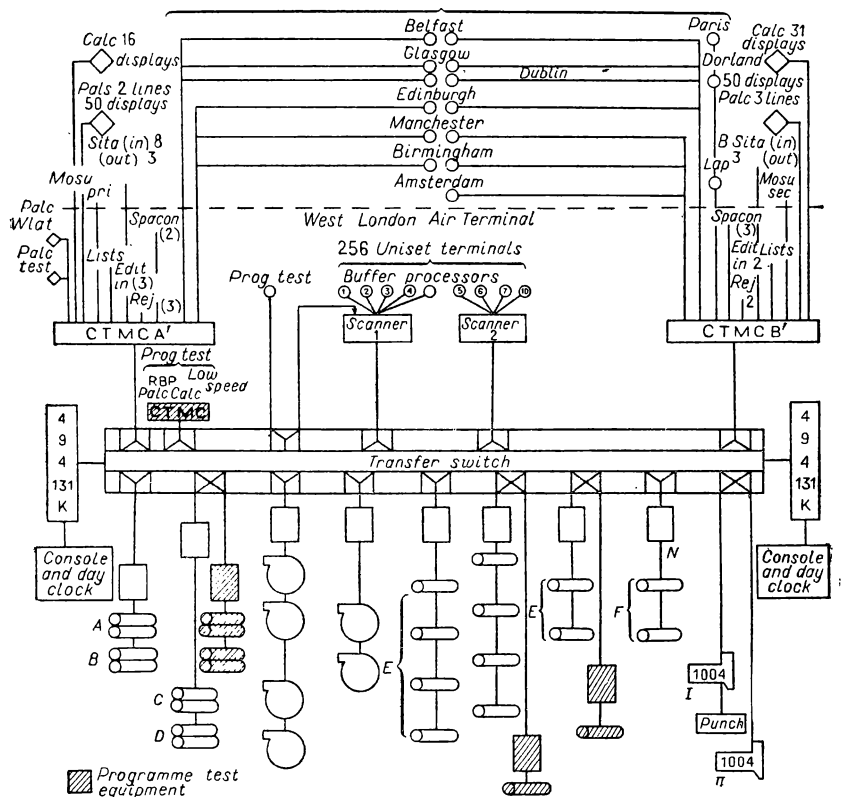
Так как система предназначена для обслуживания массового потребителя, то вероятность отказа должна быть минимальной. Ни программы, ни аппаратура не гарантированы от ошибок, и на случай отказа должно иметься резервное оборудование, способное продолжить обслуживание. Дублируются центральный процессор, файлы и коммуникационная аппаратура.

3. Большой архив файлов

Почти все действия, выполняемые в системе, так или иначе касаются файлов. Их защита имеет первостепенное значение, так как любое искажение данных могло бы породить неразбериху в работе. На рисунке показано дублирование главных файлов ($AB=CD$) и дублирование малых файлов-индексов ($E=F$). Файлы и системные программы хранятся в памяти с произвольным доступом. Магнитные ленты используются для периодического сброса на них содержимого файлов (дампинга) и для работы с пакетом фоновых задач.

В любой момент времени существует система *on-line* и система *off-line*. Система *on-line* содержит в себе всю аппаратуру, необходимую для выполнения работ в реальном времени, т. е. один центральный процессор, все четыре мультиплексора,

Remote buffer processors (250 terminals)



Р и с. 13.

Buffer processor — буферные процессоры; CTMC — мультиплексор; Console and day clock — пульт и часы; Punch — перфоратор; Programme test equipment — оборудование для отладки программ; Remote buffer processors (250 terminals) — дистанционные процессоры (250 терминалов); Scanner — сканирующее устройство; Transfer switch — коммутатор каналов; 494 131 K — процессор машины UNIVAC-494 с памятью 131 K.

файлы АВ, CD, Е и F, один ленточный канал и один канал для операций с картами и печатью. Система *off-line* состоит из остального оборудования и в то же время может заниматься обычной пакетной обработкой, тестированием программ и т. д. Коммутаторы каналов должны обеспечивать возможность соединения любого процессора с любым периферийным устройством.

9.2. Различие концепций пакетной обработки и работы в реальном времени

При рассмотрении управляющих программ систем реального времени существенно сначала понять основное функциональное отличие этих систем от систем обычной пакетной обработки. Мы уже видели, что термин «реальное время» охватывает широкий диапазон вычислительных систем, но по крайней мере в этом параграфе можно не различать их типы, так как все они имеют общее функциональное свойство, состоящее в том, что большинство действий в системе инициируется некоторыми внешними событиями. Это точная противоположность системе пакетной обработки, где большинство действий диктуется некоторыми внутренними событиями (обычно некоторыми командами в программе). Удобно называть системы реального времени *экзогенными*, а системы пакетной обработки — *эндогенными*.

Чтобы пояснить это, рассмотрим простую программу, написанную для того, чтобы обновлять (вставлять, выбрасывать, заменять) записи в файле в соответствии с некоторыми вводимыми элементами информации. В системе пакетной обработки эта программа могла бы иметь дело с каждым элементом вводимых данных отдельно и после окончания обработки одного вызывать следующий и повторять процесс до тех пор, пока не исчерпается поток вводимой информации (это верно и для программы, применяющей буферизацию). Таким образом, сама программа определяет, когда она начинает обработку каждого элемента.

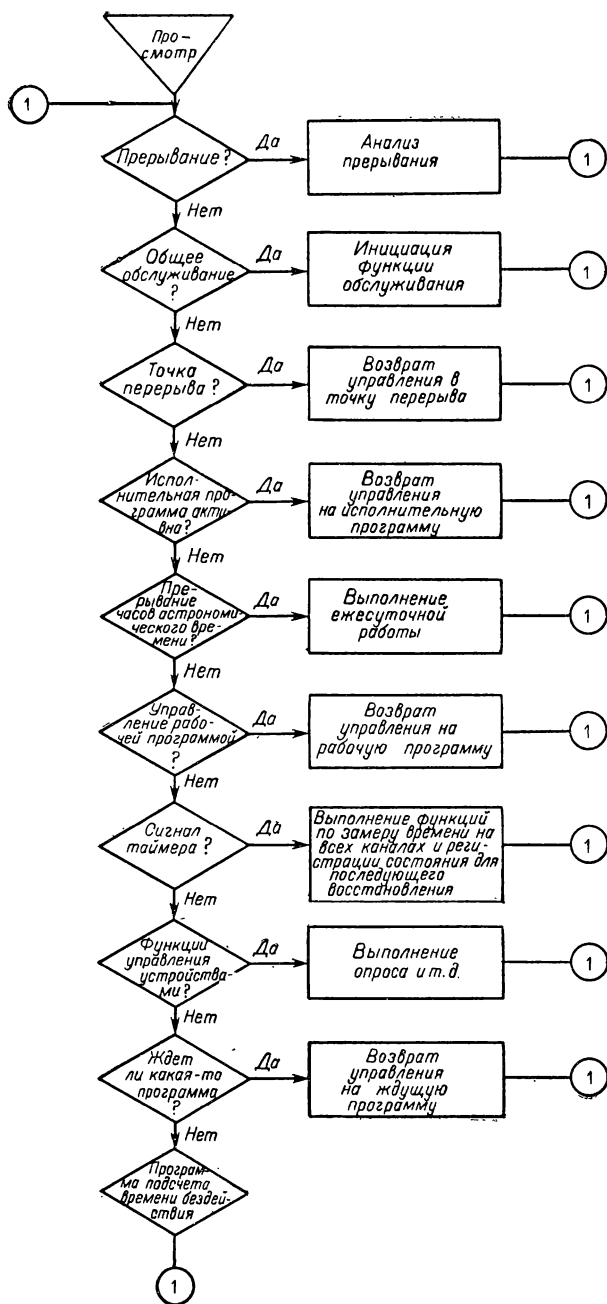
Теперь представим эту же программу в системе реального времени. Так как элементы данных приходят от терминалов в заведомо непредсказуемые моменты времени, прежняя тактика становится неэффективной из-за того, что, пропуская через себя эти элементы последовательно, программа может стать узким местом в период пиковых нагрузок на систему и вызвать задержки в обслуживании операторов, сидящих за терминалами (это подтверждается теорией очередей и напоминает, например, большой гастроном субботним утром с одной кассой). Например, на прокатном стане или химическом заводе, где роль терминалов играют пульта управления, любая задержка могла бы легко привести к хаосу и даже катастрофе. Несмотря на экзогенное свойство систем реального времени, необходимо, чтобы центральный процессор всегда мог контролировать свое состояние; одна из обязанностей системы управления состоит в том, чтобы допускать экзогенные события при эндогенном способе работы (см. 9.4), по возможности сводя к минимуму задержки в обработке.

Любое событие, возникающее внутри системы, можно рассматривать как требование выполнить некоторую задачу. Событиями могли бы быть прерывания от коммуникационного оборудования или файлов или же запросы на обслуживание от прикладных программ. Чтобы удовлетворять эти заявки, система должна выделять ресурсы соответствующим задачам. В этом смысле рабочие программы системы, память процессора и время процессора можно рассматривать как ресурсы. Одновременно может поступить много запросов на эти ресурсы, и система управления должна оптимизировать их распределение так, чтобы задержки в выполнении задач были по возможности наименьшими. Каждому возможному событию присваивается приоритет, и инициация соответствующих ему действий зависит от этого приоритета.

9.3. Структура системы управления

Центральной программой системы управления является работающая по замкнутому циклу программа, которая проверяет по соответствующему признаку появление любого события и планирует ресурсы для его обработки. Этот цикл имеет различные названия: *просмотр планировщика, просмотр приоритетов, переключатель, динамический распределитель, коммутатор управления, холостой цикл*. Все признаки событий просматриваются в строгой приоритетной последовательности, и, если никаких событий не оказалось, просмотр начинается сначала. Распределение приоритетов может быть различным в разных системах и определяется на этапе проектирования каждой системы на основе анализа зависимости событий от времени. В общем можно сказать, что сигналы прерывания от периферийных устройств имеют самый высокий приоритет, так как их появление означает изменение состояния системы. Аппаратура может также предъявлять определенные требования, согласно которым те или иные действия, необходимые для поддержания правильного функционирования аппаратуры, должны выполняться в ограниченное время. Поэтому рабочие программы, ожидающие своей очереди начать работу, стоят ниже в просматриваемом списке, чтобы все существенные для системы задачи выполнялись первыми. Конечной целью всей системы приоритетов является минимизация длин очередей.

На рис. 14 показана последовательность просмотра, принятая в системе реального времени STARS фирмы UNIVAC. Эта система управления получена в результате дальнейшего развития системы Eastern Airlines/UNIVAC CONTORTS для машины UNIVAC 494. Для авиакомпании BEA (British



Р и с. 14.

European Airways) эта система выполняет ежедневно около 200 000 деловых операций, проводимых более чем с 600 терминалами. Эти терминалы рассредоточены по Европе и служат для выполнения деловых операций, касающихся резервирования билетов, управления пассажирскими и грузовыми перевозками, правильной загрузки самолетов с точки зрения распределения веса и т. д. Каждая операция делает по несколько обращений к файлам, расположенным на барабанах, и присутствует в машине в среднем по 1/2 секунды. Основная часть этого времени уходит на ожидание в очередях для получения различных ресурсов, и если бы цикл планировщика в какой-то мере был неэффективным, то очереди увеличивались бы и время выполнения операции экспоненциально росло. Это вызвало бы серьезную задержку в обслуживании терминалов и, как следствие, задержки в работе авиалиний. В дальнейшем мы увидим, что цикл планировщика, кроме того, что он следит за появлением событий, выполняет функции синхронизации периферийных и других действий (в дополнение к функциям, выполняемым аппаратным таймером), а также заведует использованием центрального процессора.

В основном существуют три типа задач, стоящих перед системой реального времени:

1. Управление периферийными устройствами.
2. Выполнение деловых операций.
3. Поддержание работоспособности системы и управление ее работой.

Задачи первого и третьего типов выполняются главным образом системой управления; задачи второго типа выполняются рабочими программами, осуществляющими всю содержательную обработку деловых операций. Существуют программы, управление которым передается непосредственно из цикла планировщика при обнаружении признаков соответствующих событий.

Приведем примеры задач вышеуказанных типов:

1. Управление аппаратурой связи, барабанами, дисками, магнитными лентами, вводом-выводом перфокарт и печатью.
2. Резервирование билетов; посылка сообщений; регистрация пассажиров и т. д.
3. Распределение ресурсов; планирование работы программ; процедуры восстановления; действия, выполняемые систематически через определенное время или в заданное время суток.

При выполнении деловых операций для организации работы центрального процессора используется своя многоуровневая структура приоритетов с очередями на каждом уровне.

Уровни приоритетов показаны на рис. 15. Если задача готова начать или продолжить свой счет (после завершения ввода-вывода или после того, как закончены более срочные работы,

Инициация и завершение работы «нетерпеливого» устройства	Задачи управления аппаратурой связи
Продолжение прерванных задач	
Завершение обслуживания	
Инициация и завершение работы «нетерпеливого» устройства	Рабочие программы реального времени
Продолжение прерванных задач	
Завершение обслуживания	
Инициация и завершение работы «нетерпеливого» устройства	Системные операционные задачи
Продолжение прерванных задач	
Завершение обслуживания	
Инициация и завершение работы «нетерпеливого» устройства	Фоновые задачи пакетной обработки
Продолжение прерванных задач	
Завершение обслуживания	
Инициация и завершение работы «нетерпеливого» устройства	Задачи трансляции и отладки
Продолжение прерванных задач	
Завершение обслуживания	
Резервные уровни	

Рис. 15.

из-за которых она была прервана), ее помещают в конец очереди соответствующего уровня и для цикла планировщика устанавливают признак, что очередь этого уровня не пуста. При наличии этого признака делается просмотр очереди рабочих программ и в результате для счета выбирается задача

с самым высоким приоритетом. Важно то, что, если программа закончила работу или достигла своей контрольной точки, цикл планировщика начинается сначала, с тем чтобы из задач, ожидающих своей очереди, выбрать задачу с высшим приоритетом.

9.4. Управление коммуникационной аппаратурой

Мы уже видели, что сеть каналов связи в больших системах реального времени очень сложная. Одна такая сеть может содержать различные терминальные устройства и дистанционные буферные устройства управления, иметь различные способы и скорости передачи, различные кодировки и стратегии опроса терминалов. Все это усложняет сеть и, таким образом, создает трудности в управлении ею. При создании многих супервизоров, рассчитанных на работу в реальном времени, существовала тенденция рассматривать связи как некоторое дополнение главной структуры, обладающее свойством нерегулярности в общем потоке управления. Вообще же говоря, желательно, чтобы структура системы управления создавалась на основе задачи управления связями, так как в конечном счете только эта задача может быть стержневой и управлять действиями системы на всех ее уровнях.

Главная задача Программы Управления Связью (ПУС) — обеспечить вторичное сглаживание неравномерности запросов на систему. Первичное сглаживание осуществляется дистанционными буферными устройствами, если они существуют. Хотя, как мы видели, система реального времени экзотична по своей природе, все же нельзя пускать сообщения в обработку без предварительного учета загруженности центрального процессора. ПУС работает по системе запросов, т. е. она запрашивает сообщение от дистанционного устройства. Без предварительного запроса ни одно сообщение не поступает в центральный процессор. Таким образом, если в некоторый момент времени центральный процессор сильно перегружен, ПУС на некоторое время просто перестает выдавать запросы. Некоторые сети связи используют систему «опроса состояния», при которой ПУС, прежде чем запрашивать сообщения, спрашивает устройство: «Есть ли у вас какое-нибудь сообщение?», на что устройство отвечает «да» или «нет». Это позволяет ПУС представить картину предстоящей загруженности системы до того, как ПУС запросит сообщения, но это может оказаться слишком дорогим удовольствием с точки зрения использования каналов связи. Существуют разные способы работы, когда несколько устройств

подсоединены к одной линии связи. В одних случаях запрос посылается одному устройству, которое в свою очередь опрашивает следующее, пометая сообщение своим идентификатором, и т. д.; последнее устройство возвращает сигнал в центральный процессор. В других случаях устройства опрашиваются индивидуально в круговом порядке или в более сложном порядке с учетом их загруженности.

Другой аспект ввода состоит в том, что сообщения с меньшими приоритетами можно накапливать на дисках или барабанах. Это значит, что поступающие в центральный процессор сообщения немедленно пишутся во вспомогательную память, еще до их обработки. Этот способ позволяет осуществлять третий уровень сглаживания.

Но какой бы способ управления вводом ни выбирался, всегда рассматривают, в какой степени он влияет на среднее время ответа, длину очередей, эффективность использования линий связи, устойчивость системы в условиях сбоев и пиковых нагрузок.

Что касается выводного сообщения, то оно обычно снабжается идентификатором терминала, на которое оно выдается. При этом также возможны разные способы. Система STARS, о которой говорилось выше, некоторые выводные сообщения накапливает во вспомогательной памяти, в то время как некоторые посылает сразу. Этот способ мотивируется дуплексной или полудуплексной природой линий связи.

9.5. Управление запоминающими устройствами с произвольным доступом

При работе с файлами в системе реального времени характерно то, что, во-первых, с одними и теми же файлами работают одновременно много пользователей и, во-вторых, многие файлы очень изменчивы. Последнее означает, что общая длина файла постоянно меняется от добавлений и выбрасываний записей при выполнении деловых операций. При образовании новой записи требуется найти для нее место в файле, и, следовательно, должны существовать методы управления свободным пространством файла.

Главная часть от общего времени пребывания деловой операции в обработке уходит на обращения к файлам (около двух миллионов обращений к файлам делается ежедневно в системе, обслуживающей авиакомпанию ВЕА), и нужно стремиться всеми возможными способами минимизировать это время. Для этого сначала тщательно выбирается аппаратура, а затем наиболее подходящая структура файлов. Часто возникает большое желание сделать структуру файлов

многоуровневой по типу дерева, что облегчает произвольный доступ к записям, но решение должно быть компромиссным, чтобы не слишком увеличить количество физических обращений на одну запись. Часто используемые индексы, если они не помещаются в оперативной памяти, должны храниться на быстрых барабанах по возможности с неподвижными головками чтения/записи, а редко используемые записи можно помещать на менее быстрые устройства с двигающимися головками. Программа Управления Случайным Доступом (ПУСД) должна оптимизировать движения головок на больших устройствах, так как на это уходит большая часть времени.

Следующий вопрос — это поиск компромисса между степенью защищенности данных и количеством проверок, выполняемых рабочими программами и ПУСД. Существует верхняя граница для этих проверок, чрезмерность которых может серьезно увеличить время ответа. Существует другая граница, ниже которой также нельзя опускаться, так как отсутствие необходимых проверок сделает файлы очень уязвимыми.

Возможность испортить файл возникает, например, когда две независимо обрабатываемые деловые операции требуют обращения к одной записи. Если разрешить это, то могут сформироваться разные изменения для одной и той же записи, а в результате запишется только одно из них, так как последнее «забыет» первое. Чтобы избежать этого, в ПУСД должна существовать возможность воспринимать сообщение такого вида: «Прочти запись X и запрети ее использование другими программами». Должна также существовать возможность выдать противоположное сообщение: «Открой запись X для общего пользования». В некоторых случаях возникает необходимость временно закрыть доступ к файлу, например, для того, чтобы над ним можно было проделать процедуру уплотнения, но это можно делать только с редко используемыми файлами или только в том случае, если на этот период основная работа системы приостановлена.

Мы уже видели, что для надежности файлы часто дублируются. Если эти файлы соединены с отдельными каналами ввода-вывода, то можно также получить выигрыш во времени. Наличие файлов-двойников означает, что чтение можно осуществлять по любому из двух каналов, в то время как запись должна производиться по обоим. Основываясь на теории очередей, можно показать, что если отношение числа чтений к числу записей составляет 50% и более, то для уменьшения средней длины очереди выгодно иметь более од-

ного канала. Конечно, ПУСД планирует свою работу на основе одновременного обновления обеих копий файла. Если на части аппаратуры, связанной с одной копией файла, проводятся работы по профилактическому обслуживанию, то существует другая копия файла, позволяющая продолжить текущую работу в реальном времени, хотя и на более низком уровне надежности. Система управления сама восстановит копию файла для работы после профилактики, и это будет сделано без остановки системы.

Другой часто используемый способ защиты состоит в том, что все обновляемые записи файла копируются на другой носитель (на диски или магнитную ленту). При отказе в работе устройства файлы можно воссоздать по последнему дампингу файла и копии обновленных записей. Но это, конечно, требует времени.

9.6. Управление рабочими программами и распределение ресурсов

Для обработки любого поступившего в центральный процессор сообщения требуются те или иные ресурсы. Необходимо место в оперативной памяти для самого сообщения, для буферов, используемых в качестве рабочих областей, для хранения участвующих в данной обработке записей; необходим также некоторый отрезок процессорного времени для выполнения рабочей программы. О необходимости обращений к файлам говорилось в предыдущем параграфе.

В большинстве случаев несколько однотипных сообщений одновременно требуют одной и той же обработки. Для того чтобы в оперативной памяти не иметь несколько копий одной и той же программы, ее делают реентерабельной (re-entrant — буквально «повторно-входимой»). Свойство реентерабельности позволяет обрабатывать не последовательно, а параллельно в режиме разделения времени сразу несколько деловых операций с помощью всего одной копии программы в памяти. Когда вычислительные работы в основном лимитированы обменом, такое разделение времени помогает продвигать очередь заявок на ввод-вывод, а главное, удовлетворять запросы на обмен с жизненно важными файлами на устройствах с произвольным доступом. При эффективной системе управления таким путем можно свести к минимуму задержки в общем потоке операций ввода-вывода. Описанная методика требует, чтобы для каждой конкретной деловой операции была отведена своя персональная рабочая область, причем все эти области должны быть отделены от программ,

производящих обработку. Таким образом, если управление от одной деловой операции временно переходит к другой, то все регистры должны сохраняться в ее персональной области.

Управляющая программа отвечает за выделение памяти для этих рабочих областей и сохранность данных в них. В большинстве современных вычислительных машин имеется система защиты оперативной памяти, которая исключает ситуацию, когда нарушаются границы чужих буферов, но такая защита становится сложной при динамическом распределении оперативной памяти. Наиболее эффективный метод управления оперативной памятью реализуется с помощью алгоритма, который выделяет область любого размера из общего резерва свободной памяти при поступлении запроса на память и возвращает эту область в резерв, когда рабочие программы от нее отказываются. Если не принимать никаких мер, то этот метод ведет к расчленению (фрагментации) общей резервной области, и обычно при этом исходят из предположения, что буфера для отдельной деловой операции выделяются не обязательно в связной области. Другой метод, при котором память выделяется листами фиксированных размеров, слишком расточителен. Рабочие программы к своим буферам всегда адресуются через индекс-регистры, поэтому при передаче буфера в пользование рабочей программе ей сообщается его базовый адрес.

Многие рабочие программы разбиваются на сегменты, которые по мере надобности загружаются в оперативную память с барабана. Система управления должна удовлетворять запросы на сегменты и либо ставить задачу в очередь, если реентерабельный сегмент уже находится в оперативной памяти, либо загружать сегмент в соответствующее место и активизировать. Желательно, чтобы его можно было загружать в любую область достаточных размеров.

9.7. Работа таймера

Ясно, что фактор времени очень важен в системе реального времени. Система должна всегда быть в курсе времени при организации функционирования своей деятельности. Некоторая служба времени реализуется аппаратно, часто существует несколько часов — одни указывают текущее время суток, другие вызывают прерывание по истечении заданного интервала времени. Эти прерывания система управления использует, во-первых, для планирования некоторого вида задач, которые необходимо выполнять в определенное время суток, и, во-вторых, для постоянного слежения за временем всех действий системы в целях контроля.

Существует много программ, которые нужно пускать в определенное время суток, например дампинг, чистка и просмотр файлов. Вместо того чтобы заставлять оператора вызывать эти программы в соответствующее время (причем ошибка в несколько секунд при этом может быть существенной), эта работа может выполняться системой управления, которая, например, каждую минуту может проверять список таких программ и вызывать нужные. Можно также потребовать, чтобы данные об использовании системы периодически выводились на пультовый телетайп или печатающее устройство для статистики.

Обычно нет необходимости переносить специальные требования коммерческих систем реального времени на программы, обрабатываемые пакетным способом, и выделять им кванты времени для работы, если они не требуют слишком большого счета. Приоритеты рабочих программ, показанные на рис. 15, гарантируют, что обработка деловых операций в реальном времени не будет игнорироваться. Существует постоянный поток прерываний от внешних источников, который позволяет системе управления с помощью проверки приоритетов на каждом прерывании избежать нарушения программами пакетной обработки естественного течения работы в реальном времени. Программа может заикнуться, и, если не контролировать это, такое заикливание может остаться незамеченным и повредить данные. Чтобы избежать подобной ситуации, система управления всякий раз, когда управление переходит к ней, сравнивает текущее время программы с максимально возможным временем счета, которое задается в качестве параметра программы в момент ее загрузки. Если текущее время превысило максимально допустимое, то либо программа выбрасывается, либо о возникшей ситуации сообщается операторам на тот случай, если они захотят предпринять какие-то другие действия. Для повышения безопасности можно также предусмотреть проверки максимального времени счета между обращениями к управляющей системе за обслуживанием.

Сами системы управления также не застрахованы от ошибок, так что контроль за временем должен распространяться и на их внутренние действия. Большинство действий может прерываться, и если выполнение какого-то действия не укладывается в отведенное время, то об этом посылается сообщение операторам для соответствующего анализа и возобновления работы.

Для выявления подозреваемых узких мест в системе можно учитывать время пребывания заявок в очереди за ресурсами. Может накапливаться статистика числа случаев, когда

времена превышают установленные пределы, с тем чтобы создатели системы могли проверить свои первоначальные оценки. И конечно, должно контролироваться время работы периферийных устройств, так как случайная «потеря» устройства в режиме on-line другим способом не обнаруживается.

9.8. Отказы в работе и их устранение

При возникновении отказа в работе системы пакетной обработки обычно имеется возможность повторить программу с некоторой контрольной точки, и это скажется только на потери некоторой части от общего времени. К сожалению, системы реального времени более чувствительны к задержкам, которые могут привести к нарушению обслуживания пользователей. Течение времени неумолимо, и, если система застопорилась, выполнение некоторых существенных функций может быть нарушено. Более того, когда возникает отказ, некоторые операции могут быть выполнены лишь частично, и, если не сделать строгого и полного анализа прежде, чем продолжить обслуживание, существует большая вероятность того, что записи данных будут испорчены, так как одна и та же операция коррекции будет произведена дважды.

И программы управления, и рабочие программы ответственны перед системой за сохранность данных и за быстроту восстановления работы. Методы, используемые для этой цели, зависят от специфики каждой конкретной системы, но главное внимание всегда уделяется сохранности файлов и возможности продолжить выполнение деловых операций. Очень важно сообщать пользователю о поломке системы (отсутствие ответа не всегда достаточно ясно говорит ему об этом), а также иметь возможность попросить пользователя проделать специальные действия, требующиеся для восстановления работы. Не менее важно иметь диагностику, позволяющую выявить неисправность.

В связи с интенсивной диалоговой работой в системе реального времени важно как можно быстрее обнаруживать появляющиеся неисправности. Аппаратура многих систем вызывает прерывание в таких ситуациях, как нарушение четности памяти, скачки напряжения, слишком долгая работа при заблокированном прерывании, нарушение защиты памяти, несуществующий код операции, переполнение таймера и т. д. В такие моменты система управления должна убедиться, что ситуация допускает восстановление, и проделать соответствующие восстановительные действия. Они могут включать временное прекращение некоторых или всех опера-

ций реального времени, пока процесс восстановления не будет закончен. Возможно даже, что состояние системы будет восстанавливаться с помощью сделанного ранее дампинга состояния системы (такие дампинги делаются регулярно). Мы уже видели, что в больших системах некоторая аппаратура дублируется, что позволяет продолжить обработку при возникновении аппаратной ошибки, но если ошибка программная и регулярно повторяется, то, прежде чем продолжать обработку, нужно проанализировать ошибочную ситуацию.

Управляющие программы не всегда могут записать состояние системы перед остановкой, поэтому необходимо достаточно часто в процессе нормальной работы регистрировать состояние системы (записывать контрольные точки), чтобы облегчить впоследствии ее восстановление. Система UNIVAC STARS пишет на два носителя управляющие таблицы, связанные с управлением файлами и состоянием аппаратуры связи с интервалом в пять секунд, так что в случае отказа потребуются ограниченный объем восстановительных работ.

При анализе ошибки и восстановлении системы предпринимаются следующие действия (производимые вручную или с помощью программ): вся информация о динамическом состоянии системы тщательно проверяется, с помощью программ проверяется правильность подключения аппаратуры, дистанционные буферные устройства оповещаются о создавшейся аварийной ситуации, проверяется достоверность файлов, строятся заново управляющие таблицы, подготавливается продолжение неоконченной обработки сообщений, вызываются программы анализа и восстановления. Когда все это будет сделано, вновь будут опрошены линии связи и снова начнется обработка операций реального времени. На весь восстановительный процесс уходит меньше минуты.

9.9. Заключение

Итак, главные требования к управляющим программам системы реального времени следующие:

1. Особое внимание вопросам связи.
2. Управление быстро меняющимися файлами.
3. Эффективное планирование на нижнем уровне.
4. Постоянный учет времени.
5. Жизнеспособность в аварийных ситуациях.

Само собой разумеется, что накладные расходы при выполнении супервизорных работ должны быть сведены к минимуму. Это требование означает, что многие функции пакетной обработки были бы обременительны для системы реального времени. Такие возможности, как планирование на

высоком уровне, языки для организации потока работ, общий архив данных, сложные методы управления файлами, разговорные компиляторы и т. д., ослабили бы эффективность системы реального времени. Однако, поскольку современные процессоры обладают очень высокой скоростью, а требования пользователей к вычислительным машинам становятся все более многообразными, можно ожидать, что различия между системами будут уменьшаться. Уже было предпринято несколько попыток получить универсальные операционные системы, но особой выгоды от применения систем широкого профиля получено не было.

ОПЕРАЦИОННАЯ СИСТЕМА УРОВНЯ J

Мартин Уорвик

Системами реального времени мы закончили рассмотрение всех основных аспектов операционных систем. Но для иллюстрации в каждом отдельном случае использовалась только одна система. Мартин Уорвик дополняет общую картину, описывая для сравнения систему на другой машине, System 4, имеющей архитектуру, типичную для машин поколения IBM/360. Он описывает супервизор, управление данными, управление работами и систему отладки программ и, сравнивая эту операционную систему с системами, описанными в предыдущих главах, демонстрирует большое разнообразие методов, используемых для решения в конечном итоге одних и тех же основных проблем. Оценить пути дальнейшего развития операционных систем мы предоставляем читателю.

10.1. Введение

В предыдущих главах описание универсальных операционных систем главным образом относилось к системе для машин серии ICL 1900. Каждая машина или по крайней мере серия машин имеет свою собственную систему, и в этой главе для сравнения описывается система, предназначенная для другого ряда машин.

На разработку операционной системы, удовлетворяющей перечисленным в начале книги общим требованиям, оказывают влияние много различных факторов. Ясно, что аппаратные возможности влияют на проект и методологию управления системы, но, вероятно, самым важным фактором является образ типичного пользователя. Если в среднем работа выполняется с помощью большой программы длительной обработки данных, то накладные расходы от операционной системы относительно невелики. Система может позволить себе богатый набор возможностей. Если же, напротив, типичной работой будет выполнение короткой программы, написанной студентом, то накладные расходы от системы имеют очень большое значение и, вероятно, представляют большую часть от общего объема затрат.

Если система специализируется на работах какого-то определенного типа, то именно для этих работ желательно увеличить ее пропускную способность. Систему, описанную в этой главе, можно назвать операционной системой общего назначения в том смысле, что она предназначена для выполнения широкого ассортимента работ и предоставляет свои

возможности для всех пользователей, не ориентируя себя на какого-то конкретного пользователя. Эта система разработана той же фирмой ICL, но уже для машин ряда System 4. Она известна под названием операционная система уровня J и предназначена для работы на больших моделях ряда (4-40, 4-50, 4-70). Это дисковая система, т. е. она предполагает использование дисков. В своем основном варианте она ориентируется на 65К байтов оперативной памяти (32 бита=4 байта=1 слово), 2 устройства со сменными дисками, устройство ввода карт и печатающее устройство. Эта конфигурация может быть дополнена оперативной памятью, устройствами с постоянными дисками, магнитными лентами, барабанами и другими периферийными устройствами, включая коммуникационное оборудование. Для более детального рассмотрения системы мы выделим в ней следующие части:

Супервизор.

Система управления данными.

Управление работами.

Система отладки программ.

10.2. Отличия от системы 1900

Первое отличие между системой уровня J и системами, описанными в предыдущих главах, состоит в том, что многие функции, выполняемые в системах GEORGE 1, 2 и 3, в системе уровня J выполняются супервизором. Перевод некоторых «системных» функций на супервизорный уровень имеет то преимущество, что отпадает необходимость выполнять некоторые операции дважды: один раз (для обеспечения единообразия) в операционной системе и другой раз в рамках супервизора. Возражение могло бы касаться структуры полученной при этом программы: насколько она монолитна? Насколько новый супервизор прост для понимания и модификации? В действительности именно структурные вопросы и заставляют нас придерживаться принципа «модульности», который приводит к тому, что супервизор вместе с центральным управлением разбивается на множество отдельных сегментов. При этом становится сравнительно легко изменять старые и добавлять новые возможности.

Любая система имеет внутреннюю структуру, задаваемую ее создателями, и внешнюю структуру, представляющую набор возможностей для пользователя. Внутренняя структура и эффективность системы в определенной степени зависят от аппаратных возможностей. System 4 на уровне пользователя имеет систему команд, совместимую с IBM/360. В то же время ее архитектура на уровне привилегированных команд, т. е.

аппаратный интерфейс, скрытый от пользователя операционной системой, имеет существенные отличия. Существуют четыре состояния процессора: P1, P2, P3, P4, каждое из которых определяется набором аппаратных регистров; у каждого состояния имеется свой набор. Прерывание, вызванное неправильной работой аппаратуры, переводит процессор в состояние P4 (отключение питания, ошибка по четности в памяти). Все остальные прерывания вызывают переход в состояние P3. Состояния P1 и P2 — это обычные программные состояния. Они используются системой уровня J соответственно для выполнения программ пользователей и супервизорных программ.

Другое существенное различие между System 4 и машинами серии 1900 лежит в управлении внешними устройствами. System 4, как и IBM/360, управляет вводом-выводом с помощью цепочек команд. Каждый канал ввода-вывода может интерпретировать цепочку или последовательность команд, которая управляет движениями магнитной ленты, перемещениями головок на дисках, передачей данных и другими действиями. Используя это, можно с помощью программы канала ввода-вывода осуществлять ряд операций на внешних устройствах, причем действия эти выполняются независимо от центрального процессора и параллельно с его работой. С помощью цепочек команд можно составлять очень сложные процедуры, что особенно важно для полного использования возможностей запоминающих устройств с произвольным доступом. Только благодаря операционной системе удастся воспользоваться всей совокупностью этих возможностей, сохранив простоту и прозрачность для пользователя.

10.3. Супервизор

Мы уже говорили о четырех состояниях процессора. Вход в супервизор осуществляется по машинному прерыванию, в результате чего одна из подпрограмм выполняется процессором в состоянии P3. Эта подпрограмма называется Ядром и является центральной программой для анализа прерываний. На нее возлагается как можно меньшая часть работы, хотя наиболее часто работающие программы — программы, обслуживающие ввод-вывод, — выполняются в ней полностью. Ядро работает как переключатель, и его основная задача состоит в том, чтобы проанализировать состояние системы и определить другие программы системы, которые должны проработать в данной ситуации. Его операции занимают мало времени; это важно, так как Ядро является одной из частей операционной системы, которая работает без прерываний. Закончив свои операции, Ядро просматривает список работ, готовых к обработке

процессором в состоянии P1 или P2, и вызывает из них работу с высшим приоритетом. Этот список работ, задач или подпрограмм называется *таблицей программ* (Routine Table).

Работы, обрабатываемые в состоянии P1, — это главным образом работы пользователя. В состоянии P2 выполняются подпрограммы, реализующие возможности супервизора. Все они могут прерываться и, если не считать права употреблять привилегированные команды, ничем существенным от программ обычного пользователя не отличаются. Важнее всего то, что все они сами могут использовать возможности супервизора, обращаясь к ним с помощью обычной команды обращения к супервизору.

Такая структура супервизора имеет очень большое значение, и следующие четыре пункта подчеркивают ее преимущества:

1. Ядро является единственной неделимой интегральной частью операционной системы, но она невелика по размеру.
2. Все возможности системы, за исключением Ядра, реализуются в виде отдельных модулей.
3. Каждая подпрограмма может прерываться и, следовательно, сама может использовать любые возможности супервизора. Если одна подпрограмма желает использовать возможности, реализованные другой подпрограммой, она использует в точности то же обращение к супервизору, что и обычный пользователь.
4. Очень просто заменить одну возможность на другую или добавить новую. Ядро оперирует с данными, заключенными в таблицы. Поэтому вопрос получения новых возможностей — это в основном вопрос занесения нужной информации в соответствующие поля данных. Логика работы Ядра зависит только от содержимого полей в таблицах.

По своей сути супервизор системы уровня J является мультипрограммным супервизором. В мультипрограммировании участвуют не только программы пользователя, но и сами супервизорные программы. В мультипрограммировании допускается до четырнадцати уровней программ пользователя. При выборе программы для дальнейшей работы используется жесткая приоритетная система. Подпрограммы супервизора имеют более высокий приоритет, чем программы пользователя.

До сих пор мы обсуждали внутреннюю структуру супервизора, и наш интерес определялся его принципиальными отличиями от супервизоров других операционных систем. А сейчас обратимся к внешней структуре супервизора и посмотрим на возможности, которые он обеспечивает.

10.3.1. Связь с оператором

Пункт управления на машине System 4 состоит из пультового телетайпа и некоторых других органов управления. Если в машине имеется несколько потоков (см. 10.5), то можно использовать несколько пунктов управления. Вводя сообщения через пультовый телетайп, оператор управляет работами, которые выполняются на машине, и получает сообщения от операционной системы о ее состоянии и о тех действиях, которые от него требуются (например, установить пакет дисков или магнитную ленту). Подпрограмма, управляющая пультом, работает асинхронно с остальной обработкой, и это существенно, так как относительно медленный телетайп не должен задерживать работу системы. Выводные сообщения образуют очередь и непрерывно печатаются со скоростью работы телетайпа. Для посылки вводного сообщения в любое время можно прервать выдачу выводных сообщений на телетайпе.

Системный журнал хранится на диске и содержит сведения о проделанных системой работах. Журнал состоит из одного или более файлов на диске, которые доступны для программы анализа и могут быть напечатаны.

10.3.2. Управление вводом-выводом

Управление вводом-выводом и организация доступа к файлам составляют большую часть всей работы супервизора. Все операции с внешними устройствами рассматриваются как операции с файлами. Колода карт, выдаваемые на печать данные, которые читаются или пишутся на диски, магнитные ленты и т. д. — все они считаются файлами. Большинство файлов имеют метки, хотя это не обязательно. Прежде чем использовать любое внешнее устройство, необходимо открыть файл. Директива OPEN выполняет функции размещения файла, проверяет заголовок, чтобы удостовериться, что именно этот файл требует пользователь, осуществляет поиск начальной зоны в случае, например, магнитной ленты, заносит информацию о физическом расположении файла в соответствующие таблицы супервизора и помечает, что файл открыт. Обратная директива CLOSE закрывает доступ к файлу.

OPEN и CLOSE играют двоякую роль. Во-первых, они дают разрешение на пользование внешним устройством некоторым определенным способом. Супервизор только тогда обеспечивает доступ к файлу, когда убедится, что он открыт

для пользования. Во-вторых, они дают возможность пользователю написать в программе всего-навсего

OPEN FRED

чтобы подготовить файл для пользования и установить его в нужное положение. Имя FRED, которое появляется в программе, не является действительным именем файла, которое хранится на магнитной ленте или диске. Это лишь символическое имя файла внутри программы. И только во время выполнения работы это внутреннее имя связывается с действительным именем файла.

После того как открыт файл, супервизор решает много других задач.

- а) Проверяет корректность запроса на передачу данных, а также наличие разрешения на использование данного устройства.
- б) Организует очередность в выполнении запросов на передачу данных, если один канал обслуживает несколько внешних устройств.
- с) Для устройства с произвольным доступом проверяет, чтобы передачи данных не выходили за границы файла.
- д) В случае успешного окончания передачи выдает сообщение об этом программе пользователя, и если существует на очереди следующая передача, то она производится.
- е) В случае сбоев при передаче данных выполняет соответствующие операции по восстановлению. Для магнитной ленты или дисков передачи повторяются несколько раз. Особого внимания требуют к себе ошибки в работе дисков, вызываемых загрязнением поверхности дисков. Некоторое количество дорожек на диске можно забраковать и заменить другими. На супервизор возлагается переключение с плохой дорожки на хорошую.

Система уровня J допускает установку магнитных лент и пакетов дисков на любое свободное устройство, и операционная система автоматически распознает их по метке тома, а не по номеру устройства.

До сих пор речь шла только о местных устройствах. Дистанционные устройства, такие, как телетайпы, дисплеи, терминалы и т. д., управляются с помощью «пакета управления телекоммуникациями», который является необязательной частью супервизора и устанавливается по желанию потребителя. Для программ пользователя супервизор образует интерфейс, за которым скрываются индивидуальные характеристики устройства. Многие из этих устройств подсоединены

к линиям связи, обслуживающим по несколько устройств. Пользователь же благодаря этому пакету имеет дело с простой передачей данных на любое устройство (или с любого устройства).

10.3.3. Действия программы

Ошибки в программе пользователя, запрещенные операции, обращения в чужие области памяти и т. д. вызывают передачу управления супервизору, который запоминает регистры пользователя и входит в так называемую STXIT-программу. Эта подпрограмма организуется самим пользователем в его программе для анализа ошибок и в некоторых случаях позволяет продолжить работу. Пользователь может указать адреса для следующих трех подпрограмм:

1. Подпрограмма восстановления счета, позволяющая пользователю продолжить вычисления, если ошибку можно исправить.
2. Подпрограмма окончания счета по ошибке, позволяющая пользователю должным образом прекратить счет, если ошибку нельзя исправить.
3. Подпрограмма связи с оператором. При соответствующих аппаратных возможностях оператор может разговаривать с выполняющейся программой, прерывая машину в любое время с пульта и посылая сообщение, адресованное данной работе.

Операционная система допускает, чтобы программа состояла из нескольких перекрывающихся сегментов, каждому из которых присваивается свой номер. Работа по настройке сегментов на конкретные адреса памяти незначительна.

Супервизор предоставляет возможность любой программе указать контрольную точку, в которой она может потребовать сформировать запись. Запись эта, содержащая текущее состояние работы, включая состояние оперативной памяти, запоминается во вспомогательной памяти. Впоследствии работу можно начать с этой контрольной точки с помощью предложения RESTART на языке управления работами.

Любая работа завершается обращением к подпрограмме Конец Работы или принудительной передачей ей управления из системы в случае возникшей в программе пользователя аварийной ситуации. При этом все незакрытые файлы закрываются, все устройства и оперативная память, используемые данной работой, объявляются свободными, что учитывается при новом планировании, сообщения об этом посылаются на пультный телетайп оператору и в системный журнал для

статистики. Освобождение ресурсов машины осуществляется специальной подпрограммой Освобождение Ресурсов (Deallocator), инициирующей другие подпрограммы, которые попытаются загрузить следующую работу из очереди, установленной подпрограммами управления работами.

10.3.4. Генерация системы

Процесс формирования супервизора для каждой конкретной установки называется *генерацией системы*. Полная система выпускается изготовителем на магнитной ленте (или на дисках, если на установке нет магнитных лент). Из этого полного набора системных программ процесс генерации порождает супервизор и размещает его на дисках, которые на данной установке и становятся новыми системными дисками. В процессе генерации супервизор настраивается по многим параметрам. Основные из них перечисляются ниже. Данные, описывающие требуемую систему и позволяющие провести настройку, вводятся с перфокарт.

1. Таблицы супервизора загружаются информацией, отражающей конфигурацию машины, т. е. количество и физические адреса внешних устройств, размер оперативной памяти, описание способа использования устройств и, в частности, деление машины на потоки для управления работами. (Понятие потока будет пояснено в 10.5).
2. Задается набор возможностей, определяемых по желанию заказчика, от незначительных, таких, как дополнительные пульта управления, до очень важных, таких, как пакет управления телекоммуникациями.
3. Задаются количество уровней, допустимых в мультипрограммировании, а также относительные приоритеты супервизорных подпрограмм.
4. Супервизорные подпрограммы по желанию могут постоянно быть резидентными или вызываться из внешней памяти по мере надобности (некоторые по логике их использования всегда должны быть резидентными). Отбираются резидентные программы и устанавливается число областей для вызова нерезидентных программ.

10.4. Управление данными

Мы уже установили, что любой набор данных, который можно прочитать с внешнего устройства или записать на него, рассматривается как файл. В управление работой с файлами входит:

1. Задание в программе типа файла и способа доступа к нему. Это задание представляется в виде таблицы определения файла, задаваемой, например, макрокомандой DTFSR (Define The File for Serial Records — определить последовательный файл). Символическое имя файла является меткой Таблицы Определения Файла.
2. Программа, которая преобразует логические запросы на работу с файлами в физические команды для супервизора.
3. Карты управления работами, сопоставляющие программные символические имена файлов с их физическими именами.
4. Управление супервизора, который следит за правильностью передачи данных и за распределением емкости канала между пользователями.

Система управления данными включается в программу пользователя следующим образом. Пользователь в своей программе определяет по одной Таблице Определения Файла на каждый файл, к которому он хочет обращаться. После определения всех DTF-таблиц пользователь вставляет в программу макрокоманду DTFEN. Ассемблер, встретив эту макрокоманду, сгенерирует в этом месте программу, обеспечивающую те виды работ с файлами, которые были заказаны при предварительном задании Таблиц Определения Файлов. В случае языков высокого уровня используется более простой для пользователя способ, являющийся по существу по своей внутренней организации тем же способом.

Сложность операций, которые возможны при использовании этих пакетов управления файлами, главным образом зависит от типа устройства, к которому адресуются. Из всех операций управления данными самыми сложными являются операции, имеющие дело с устройствами с произвольным доступом. Приведем в качестве примера четыре способа доступа:

1. *Физический доступ* (DTFPH — Physical Handling). При этом способе пользователь проделывает сам всю работу по написанию цепочек команд и передает их супервизору, используя EXDP-макрокоманду (Execute Disc Program).
2. *Последовательный доступ* (DTFSR—Serial Records) Файл может состоять из блоков постоянной или переменной длины. В каждом блоке могут быть записи постоянной или переменной длины. Пользователь, определив структуру файла по Таблице Определения Файла, получает последовательный доступ к записям с помощью макрокоманд GET и PUT.

3. Прямой доступ (DTFDT — Direct Access)

В этом случае пользователь может иметь доступ к физическим блокам произвольным или последовательным способом. Необходимые для обмена с дисками цепочки команд могут быть очень сложными, их создаст система для пользователя, которому достаточно знать только логическую структуру данных в файле.

4. Индексно-последовательный доступ (DTFIS — Indexed Random and Sequential Processing)

К записям можно обращаться произвольно или последовательно от начальной записи не с помощью физических адресов, а с помощью ключей, содержащихся в записях. Этот способ позволяет не только читать записи, но также их заменять, вставлять и выбрасывать в соответствии со значениями ключей.

Система управления данными такова, что пользователю предоставляется возможность оперировать со своими данными как в терминах физических адресов, так и в терминах записей, предоставляя операционной системе право самой позаботиться об их физическом расположении.

Используя более сложные методы доступа (в тех случаях, когда это возможно по логике работы), программа становится независимой от физического расположения данных и может использоваться для работы с устройствами различных типов.

До сих пор при описании системы файлов рассматривались только стандартные файлы. Они известны как *постоянные* (Dedicated) файлы, так как места, которые они занимают, отданы им в постоянное пользование и представляют собой связанные области или по крайней мере блоки связанных областей. Внутри системы используется дополнительная система файлов на дисках, известная как DTFPA (Define The File for Partitioned Access), которая динамически, дорожка за дорожкой, запрашивает или освобождает место в памяти. Системные файлы с коротким временем существования по своей природе тяготеют именно к этому типу файлов, непрерывно изменяющихся по объему. Такие файлы называются *динамическими* (Non-Dedicated), и хотя в операционной системе уровня J они используются только для системных нужд, в операционных системах Multijob и системах уровня R с ними может работать также и пользователь. Динамический файл состоит из набора связанных ссылками дорожек, причем ссылки хранятся на самих же дорожках. Информация о свободных дорожках в системе уровня J сведена в специальный список. Все файлы системы отладки динамические, так как они постоянно

меняются в процессе исправлений разрабатываемых и отлаживаемых программ, что избавляет нас от проблем фрагментации на дисках.

10.5. Управление работами

Управление работами пользователей осуществляется с помощью нескольких подпрограмм:

1. Подпрограмма Ввода Работы (Job Input)

Эта подпрограмма вводит описания работ, читая их с карт или с бумажной ленты, и запоминает данные на системных дисках.

2. Планировщик (Sheduler)

Эта подпрограмма включается в работу, как только описание какой-либо работы полностью введено или освободелся какой-нибудь из ресурсов машины (например, память или внешнее устройство). Она просматривает список работ, содержащихся на системных дисках в порядке их приоритетов, чтобы найти подходящую работу для загрузки в машину.

3. Распределитель Ресурсов (Allocator)

После того как Планировщик выбрал работу для загрузки, Распределитель Ресурсов выполняет действия по выделению ей оперативной памяти и внешних устройств. Распределитель иницирует загрузку работы и затем передает ее для выполнения под контролем супервизора. После этого снова вызывается Планировщик, чтобы попытаться найти другую работу для загрузки в машину.

4. Программа Освобождения Ресурсов (Deallocator)

Она работает, когда освобождаются ресурсы машины, например при завершении работы или при отказе от устройства, если известно, что от него не потребуется больше никаких операций. Далее управление переходит Планировщику.

Управление работами системы уровня J может следить за автоматической загрузкой более 100 работ. Порядок загрузки и работа используемой системы приоритетов основываются на понятиях *потока*, *ранга* и *приоритета*. Потоки делят машину на части так, что каждый поток имеет свою часть оперативной памяти, свои внешние устройства, и по желанию пользователя ему может быть выделен свой операторский пульт управления. Машина может иметь от одного до шести потоков. Никакая работа не должна превышать ресурсы своего потока и не может одновременно участвовать в не-

скольких потоках. Но внутри каждого отдельного потока количество работ не ограничено. На практике одни пользователи организуют только один поток в том случае, если какие-то из работ используют все ресурсы машины, другие строго ограничивают работы размерами потоков и используют машину как несколько независимых вычислительных систем.

Каждой работе присваивается поток (A—F), ранг (0—9) и приоритет (1—14). Ранг управляет порядком, в котором работы отбираются для счета внутри потока. Может случиться, что из нескольких работ потока, ожидающих своей очереди, работа с более низким рангом будет обеспечена ресурсами раньше работы с более высоким рангом. Если это происходит, то ранг работы, которую опережает работа с более низким рангом, увеличивают на 1. Это наращивание ранга делается до тех пор, пока он не достигнет максимального значения, равного девяти. Работу с рангом 9 выполняют сразу, как только освободятся нужные для нее ресурсы. В этом случае работа с рангом 9 выполняется максимально быстро, насколько это возможно, и так как у большинства других работ ранг со временем приближается к 9, никакая работа не может постоянно игнорироваться и «застрять» в системе.

Ранги 0 и 1 играют особую роль. Ранг 0 используется для работ, которые находятся как бы в состоянии «спячки», так как он не растет и никакая работа с таким рангом не пускается в счет. Он используется для управления последовательностью выполнения работ, поскольку ранг может быть изменен оператором с пульта или по команде обращения к супервизору, заданной в выполняемой программе. Программа ввода может ввести последовательность работ, в которой первая имеет «соответствующий» ранг, а остальные имеют ранг 0. Последующие работы из последовательности вводятся в действие с помощью изменения их рангов в соответствии с завершением счета предыдущих работ. Особенностью ранга 1 является то, что он никогда не меняется и планировщик не пытается запустить в счет следующую работу с рангом 1, пока не удовлетворит данную, т. е. работы с рангом 1 выполняются строго в том порядке, в каком они поступают на машину.

Аппарат потоков и рангов сложен, но достаточно гибок, чтобы удовлетворить требованиям большинства пользователей. Думается, что немногие пользователи будут использовать все шесть потоков и десять рангов. Многих пользователей устроит один поток и присвоение всем работам ранга 1, чтобы загружать их в память в порядке поступления.

Приоритеты (от 1 до 14) служат для распределения времени центрального процессора, когда несколько задач выполняются в мультипрограммном режиме. Приоритет 14 — наивысший.

10.6 Система отладки (The Trials System)

В систему отладки входят все пакеты программ, помогающие пользователю создавать и отлаживать свои программы. Она рассматривается как обычная работа пользователя и вводится в действие с помощью предложения JOB TRIALS. Главные составные части системы: *Редактор текста* (Amender), *Ассемблер*, *Компиляторы и Композитор* (Composer). Вся информация о создаваемой и отлаживаемой программе хранится на дисках, включая исходный текст. В разных стадиях обработки программа может быть в четырех видах:

1. Исходный текст, или *исходный модуль*.
2. Представление после компиляции, или *объектный модуль*.
3. Представление после компоновки (редактирования связей), или *сегмент загружаемой программы*.
4. Окончательный вид после загрузки в соответствующее место памяти.

Исходные модули в виде образов карт с «упакованными пробелами»¹⁾ хранятся на дисках, и с помощью Редактора текста можно выполнять замены, вставки, добавления и выбрасывания различных предложений исходного текста. Исходный модуль после переработки его компилятором (имеются компиляторы с языков FORTRAN, ALGOL, COBOL и CLEO) или Ассемблером превращается в объектный модуль, не представляющий еще программу в окончательном виде, но имеющий вид, удобный для компоновки, т. е. объединения его с другими ранее полученными модулями. Объектный модуль имеет относительные адреса для того, чтобы его можно было загружать в любое место памяти, и ненастроенные внешние ссылки. Теоретически любые два модуля, написанные на любых языках, могут вместе компоноваться. Однако это возможно, если только модули, написанные на разных языках высокого уровня, подчиняются определенным соглашениям.

Композитор имеет список объединяемых объектных модулей, а также информацию о структуре получаемой программы. Можно задавать различные оверлейные структуры (структуры, для которых некоторые части программ перекрывают друг друга в памяти). Каждая загружаемая единица

¹⁾ В целях экономии места подряд стоящие пробелы заменяются одним байтом, в котором указывается их количество. — *Прим. перев.*

представляет собой сегмент программы. Эти сегменты имеют настроенные внешние ссылки, и только адресные константы еще не имеют своего окончательного вида. Прежде чем пустить программу в счет, относительные адреса в этих константах нужно заменить на значения абсолютных адресов памяти. Это делает специальная программа супервизора — Загрузчик — при загрузке сегмента в память.

Программы для System 4 пишутся в виде набора исходных модулей. Во время отладки нет необходимости каждый раз перетранслировать всю программу. Достаточно исправить и протранслировать только один модуль и получить программу для загрузки, komponуя только что исправленный модуль с ранее полученными объектными модулями. В процесс компоновки вместе с программами, написанными на языках высокого уровня, включаются также программы ввода-вывода и стандартные подпрограммы в виде ранее оттранслированных объектных модулей. Существуют также несколько отладочных подпрограмм, которые по желанию могут подсоединяться к программе пользователя на этапе компоновки. Это, например, такие программы, как Snapshot (выдача промежуточных данных), Trace (прокрутка) и Trial Data Fa-pout (служит для образования дополнительных отладочных файлов на ленте или дисках).

10.7. Заключение

Из описания системы уровня J читателю должно быть ясно, что по концепциям она во многом отличается от систем для машин серии 1900. То же самое можно было бы сказать, если бы мы выбрали для рассмотрения другую систему. Читателю оставляем возможность произвести свои собственные сравнения. Современная тенденция при создании операционных систем такова, что все достоинства и некоторые недостатки предыдущих систем переносятся в следующие, и в результате операционные системы общего назначения приобретают большие размеры и занимают все больше процессорного времени и памяти. Первоначальный проект OS360 включал концепции почти всех известных операционных систем того времени. В то же время системы общего назначения стали более громоздкими и сложными в использовании, а в некоторых случаях колода карт языка управления работами достигала толщины в один или полтора дюйма. Необходимо вернуть утерянную простоту, но сохранить возможности системы. При этом система должна стать легкой в обращении, раскрывая свою сложность лишь тому, кто захочет использовать ее максимальные возможности.

СПИСОК ЛИТЕРАТУРЫ

1. Operating Systems George 1 and 2, Mark 6, International Computers Ltd., First edition, 1968.
2. Operating Systems George 3 and 4, International Computers Ltd., Second edition, 1968.
3. Disc Operating System J Level, Vols 1 and 2, International Computers Ltd.
4. OS/360 Concepts and Facilities, International Business Machines Ltd.
5. UNIVAC 494 Real-Time System Central Processor General Reference Manual, UNIVAC Division of Sperry-Rand Corporation.
6. UNIVAC 494 STARS Functional Description Manual, UNIVAC Division of Sperry-Rand Corporation.
7. ICL 1900 Series: Jean Manual, International Computer Ltd.
8. RUSH Terminal User's Manual, 1966, Allen-Babcock Computing Inc.
9. Bryan G. E., JOSS: Introduction to the System Implementation, p-3486, Rand Corporation, 1966.
10. Bryan G. E., Smith J. W., JOSS Language: Memorandum RM-5377-PR, Rand Corporation 1967.
11. Evans T. G., Darley D. L., On-Line Debugging Techniques: a survey, Proceedings-Fall Joint Computer Conference, 1966.
12. Baecker H. D., The Impact of Multiaccess, *Computer Bulletin*, March 1968.
13. Shaw J. C. JOSS, a designer's view of an experimental on-line computing system, Proceedings-Fall Joint Computer Conference, 1964.
14. Oestreicher M. D., Bailey M. J., Strauss J. L., GEORGE 3. A General Purpose Time-Sharing and Operating System, *Comm. ACM.*, Nov. 1967.
15. Proceedings of the A. C. M. Symposium on Operating System Principles, Gatlinburg, Tennessee, Oct. 1967, *Comm. ACM.* 77, № 5, May 1968, and including:
 Oppenheimer G., Weizer N., Resource Management for a Medium Scale Time-Sharing Operating System.
 Denning P. J., The Working Set Model for Program Behaviour.
 Dijkstra E. W., The Structure of the 'THE' Multiprogramming System.
 Lampson B. W., A Scheduling Philosophy for Multiprocessing Systems (resumé).
 Huxtable D. H. R., Warwick M. T., Dynamic Supervisors — their design and construction (resumé).
 Ackermann W. B., Plummer W. W., An implementation of a multiprocessing computer system (resumé).
16. Corbato F. J. et al., The Compatible Time-Sharing System: a Programmer's Guide, M. I. T. Press, 2nd edition 1965.
17. Martin J., Programming Real-Time Systems, Prentice Hall, 1965.
18. Daley R. C., Neumann P. G., A General-Purpose File System for secondary storage, Proceedings AFIPS 1965; Fall Joint Computer Conference, vol. 27, Pt. I, p. 213—229.
19. Wilkes M. V., The design of multiple-access computer systems: Part I, *Computer Journal*, May 1967.

20. Wilkes M. V., Needham R. M., The design of multiple-access computer systems: Part 2, *Computer Journal*, Feb. 1968.
21. Datafair 1966 Report. Articles on real-time operation, esp. Gill S., Why real-time?, *Computer Bulletin*, June 1966.
22. Martin J., Design of Real-Time Computer Systems, Prentice Hall, 1967.
23. Programming Systems and Languages, ed. Rosen S., McGraw-Hill, 1967.
24. Introduction to System Programming, ed. Wegner P., Academic Press, 1965.
25. Wilkes M. V., Time-sharing Computer Systems, Macdonald, 1968. (Русский перевод: Уилкс М. В., Системы с разделением времени, изд-во «Мир», М., 1972.)
- 26*). Системы с разделением времени, под редакцией Карплюса У., изд-во «Мир», М., 1969.
- 27*). Мили Г. и др., Функциональная структура OS/360, изд-во «Советское радио», М., 1971.
- 28*). OS/360: Супервизор и управление данными, изд-во «Советское радио», М., 1972.
- 29*). Джермейн К., Программирование на IBM/360, изд-во «Мир», М., 1971.

*) Звездочкой отмечены издания, добавленные при переводе. — *Прим. ред.*

СЛОВАРЬ

В словарь включены английские слова, встречающиеся в тексте книги. Как правило, указано то значение слова, которое используется в данной книге (и, возможно, не совпадает с наиболее распространенным).

ACCOUNT	— расчет	ENDMAC	— end of macro — конец макроопре- деления
ALGOL	— название языка	ENGAGE	— занято
ALLOT	— выделить	ENTER	— вход
AREA	— область	ERROR	— ошибка
ASSIGN	— отвести, выделить		
AT	— при		
BALANCE	— баланс	FAIL	— неудача
BEA	— название авиа- компаний	FORM	— format — формат
		FORTRAN	— название языка
		FUNCTION	— функция
CARD	— карта		
CLOSE	— закрыть		
COBOL	— название языка	GET	— получать
COMPILATION		GO	— перейти
FAILURE	— ошибка компиля- ции		
CORE	— оперативная па- мять или память на торах	HALTED	— остановленный
		HAVE	— иметь
CREDIT	— кредит		
CR0	— нулевое устрой- ство чтения карт	ICL	— International Computers Ltd — название фирмы
DATA	— данные	IF	— если
DEBIT	— дебет	IN	— в
DELETED	— уничтоженный	INPUT	— ввод
DEMAND	— требовать, запра- шивать	INTERRUPTED	— прерванный
DIRECTORY	— каталог		
DISCONNECT	— отсоединиться		
DISPLAY	— дисплей, индика- ция	JEAN	— название языка
DIVISOR	— делитель	JOB	— работа
DONE	— сделано	JOSS	— название языка
END	— конец	LINK	— связать
ENDJOB	— конец работы	LIST	— список
		LOAD	— загрузить

LOGIN	— зарегистрироваться, начать сеанс, войти в систему	READER	— считыватель
LOGOUT	— окончить сеанс, выйти из системы	REAL	— действительный
LP0	— нулевое печатающее устройство	RESTART	— возобновить, начать сначала
MACDEF	— macro-definition — макроопределение	RESUME	— продолжить
MISSING	— отсутствие, пропуск	RETURN	— возврат
NEED	— нуждаться	ROOT	— корень
OFFLINE	— автономно	RUN	— прогон, счет
ONLINE	— в системе	SALARY	— жалование
OPEN	— открыт	SAVE	— сохранить
OTHER	— другой	SEGMENT	
OUTPUT	— вывод	MISSING	— отсутствие сегмента
PARENTHESIS	— скобки	SOURCE	— исходный
PART	— часть	SQRT	— square root — квадратный корень
PASSWORD	— пароль	STEP	— шаг
PERSONNEL	— личный	STREAM	— поток
PL/I	— название языка	TAPE	— лента
PRINT	— печать	TO	— к
PROGRAM	— программа	TYPE	— печатать
PUT	— положить	USER	— пользователь
		WAGE	— заработная плата
		ZERO	— ноль

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

Автономный ввод-вывод 32
Архив файлов 72, 97
Ассемблер 149

База, предел 41, 43
Блок действия 89
Бюджетная система 74

Восстановление после сбоев 17, 135
Временной квант 39, 82—84

Генерация системы 144
Глава 85

Дампинг 57, 104
Действие 88
— начальное 93
Диалоговый фортран 115
Директор вычислительного центра 16

Журнал работы системы 52

Загрузка программ 40
Загрузчик 57, 150
Защита памяти 42—44
— программ 42—44

Идеальное время ожидания 82
Имя файла абсолютное 102
— — локальное 102
Индексно-последовательный доступ 146
Индекс программы для планирования 82
Интерфейс 14
Исходный модуль 149

Каталог пользователей 102
— собственный 102

Квант времени 39, 82—84
Команда ASSIGN 73
— DIRECTORY 100
— INPUT 73
— LINK 102
— OFFLINE 73
— ONLINE 78
— RESUME 79
— SAVE 79
Компоновщик 149
Координатор 91

Макро 62
Макропроцедура 63
Микропрограмма 28
Модуль исходный 149
— объектный 149
Мультидоступ 13, 15, 34, 75
Мультипрограммирование 13, 30

Область событий с периферийными устройствами 93
Объектный модуль 149
Оператор за пультом 18
Операционная система уровня J 137
Отказы и их устранение 134
Открытие файла 46

Пакетная обработка 14
Параллельная обработка 28
Перемещаемость программ 41
Планировщик 81, 147
— верхнего уровня (стратегический планировщик) 81
— нижнего уровня (тактический планировщик) 81
Подзадачи и подпроцессы 33
Подпрограмма 26
Пользователь 10
Последовательный доступ 105
Поток 147
Прерывания от периферийных устройств 35
Приоритет 37—39, 147

- Программа восстановления архива 105
 — пошагового дампинга 104
 — расчета с пользователем 74
 — управления связью (ПУС) 128
 — — случайным доступом (ПУСД) 130
 Программист 20
 Программное обеспечение 25
 Прямой доступ 146
 —
 Разговорные компиляторы 80, 110
 Разделение времени 13
 Ранг 147
 Распределение внешних устройств 44
 — памяти 40, 51
 Распределитель ресурсов 147
 Реальное время 119
 Редактор текста 149
 Реентерабельная программа 131
 Резидентная управляющая программа 13
 —
 Связь с оператором 44, 48
 Сегмент программы 149
 Секретность 103
 Система коммерческая 120
 — мультидоступа 121
 — отладки 149
 — пакетной обработки 14
 — реального времени 15
 — с разговорными языками 109
 — RUSH 114
 — UNIVAC STARS 119
 Сменщик глав 86
 Список действий 91
 Стандарты операционных систем 55—58
 Стоимость операционных систем 21
 Страничное сегментирование 42
 Структура архива 100
 Супервизор — главы 2, 3
 —
 Таймер 132
 Терминальный узел 100
 Типы файлов 99
 —
 Узел каталога 100
 — терминальный 100
 Управление данными 144, 145
 — запоминающими устройствами с произвольным доступом 129
 — коммуникационной аппаратурой 128
 — процессами 120
 Учет 73, 103
 Файл динамический 146
 — последовательный 99
 — постоянный 146
 — связи 99
 — с прямым доступом 99
 — on-line 97
 Физический доступ 145
 Функции супервизора 50
 —
 Цепочки 87
 — действий 89
 — команд 139
 —
 Экстракоды 21
 —
 Ядро 139
 Язык описания работ 55, 59
 — управления работами 55, 59
 — PL/1 107, 114
 —
 ALGOL 106
 —
 COBOL 106
 CONTORTS 124
 —
 EDSAC 26
 —
 FORTRAN 106
 —
 GEORGE 1 9, 54, 60
 GEORGE 2 9, 63, 69
 GEORGE 3 9, 71
 GEORGE 4 118
 —
 IBM/360 137
 —
 JEAN 80, 110, 111
 JOSS 80, 110
 —
 Off-lining 32, 67
 On-line файлы 97
 —
 RAND Corporation 80, 110
 —
 System 4 137
 —
 WATFOR 107

СОДЕРЖАНИЕ

Предисловие редактора перевода	5
Предисловие	9
<i>Мартин Уорвик.</i>	
1. Введение в операционные системы. Основные понятия	11
<i>Брайен Миллис.</i>	
2. Роль супервизорных программ	25
<i>Брайен Миллис.</i>	
3. Свойства супервизора	37
<i>Питер Беркиншоу.</i>	
4. Операционные системы. Введение в GEORGE 1 и GEORGE 2	54
<i>Тим Голдингем.</i>	
5. Концепции GEORGE 3	71
<i>Тим Голдингем.</i>	
6. Внутреннее функционирование системы GEORGE 3	84
<i>Ангус Битти.</i>	
7. Архив	97
<i>Дэвид Фостер.</i>	
8. Языки для работы в непосредственном контакте с машиной. Режим диалога	106
<i>Стюарт Дж. Миллер.</i>	
9. Реальное время. Специальные требования к управляющим программам	119
<i>Мартин Уорвик.</i>	
10. Операционная система уровня J	137
Список литературы	151
Словарь	153
Предметный указатель	155

УВАЖАЕМЫЙ ЧИТАТЕЛЬ!

Ваши замечания о содержании книги, ее оформлении, качестве перевода и другие просим присылать по адресу: 129820, Москва, И-110, ГСП, 1-й Рижский пер., д. 2, издательство «Мир».

СУПЕРВИЗОРЫ И ОПЕРАЦИОННЫЕ СИСТЕМЫ

Редактор *Л. Н. Бабынина*

Художник *А. Д. Смеляков*

Художественный редактор *В. И. Шаповалов*

Технический редактор *З. И. Резник*

Сдано в набор 17/IV 1972 г.

Подписано к печати 15/XI 1972 г.

Бумага № 3 60×90¹/₁₆ = 5 бум. л. 10 усл. печ. л.

Уч.-изд. л. 9,11. Изд. № 1/6505

Цена 64 к. Зак. № 145

ИЗДАТЕЛЬСТВО «МИР»

Москва, 1-й Рижский пер., 2

Ордена Трудового Красного Знамени

Ленинградская типография № 2

имени Евгении Соколовой Главполиграфпрома

Государственного комитета Совета

Министров СССР по делам издательств,

полиграфии и книжной торговли

Измайловский пр., 29