В.П. Симоненко

Организация вычислительных процессов в ЭВМ, комплексах, сетях и системах

1997

ГЛАВА 1(3)

ОСОБЕННОСТИ ОРГАНИЗАЦИИ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

ПРЕДИСЛОВИЕ

Параллельная ВС— это набор процессоров, способных совместно решать некоторую вычислительную задачу [82]. Данное определение достаточно широко и включает в себя; параллельные суперЭВМ, которые содержат сотни и тысячи процессоров, сети рабочих станций, мультипроцессорные рабочие станции и устроенные системы. Параллельные ЭВМ интересны тем, что в них заложены огромные возможности по концентрации вычислительных ресурсов, таких как процессоры, память, либо пропускная способность устройств ввода/вывода для решения важнейших вычислительных задач.

Параллелизм иногда рассматривается как редкая и экзотическая область вычислительной техники, интересная, но **не** имеющая практического применения и не используемая обычными программистами. Изучение тенденций развития программных приложений, архитектуры ЭВМ и вычислительных сетей показывает, что параллелизм в настоящее время является вездесущим, а параллельное программирование становится основным для разработчиков программного обеспечения для современных ВС.

По Мере того как компьютеры становятся все более быстрыми, можно справедливо предположить, что в конце КОНЦОР они станут "достаточно быстрыми" для практических приложений и необходимость в увеличении мощности компьютеров уйдет в прошлое. Однако, история показывает, что определенная технология, удовлетворяющая некоторое существующее программное обеспечение, способствует появлению новых программных продуктов, которые требуют развития новых технологий. В качестве интересной иллюстрации этого феномена можно привести отчет, составленный для правительства Великобритании в 1940 году, из которого следует, что по крайней мере две или возможно даже **три** ЭВМ способны удовлетворить возникшие потребности, связанные с вычислениями. В то время компьютеры использовались преимущественно для расчетов баллистических таблиц, так **что** авторы этого отчета не учитывали других научных, инженерных и коммерческих программных приложений.

Традиционно, развитие компьютеров высокого класса мотивировалось необходимостью численного моделирования сложных систем, таких как:

погода, климат, механические устройства, электрические цепи, производственные (4) процессы и химические реакции. Однако, в настоящее время наиболее значительными факторами, управляющими развитием высокопроизводительных ЭВМ, являются коммерческие приложения, которые требуют от компьютера способности оперировать большими объемами данных. Эти приложения включают в себя видеоконференции, компьютерную медицинскую диагностику, параллельные базы данных для поддержки решений, а также усовершенствованную графику, виртуальную реальность и многое другое. Интеграция параллельных вычислений и технологий мультимедиа ведет высокопроизводительные компьютерные сети к развитию *видео услуг,* т. е. проектируются компьютеры, обслуживающие сотни и тысячи одновременных запросов для формирования видео изображений в режиме реального времени. Причем, каждый видеопоток может содержать в себе как передачу данных со скоростью нескольких мегабайт в секунду, так и огромное количество операций по кодированию и декодированию дачных.

Несмотря на то, что коммерческие приложения могут определять архитектуру большинства будущих параллельных компьютеров, тем не менее традиционные научные программные приложения останутся важными для пользователей параллельной вычислительной технологии. Дело в том, что результаты, полученные чисто теоретическими методами, в отдельных приложениях требуют подтверждения практикой. Вместе с тем, проведение натурных экспериментов становится все более дорогим и непрактичным, поэтому компьютерное изучение сложных систем становится все более важным. Возрастающая требовательность исследователей к точности проводимых вычислений увеличивает вычислительные затраты таких расчетов и определяют необходимость повышения мощности и/или "разрешающей способности" вычислительной системы. Однако увеличивающиеся возможности ВС позволяют пользователям использовать их для более глубоких исследований, требующих более сложных расчетов. Этот подход имеет практически неудовлетворяемые требования к большой вычислительной мощности. Кроме того, ВС характеризуются требованиями к большим объемам памяти и повышенными требованиями к устройствам ввода/вывода и отображения информации. К примеру, моделирование десятилетнего изменения климата Земли, использующее современные модели, может включать в себя до 10^{16} операций с плавающей точкой — на это требуется десять дней при скорости выполнения 10^{10} операций с плавающей точкой в секунду. Кроме того, решение задач такой сложности может легко генерировать

сотни и даже большее гигабайт данных.

В общем случае, необходимость в высокоскоростных компьютерах определяется как большими объемами данных в коммерческих приложениях, так и огромным количеством вычислений в научных и инженерных приложениях. И по мере того, как инженерные приложения используют (5) все возрастающее количество данных, а в коммерческих приложениях появляются сложные расчеты, происходит все большее объединение запросов этих двух областей прикладного программирования.

С 1945 года и до настоящего времени производительность высокоскоростных компьютеров возрастала по экспоненциальному закону с коэффициентом роста равным 10 каждые 10 лет. В то время, как первые ЭВМ выполняли до нескольких десятков операций с плавающей точкой в секунду, производительность параллельных компьютеров середины 90-х годов достигает десятков миллиардов операций в секунду (рис. 1.1).

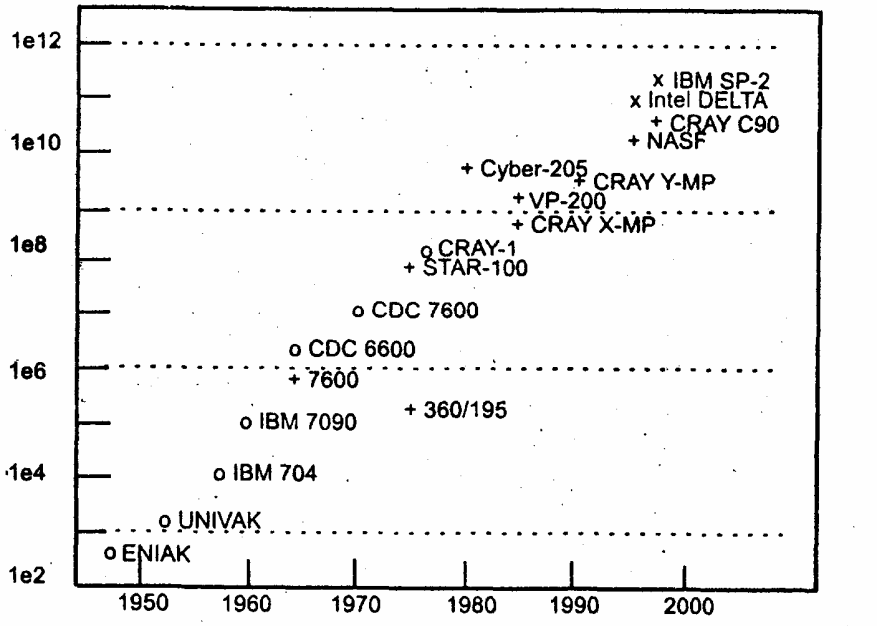


Рис. 1.1 (по вертикали указано количество операций с плавающей точкой в (секунду) *Высшие показатели производительности некоторых высокоскоростных суперкомпьютеров, 1945 — 1995. Экспоненциальный рост кривой несколько спадает в 1980-", однако, он снова возрастает по мере того, как появляются параллельные суперкомпьютеры. Символ "o" означает однопроцессорные системы, "+" — параллельные векторные компьютеры с 4-16 процессорами и символ "x"— означает мощны параллельные ЭВМ с сотнями или тысячами процессоров.*

(6)

Аналогичные тенденции наблюдаются и в компьютерах более низкого класса, принадлежащих к разным поколениям: калькуляторах, персональных компьютерах и рабочих станциях. И мало вероятно, что этот рост не будет продолжаться в дальнейшем. Тем не менее, архитектура компьютеров, поддерживающая данный рост, радикально изменяется от последовательной к параллельной.

Производительность компьютеров зависит от времени, необходимого для выполнения основных операций или **"тактового цикла",** процессора, т.е. времени, необходимого для выполнения элементарных операций. Однако, времена тактовых циклов уменьшаются медленно и ограничены величинами физического характера, такими как скорость света (рис 1.2). Можно с уверенностью сказать, что в будущем увеличение производительности вычислений в ВС мало будет зависеть от скорости самого процессора.

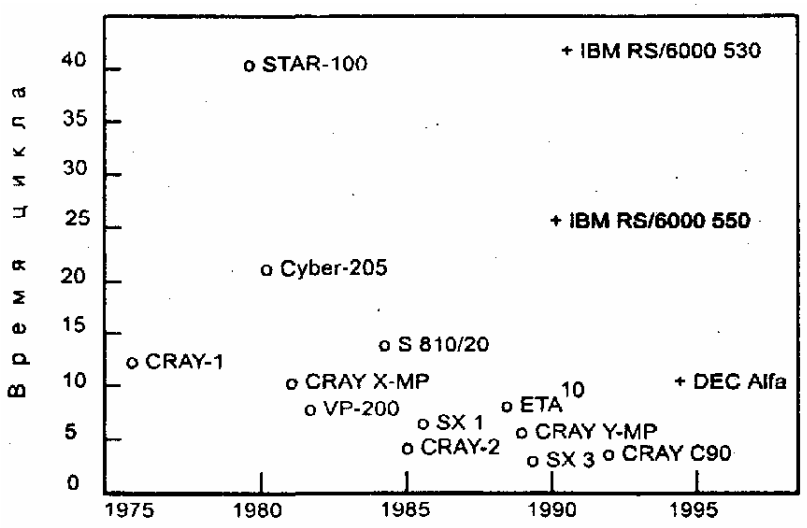


Рис. 1.2 (время цикла дано в наносекундах)

Тактовые циклы векторных суперкомпьютеров (обозначены символом "о") уменьшились за шестнадцать лет в три раза, начиная от CRAY-! (12.5 наносекунд) и заканчивая С90 (4.0). RISC микропроцессоры (обозначены символом "+") быстрее приближались к подобной производительности. Обе архитектуры похоже уже близки к физическим пределам. (7)

Для того, чтобы обойти эти ограничения используется внутренний параллелизм в самом чипе (кристалле). Применяются и другие различные подходы, включая использование конвейерного режима (различные стадии нескольких команд выполняются одновременно) и многофункциональных устройств (несколько умножителей, сумматоров, контролируемых одним потоком команд). Все больше и больше внедряются "мультикомпьютеры", каждый из которых имеет свой процессор, память и ассоциативную взаимосвязанную логику. Этот подход перспективен благодаря развитию VLSI технологии и позволяет уменьшать количество технологических компонентов, встраиваемых в компьютер. Т.к. стоимость компьютера (очень приблизительно) пропорциональна количеству содержащихся в нем компонент, то увеличение уровня их интеграции ведет к увеличению количества процессоров, входящих в компьютер, при той же относительной цене.

1.1. МОДЕЛИ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Общепринято рассматривать раздельно различные классы моделей для последовательных и параллельных ВС: машинные, архитектурные» вычислительные и программные (12,16,30,52,144). Эти классы различаются не только по назначению, но и также по уровню абстракции.

Для организации и планирования параллельных вычислений в многопроцессорных вычислительных системах, комплексах и сетях целесообразно рассматривать два типа моделей:

- машинная модель, которая определяется архитектурой компонентов ВС;
- программная модель вычислений, включающая различные компоненты для выполнения и отображения алгоритма.

В данной работе рассматриваются модели ВС, которые уже внедрены или, возможно, будут внедрены в ближайшем будущем. Все обобщения, сведенные в модели параллельных вычислительных систем, базируются на концепциях универсальной алгебры представленной в понятиях абстрактной алгебры [90]:

Абстрактная алгебра использует двойку (E,F), где E — не пустое множество элементов, а F— множество (внутреннее или внешнее) операций, определенных на E.

Собственно говоря, ВС выполняет то, что определено в абстрактной алгебре; использование или обращение (автоматическое или нет) исполняемых операций (команд) к данным- Единственное различие лежит в понятии исполнительной среды- в случае абстрактной алгебры обращение к операциям, описанным множеством F, и их выполнение (8)осуществляет пользователь; для компьютера исполнительной средой является устройство управления, а его поведение определяется программой.

Оговоренные понятия приводят к нижеследующему обобщенному описанию модели компьютера:

компьютер = данные + операции (1)

Это выражение является исходным для последующих заключений.

- *Общие характеристики моделей вычислительных систем*

Анализ уравнения (1) приводит к выводу, что любая модель ВС должна иметь такие характеристики, которые бы допускали возможность применения концепции универсальной алгебры, т. е. способ представления данных (и доступа к ним), и выполнение операций с этими данными.

Кроме того, любая модель должна определять:

- существование двух синергических компонент:
- памяти, как физической поддержки представления данных.
- аппаратной части для управления выполнением операций (используя достоверные данные).
- возможность произвольной, прямой (параллельной или последовательной) выборки данных.
- число возможных параллельно выполняемых операций или одновременно выполняемых операций с памятью;
- обмен данными между синергичсскими компонентами (памятью и аппаратной частью) осуществляется как выполнение команд в логическом устройстве.

Параллельность доступа к памяти или выполнения команд - это критерии, которые переводят компьютеры в разряд параллельных или последовательных машин.

Модель ВС не определяет конкретный способ ее физической реализации. Например, реальный доступ к памяти ВС может быть реализован следующими способами:

- случайный или прямой, когда исполнительная часть имеет непосредственную физическую связь с памятью (bus), что дает возможность обращаться к любой ячейке памяти;
- через сетевую взаимосвязь (network), которая является обобщением обшей шины памяти.(9)

Требования к выполнению программы также заключаются в том, что программа и данные должны находиться в памяти, какая бы ни была физическая реализация доступа к ней (реализации фон Неймана или Гарварда). Например, системы 180x86 имеют память с сегментной организацией, в которой одновременно хранятся данные и программы, а в вычислительных машинах с массовым параллелизмом, таких как Connection Mashine [9б], Sphinx [130|, или MasPar MP-1 [122], данные передаются параллельно и располагаются в рабочих ОЗУ исполнительных устройств, а команды — в памяти центральной (host) вычислительной машины.

1.2. МОДЕЛЬ С ПОСЛЕДОВАТЕЛЬНЫМ ДОСТУПОМ К ПАМЯТИ (RAM МОДЕЛЬ)

Некоторые модели ВС [97] определены для последовательных вычислительных машин. Модель Тьюринга (Turing machine model) или конечный автомат (который по сути является частным случаем машины Тьюринга) отличается ограниченностью характеристик случайного доступа к памяти.

Память с RAM доступом [67] хорошо подходит для проектирования и организации последовательных ВС и хорошо исследована. Эта модель имеет исполнительную (.рабочую) часть, которая взаимодействует с основной памятью. ВС с RAM имеют SISD архитектуру (рис. 1.4.).

Рис. 1.4. Структура ВС SISD организации с RAM памятью

Модель с RAM организацией памяти позволяет выполнять только одну команду в каждый момент времени, независимо от того, какой окажется данная команда (ссылкой к памяти или арифметической операцией).

Операции, выполняемые на одном исполнительном устройстве, следуют в жесткой последовательности, не допускающей параллелизма, даже если архитектурные усовершенствования допускают конвейерное выполнение некоторых из операций (не все операции требуют для своего выполнения всех внутренних устройств) или их параллельно-подобное выполнение (например, в случае с DMA. Direct Memory Access - прямой доступ к памяти — способ обмена данными между адаптером и памятью без участия процессора,

что может заметно снизить нагрузку на (10) процессор и повысить общую производительность системы). Для повышения эффективности работы ЭВМ также используется организация просмотра команд вперед или наличие нескольких исполнительных блоков, а также использование многоуровневой кэш памяти или внутреннее процессорное буферирование запросов записи.

Программа и данные хранятся в памяти с прямым или случайным доступом. Практически, каждый одноадресный компьютер с организацией фон Неймана – это RAM машина. Анализ сложности алгоритмов для RAM машин сводится к анализу времени выполнения отдельных команд или программ, работающих в многопрограммном режиме, при реализации истинного или кажущегося совмещения.

1.3. МОДЕЛИ ПАРАЛЛЕЛЬНЫХ МАШИН

Параллельные компьютеры, так же как и последовательные состоят **из** двух функциональных компонент: *исполнительной (процессора)* и *памяти.* Различия между ними состоит только в количестве используемых компонент (рис. 1.5).

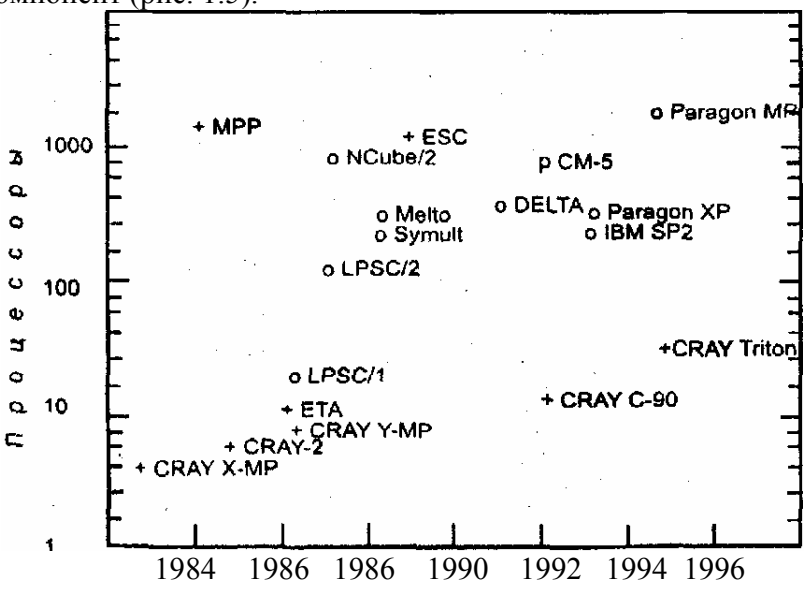


Рис. 1.5. Число процессоров и мощных параллельных компьютерах ("o") 11 и векторных мультипроцессорах ("+")

(11)

Особенности архитектуры параллельных ВС (рис. 1.6) и организации параллельных вычислений позволяют разделить их на несколько классов: SIMD, M1SD, M1MD (классификация флина (Flynn's classification) 1966) (12,97. I30].

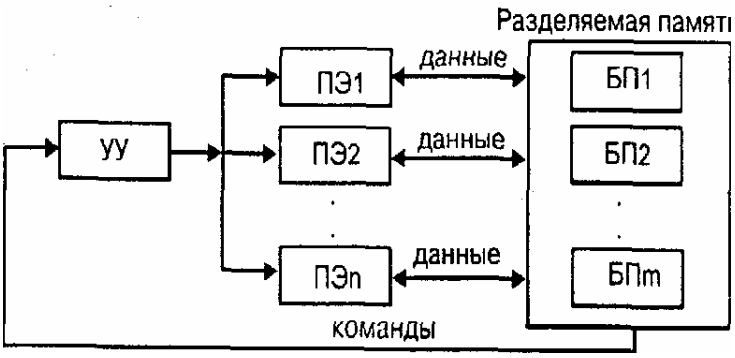


Рис. 1.6.а Структура ВС - SIMD

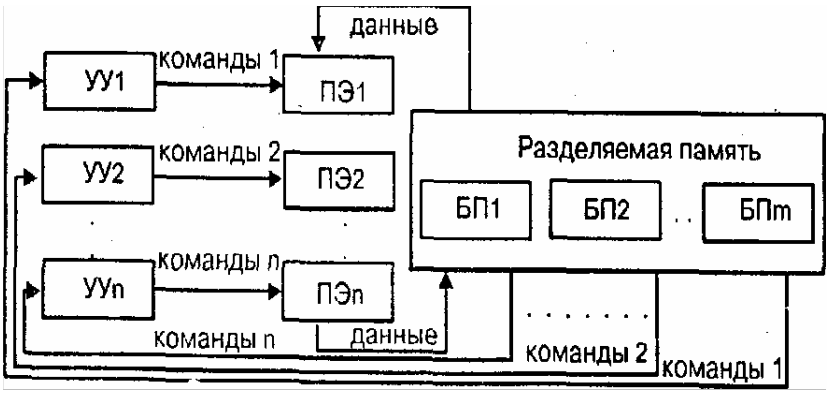


Рис. 1.6.б Структура ВС - M1SD

(12)

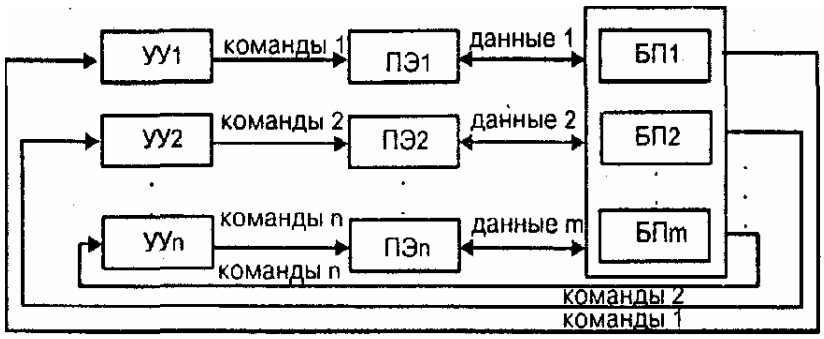


Рис. 1.6.в Структура ВС – MIMD

В табл. 1.1 приведены некоторые наиболее известные коммерческие параллельные и последовательные ВС.

Табл. 1.1

SISD с одним функциональным блоком	IBM 701, IBM 1620, IBM 7090. PDP VAX il/780
SISD с множеством функциональных блоков (п)	IBM 360/91, IBM 370/168UP, CDC 6600, CDC Star-100. Tl-ASC. FPS AP-120B, FPS AP-164, IBM 3838, Cray-1. CDC Cyber-205, Fujitsu VP-200. CDC-NASF, Fujitsu FACOM-230/75
SIMD	illiac-IV. PEPE. BSP. STARAN, MPP. DAP, MasPar, MP
MIMD	IBM 370/168 MP, Univac 1100/80, Tandem/16, IBM 3081/3084, Cm*, Burroughs D-825, C-mmm, Cray-2, S-1, Cray-x MP, Denelcor HEP, IBM SP , Imcl Paragon, CM5, Cray T3D, Meiko CS - 2, nCUBE Silicon Graphics Challenge. Sequent Symmetry.

Тзе-Йан Фенг в 1972 г. (Tse-Yun Feng) предложил использовать степень параллелизма для классификации {различных архитектур ВС, а X^ндлер (Haendler) в 1977 г. предложил новую классификацию ВС, учитывающую степень параллелизма и степень конвейеризации вычислений.(13)

В параллельных ВС исполнительная среда позволяет осуществлять параллельное выполнение нескольких команд, однако, при этом может потребоваться одновременный параллельный доступ к памяти. Следовательно, необходим более сложный механизм взаимодействий исполнительных вычислительных узлов с модулем или модулями памяти для выполнения операций над данными или получения инструкций. По этому признаку архитектуры параллельных ВС можно разделить на два класса:

- ВС, допускающие одновременный и случайный доступ к обшей (глобальной и локальной) памяти. Некоторые из них относятся к PRAM ВС (машины с параллельным случайным доступом). С точки зрения архитектуры - это компьютеры с разделяемой памятью.
- ВС, в которых каждый процессор имеет прямой или случайный доступ только к своей локальной памяти и, кроме того, может косвенно- через сеть (interconnection network) обратиться к памяти других процессоров.

Если сеть статична, мы имеем дело с компьютером FITM (машины с постоянной топологией), в противном случае

это FLTM (машины с динамической топологией или реконфигурируемые).

Концепция сети связи заимствована **из** PRAM модели, исходя из предположения о том, что доступ к памяти выполняется в один и тот же момент времени. В реальных машинах, базирующихся на концепции PRAM архитектуры, необходимы мощные средства связи с высокой пропускной способностью, что на практике не достижимо потому, что доступ к разделяемой памяти происходит через коммуникационную сеть и требует высоких аппаратных затрат. В настоящее время активно ведутся разработки высокопроизводительных коммуникационных систем передачи информации для параллельных вычислительных систем. За последние пятнадцать лет скорость передачи информации в распределенных системах обработки информации увеличилась на несколько порядков (рис. 1.7). Среди получивших наибольшее распространение сетевых технологий можно отметить следующие [89]:

- Ethernet — имя данное наиболее популярной технологии построения локальной сети, разработанной компанией Xerox PARC. Ethernet это шинная технология передний информации с распределенным управляемым доступом со скоростью передачи до 10 Мбит/с.
- FDDI — Fiber Distributed Data Interface. FDDI — это ВС с кольцевой архитектурой использующая оптоволоконный канал связи между станциями в двойное кольцо для обеспечения надежности.
- HiPP — High-performance parallel interface. HiPP — это стандарт передачи данных на 800 Мбит/с по 32 или на 1,6 Гбит/с по 64 параллельным медным линиям. Большинство распространенных коммерческих ВС используют HiPP интерфейс.(14)
- SONET — Synchronous Opptica] Network. SONET — это серия оптических сигналов на основе умножения базовой скорости 51.84 Мбит/с названной OC-1. OC-3 (155.52 Мбит/с) и OC-12 (622.08 Мбит/с) были разработаны для B-1SDN сетей. Кроме того, уже определен стандарт OC-192 (9.952 Гбит/с) для будущих сетей.
- ATM — Asynchronous Transfer Mode. ATM является технологией для передачи, мультиплексирования и переключения, что обеспечивает высокую степень гибкости, требуемой в B-ISDN. ATM — это протокол связи, который использует пакеты с фиксированным 48 байтовым размером имеющим 5-ти байтовые заголовки.

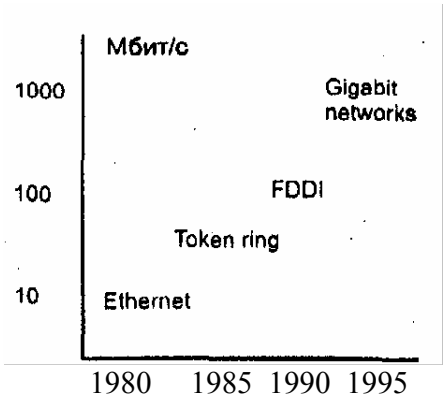


Рис. 1.7, Скорости передачи информации в сетях

Однако, для реализации PRAM архитектур требуется достаточно сложная технология организации вычислительного процесса для реализации концепции конкурирующею доступа, даже в том случае, когда для одновременной множественной операции чтения в разделяемой памяти используется секционирование памяти или ассоциативная память.

Обе эти модели (P1TM, FLTM) предполагают, что обращение к памяти осуществляется в одно и то же время, независимо от способа организации памяти.

В любом случае, приведенные ниже различные модели параллельных Мишин описываются одними и теми же характеристиками абстрактной мелели абстрактной алгебры.

Представим наиболее важные особенности этих классов параллельных машин.(15)

1.3.1. Модель с разделяемой памятью (^RAM модель)

На рис.1.8 приведена структура PRAM модели. NRAM процессоров P_i, $[0 \quad i \quad (N-1)]$ могут иметь случайный и одновременный доступ к обшей разделяемой памяти в одно и то же время.

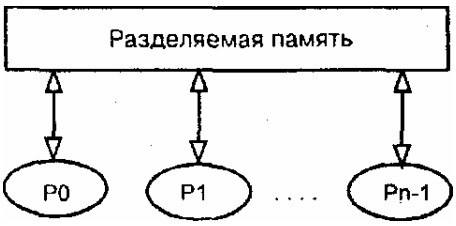


Рис. 1.8

Эта модель демонстрирует проблему доступа, когда несколько компьютеров одновременно обращаются к одной и той же ячейке памяти. Если одновременность выполнения операций чтения кажется концептуально корректной (даже если физически эта реализация не очевидна), то это не относится к совмещению операций записи. Поэтому оценка параллельного выполнения операций записи и/или чтения приводит к разделению ВС PRAM архитектуры на несколько подклассов организации и управления памятью: SRSW, SRMW. MRSW и MRMW.

- SRSW модель организации памяти предусматривает, что только один процессор имеет право-на операцию чтения или записи в одну ячейку памяти;
- SRMW модель допускает один процессор для чтения и множество процессоров для записи;
- MRSW модель допускает одновременное чтение множества данных, но только единственную операцию записи.
- MRMW модель позволяет обоим операциям чтения и записи выполняться одновременно несколькими процессорами.

SRSW и MRMW наиболее популярны при исследовании параллельных алгоритмов и организации процессов в параллельных системах, а PRAM SRSW — для технической реализации. SRSW модели простые и доступ к памяти в них может быть реализованы в виде RAM модели.

В случае машин, допускающих параллельную, но конкурентную операцию записи в память, понятие параллельности записываемых в область памяти данных не всегда точно определено. Несколько процессоров одновременно требуют выполнения операции записи множества данных в(16) память. Здесь можно определить некоторые наиболее распространенные реализации этого "параллелизма".

- детерминированный — это означает, что предпочтение на запись данных отдается процессору с наивысшим приоритетом;
- комбинированный — это означает, что данные некоторых процессоров объединены путем сокращения операций (обычно ассоциативных и коммутативных, таких как сложение, умножение, max, or,...) и уменьшено количество обращений к памяти за счет хранения промежуточных результатов в процессоре (только результат операции передается в память);
- недетерминированный — непредсказуемый, когда произвольное значение должно быть немедленно запомнено в памяти.

1.3.2. Вычислительные системы со статической (FITM) топологией

ВС с фиксированной топологией (FITM) предполагают наличие жесткой сети, которая связывает различные процессоры. Доступ процессоров к памяти других процессоров в таких машинах осуществляется через промежуточные процессоры с использованием сетевой связи, к собственной же памяти процессоры имеют прямой доступ.

Рис-1.9, в общих чертах представляет структуру архитектуры FITM ВС. Сеть связи соединяет пКАМ процессоров P_i, $[0 \quad i \quad (N-1)]$, каждый из которых имеет свою собственную память со случайным доступом M_i.

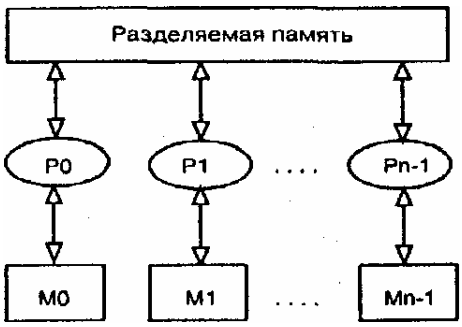


Рис 1.9

Некоторые коммерческие компьютеры (CM, Sphinx. MasPar's MP-1) поддерживают именно эту модель, которую значительно легче реализовать, чем общую PRAM модель.(17)

В FITM модели обмен данными между вычислительными узлами играет важную роль и в значительной степени определяет характеристики ВС. В этом случае описание компьютерной модели (1) модифицируется в форму (2):

компьютер = данные + операции + обмен данными (2)

Для соседних ячеек памяти статическая сетевая топология допускает непосредственный обмен. Для осуществления глобального обмена используют механизм, называемый трассировкой, допускающий обмен данными через связывающую сеть определенной топологии (матричная, гиперкуб, бинарное дерево, пирамидальная и т.д.).

В случае SIMD FITM (только коммерческих) возможна синхронизация общих связей (выполняемая параллельно с вычислениями, например Connection Masliine), В случае MIMD FITM возможен как синхронный, так и асинхронный обмен данными.

Синхронизация связи последовательных процессов поддерживается на основе принципа Гоара [98] (Hoare's principle), который требует предварительной синхронизации процессоров перед обменом данными. Этот механизм часто использовался в транспьютерных системах.

Асинхронный обмен данными возможен, но требует существования канала связи. Это механизм нашел применение в пирамидальных компьютерах SPY1NX [58| и в компьютерах MEDECINE.

Статические системы (FITM) часто используются в системах с распределенной памятью. Имеются следующие типы систем этого класса.

- Полносвязанная сеть (Completely-Connected Network), где каждый вычислительный узел имеет прямую связь сдругим (рис. 1.10).
- Звезда (Star-Connected Network), где имеется один цешральный процессор, который связан со всеми другими процессорами и через него осуществляется связь между ними (рис 1.11).
- Линейная и Кольцевая сети (Linear Array and Ring), где каждый вычислительный узел связан с двумя соседними узлами (рис 1.12.а. рис. 1.12.6).
- Петля (сеть) (Mesli Network). Эта сеть может быть одномерной (Кольцевая), даумерной — матричная (рис 1.13 а), ... и также с обратной связью—торроидальная (рис 1.13.6), Примерами двумерной системы являются DAP и Paragon XP\S, трехмерной системы (рис. !.13.в). являются Cray T3D, Тега computer, и J-Machinc.
- Дерево (Tree Network), где существует только одна связь между каждой парой вычислительных узлов. Примеры такой системы является DADO. Для увеличения способности передачи имеется, так называемое, жирное дерево (рис 1.14) в CM-5.(18)

Рис 1.12.а. Линейная

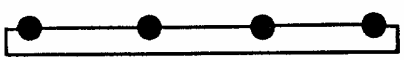


Рис. 1.12.6. Кольцевая

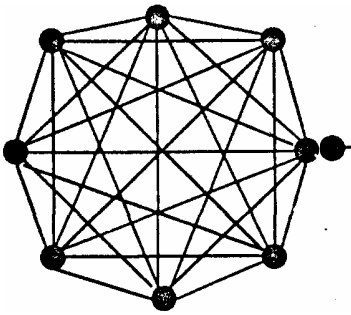


Рис. 1.10. Полносвязная

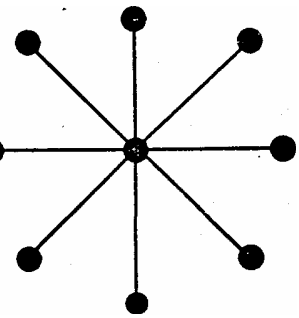


Рис. 1.11 Звезда

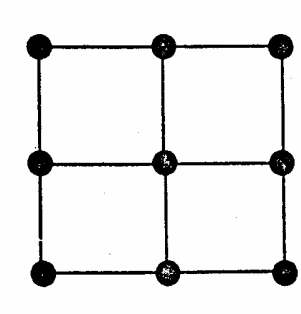


Рис 1.13.а Матричные

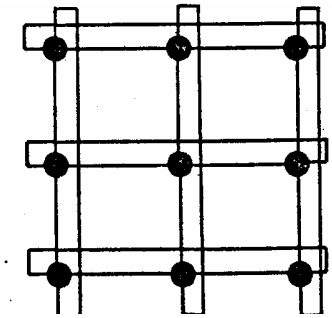


Рис 1.13.б Торроидальные

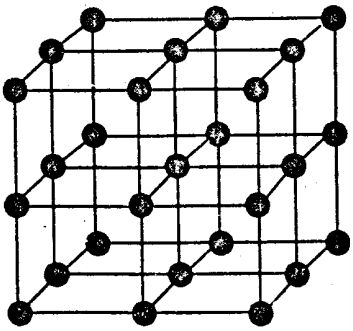


Рис 1.13.в Трёхмерная сеть

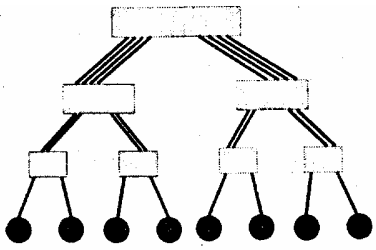


Рис. 1.14. Жирное дерево(19)

Гиперкуб (Hypercube Network). Это многомерная сеть процессоров с двумя процессорами в каждом измерении (рис. 1.15.а-г). D-мерный гиперкуб содержит p=2° процессоров. Примерами такой системы являются nCUBE 2, Cosmic Cube, и iPSC.

k-d Сеть (k-ary d-cube Networks). Эти топологии определяют пределы класса топологий, называемых k-ary d-cube. Где D размерность сети, а К — основание системы счисления сети, которое определяется как количество процессоров в каждом измерении. Число процессоров в сети равна $К^0$, d-мерный гиперкуб также называемый бинарным d-кубом — это d-мерная петля с 2-мя процессоров в каждом измерении.

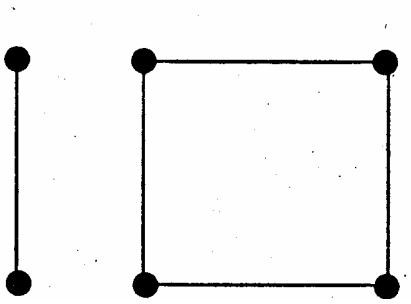


Рис. 1.15.а. 1-D

Рис. 1.15.6. 2-D гиперкуб

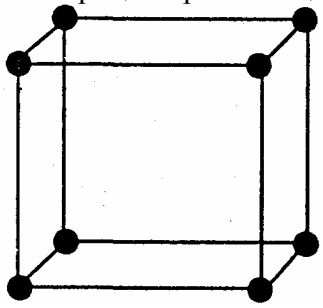


Рис. 1.15.В. 3-D гиперкуб

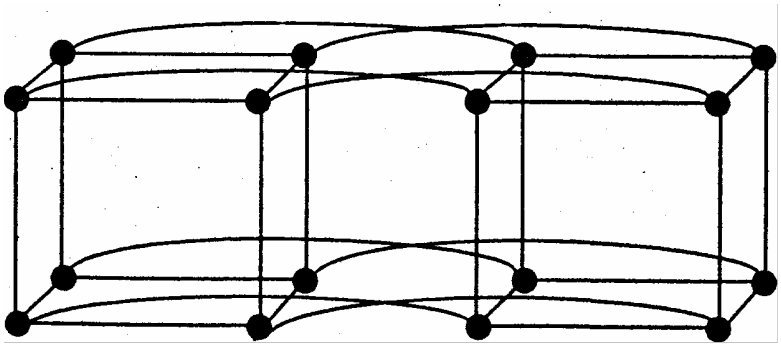


Рис. 1.15.г. 4-D гиперкуб(20)

Табл. 1.2 Сравнение характеристик статических сетей с p процессорами

Вычислительные системы	Диаметры	Бисекторная ширина	Связность по дугам	Число каналов
Полносвязная сеть	1	$P^2/4$	P-l	p(p-1)/2
Звезда	2		1	P-l
Линейная	P-l	1	1	P-l
Кольцевая	[P/2]	2	2	P
Матричная	$2[\sqrt{p}/2]$	$2\sqrt{p}$	4	2p
Древовидная	21og((p+l)/2)	1	1	P-l
Гиперкуб	logp	P/2	logp	(p)ogp)/2
k-d сеть	d[k/2]	$2k^{d-1}$	2d	dp

- Диаметр сети — это максимальное расстояние между любыми двумя процессорами в сети. Расстояние между двумя процессорами определяется как ближайший путь (в количестве соединений) между ними. Поскольку большее расстояние определяет время связи, сети с меньшим диаметром лучше.
- Связность сети — это мера сложности пути между любыми двумя процессорами. Желательно, чтобы сеть была с высокой связностью, т.к. это снижает число соединений для коммутации ресурсов. Одно из измерений связности — это число дуг, которое должно быть удалено из сети, чтобы превратить ее в две взаимонесвязанные сети. Это называется связность по дугам.
- Бисекторная ширина и бисекторная полоса пропускания. Бисекторная ширина сети определяется как минимальное число связей, которое должно быть удалено, чтобы разбить сеть на две равные половины. Число бит, которые могут передаваться одновременно через соединение двух процессоров, называется шириной канала. Ширина канала равна числу физических шин в каждом канале связи. Пиковая частота на которой одиночный физический носитель может выдавать (передавать без ошибки) биты называется частотой канала. Пиковая частота, на которой данные могут передаваться между концами канала связи, называется полосой пропускания канала. Полоса пропускания канала зависит от частоты канала и шириной канала.
- Себестоимость (число каналов). Много критериев может быть использовано для оценки себестоимости сети. Один из методов определения себестоимости сети — число каналов связи или число физических носителей в сети.(21)

1.3.3. Вычислительные системы с динамической (FLTM) топологией

Рассмотрим параллельный FLTM компьютер PRAM модели с p процессорами и общей памятью, состоящей из m слов. Процессоры соединяются с памятью через множество переключательных элементов. При этом обеспечивается доступность слова памяти каждому процессору. Любой из p процессоров может обратиться к любому слову памяти, обеспечивая при этом единичность своего доступа. Для обеспечения этого условия общее число переключательных элементов, в общем случае, должно быть пропорционально произведению txp. Отсюда видно, что даже при небольших размерах памяти построение переключательной сети такой сложности является очень дорогостоящим. Один из способов устранения данного недостатка состоит в разбиении всей памяти на отдельные участки, называемые банками. При этом процессоры системы будут переключатся между банками. Однако и этот способ не лишен недостатков. При обращении процессора к одному из банков все остальные процессоры обратиться туда уже не могут. В параллельных компьютерах применяются следующие способы организации взаимодействия процессоров и банков памяти:

- Перекрестные переключательные сети (Crossbar switching Networks);
- Шинно-ориентированные сети (Bus-Based Networks);
- Многоступенчатые взаимосвязанные сети (Multistage Interconnection Networks).

Перекрестные переключательные сети

Наиболее простой способ соединения p процессоров и Ь банков памяти получается при использовании перекрестных переключателей. Перекрестный переключатель (рис. 1.16) использует решетку переключательных элементов.

Перекрестные переключательные сети являются неблокирующими сетями в том смысле, что соединение одного процессора с банком памяти не препятствует другим процессорам соединяться с любыми другими банками. Обычно число банков b берется большим или равным числу процессоров, так что каждый процессор в худшем случае имеет один банк памяти для доступа. Общее число переключательных элементов, требуемых для построения такой сети, равно pxb. Это приемлемо, если предположить, что число банков по крайней мере равно p. Если в системе $p_x b$, то тогда в любой момент времени могут появиться процессоры не имеющие доступа к памяти. Отсюда следует, что с увеличением числа процессоров p сложность сети возрастает как p^2. Поэтому переключательные сети становятся труднореализуемыми при большом числе (22) процессоров, при этом значительную роль играет также и ценовой фактор, Перекрестные переключательные сети в настоящее время используются в таких системах как Cray-YMP и Fujitsu VPP 500. В VPP 500 используется перекрестная сеть размером 224x324.

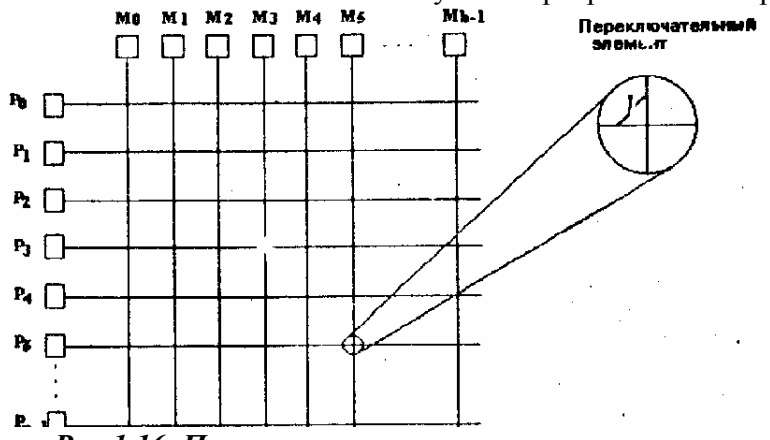


Рис.1.16. Перекрестная переключательная есть

Шинно-ориентированые ***сети***

В шинно-оринтированных сетях процессоры соединяются с обшей памятью посредством единого канала данных, называемого шина. Такая система очень проста для реализации. Рис. 1.17(а) иллюстрирует обычную шинную архитектуру. При обращении процессора к общей памяти на определенных линиях шины выставляются соответствующие сигналы.

Главным недостатком данного способа является то, что при увеличении числа процессоров, связанных с

общей шиной, весьма значительно увеличивается время ожидания каждым процессором доступа к памяти. Данный способ пригоден для небольшого числа процессоров.

Один из способов решения данной проблемы заключается в обеспечении каждого процессора локальной кэш-памятью, **как** показано на рис, 1.17(б).(23)

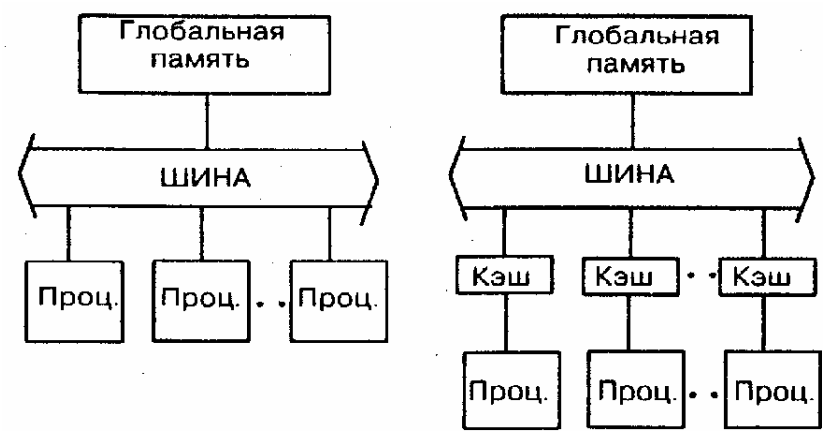


Рис.1.17.а Шинно-ориентированые сети

Рис. 1.17.6 Кэш шинно-ориснтированые сети

Использование памяти в шинно-ориентированных сетях с FLTM топологией отличается от F1TM ВС. При обычных вычислениях, когда ссылка делается на какой либо фрагмент памяти, следующая ссылка вероятней всего будет сделана на фрагмент" памяти, следующий за этим фрагментом. Этот вывод сделан на основе свойств "временной" и "пространственной" локальности теории рабочих множеств. Следовательно, весь этот фрагмент может быть помещен J локальную кэш-память процессорного элемента ВС и всю дальнейшую обработку данных он может вести не обращаясь к общей памяти. При этом существенно сокращается время работы процессора с общей памятью. В случае кэш-промаха (т.е. когда в локальной памяти не оказалось требуемого слова) процессору необходимо еше раз обратиться к общей памяти. Данный способ применяется в машинах Symmetric, Multimax и других.

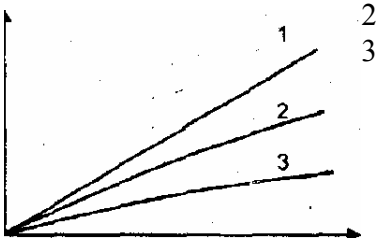
Многоступенчатые взаимосвязанные сети

Перекрестные переключательные сети выигрышны с **точки зрения** производительности, но не всегда приемлемы из-за высокой цены. Наоборот, шинно-ориентированные сети приемлемы по стоимости, но имеют ограничения по производительности. В качестве среднего класса сетей используются многоступенчатые взаимосвязанные сети. Они более производительные чем шинные и дешевле перекрестных. Рис,1.18(а) [97] иллюстрирует зависимость стоимости перекрестных, многоступенчатых,(24) шинно-ориентированных сетей в зависимости от числа процессоров. На рис. 1.18(6) |97] показана зависимость производительности данных сетей от числа процессоров.

1 - Перекрестные
2 - Многоступеньчатые
3 - Шинные

Рнс.1.18(а). Зависимость стоимости реализации от числа процессоров

Производительность



1 - Перекрестные
2 – Многоступемьчатые
3 - Шинные

Количество процессоров

Рис. 1.18(6). Зависимость производительности от числа процессоров.

Обычно многоступенчатая сеть включает p процессоров и b банков памяти, как показано на рис.1.19.

Многоступенчатая взаимосвязанная сеть обычно использует для связи ступеней ***delta*** сеть. В сети присутствует **log p** ступеней, где p число процессоров и банков памяти. ***Delta*** сеть включает взаимосвязанную модель, содержащую p входов и p выходов. Между входом j и выходом i есть связь при выполнении условия:

$$j \quad \begin{matrix} 2^i, при 0 \quad i \quad p/2 \quad 1 \\ \lfloor 2^{i+1}, при p/2 \quad i \quad p \quad 1 \end{matrix} \quad (25)$$

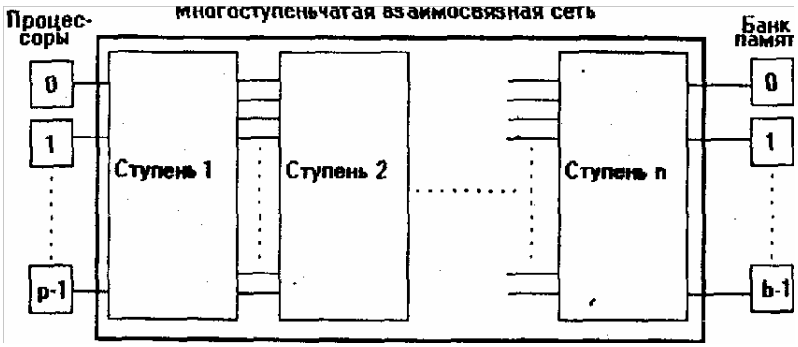


Рис.1.19. Многоступенчатая взаимосвязанная сеть

Эта модель связи называется "полная перестановка" (Perfect shufle). Рис.1.20 иллюстрирует полную модель для 8 входов и выходов. В каждой ступени ***delta*** сети полная перестановочная модель связи включает **p/2** переключательных элементов. Каждый переключатель может находиться в одном из двух состояний. В первом состоянии входы связываются с выходами напрямую, как показано на рис.1.21(а). Это называется прямой связью. В другом состоянии входы соединяются с выходами перекрестным способом, как показано на рис. 1.21(6). Это называется перекрестной связью. Возможны еще два варианта связей рис.1.21(в,г).

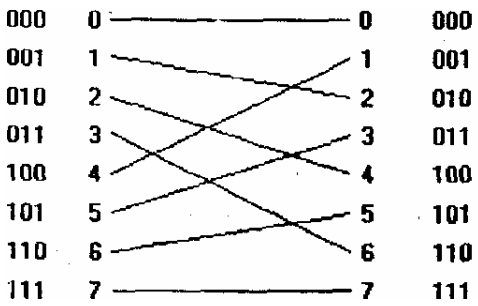
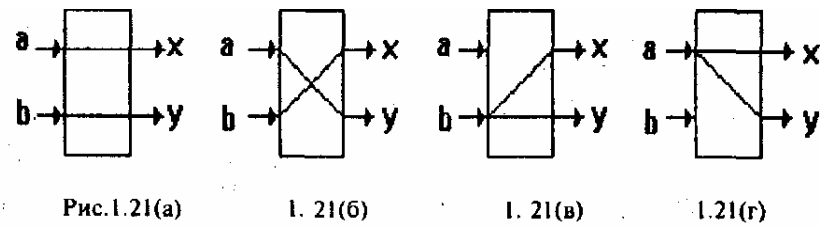


Рис. 1.20(26)



Вся *delta* сеть имеет p/2xlog(p) переключательных элементов и сложность такой сети растет как pxlog(p). Заметим, что эта сложность ниже чем p^2 сложности перекрестной сети. Рис. 1.22 показывает *delta* сеть для 8-ми процессоров. Процессоры расположены на входе сети, банки памяти - на выходе. Передача сообщений в *delta* сети выполняется по простой схеме, пусть s и t двоичное представление источника и приемника сообщения. Сообщение пересекает звено первого переключательного элемента. Если старшие

биты в s и t одинаковы, тогда сообщение передается по Прямой связи переключателя. Если эти биты различны, то тогда сообщение передается по перекрестной связи переключателя. Это правило повторяется в следующих ступенях, где для анализа используются следующие младшие разряды. При пересечении log(p) ступеней используются все log(p) биты в двоичном представлении s и t.

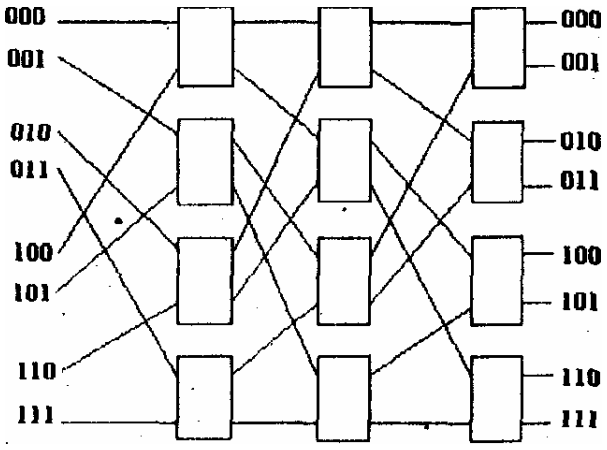


Рис 1.22, Delta сеть для 8-ми процессоров(27)

На рис. 1.23 показана передача сообщения в 8-ми процессорной *delta* сети из процессора 2(010) в банк 7(111) и из процессора 6(110) в банк 4(100). Этот рисунок иллюстрирует важное свойство *delta cemu.* Когда процессор 2(010) связывается с банком 7(111), он блокирует путь из процессора 6(110) в банк 4(100). Связывающее звено AB входит в оба пути, Отсюда видно, что в *delta* сети обращение одного процессора к банку памяти может запретить обращение другого процессора к другому банку памяти. Сети с этим свойством относятся к *блокирующим* сетям (blocking networks).. На основе *delta* сети строятся параллельные компьютеры BBN Butterfly, IBM RP-3, NYU Ultracomputer.

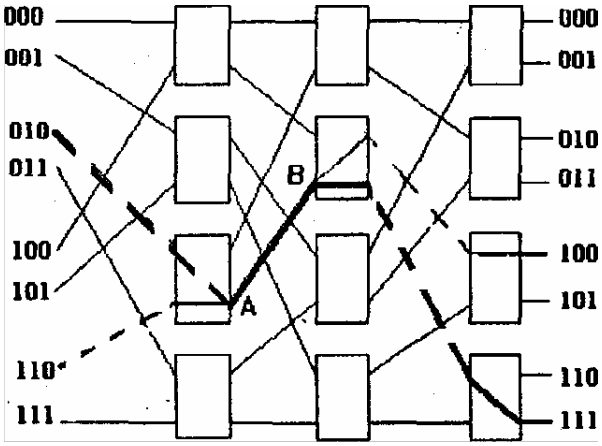


Рис. 1.23. Блокирование пути одним сообщением

1.4. ЗАКЛЮЧЕНИЕ

На рис. 1.24. приведена классификация различных существующих моделей компьютеров, а в табл, 3 [106] некоторые наиболее известные суперкомпьютеры и минисуперкомпьютеры.

RAM и PRAM модели соответствуют уравнению (1).

RAM модель, которая поддерживается компьютерами с жесткой последовательностью операций (последовательные), может быть рассмотрена как SRSW машина.

ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ			
машины с прямым доступом к памяти		ВС со случайным доступом к локальной памяти и косвенным доступом к локальной памяти других машин	
RAM	PRAM	FITM	FLTM
последовательные машины	машины с разделяемой	машины с жесткой топологией	машины с динамической топологией
	памятью	с распределенн	ой памятью
	SRSW	гиперкуб	пол несвязная
	SRMW	ссс	ОМЕГА
	MRSW	древовидная	Cray-YMP
	MRMW	пирамидальная	VPP 500
		кольцевая	Sym metre
		заезда	BBN ВииегЛу
			IBM RP-3
компыотер=данные+операции		компыотер=данные + опрерации + обмен данныи	

Рис. 1.23- Компьютерные модели

PRAM модель реализована в ВС с разделяемой памятью. В этих машинах допускается одновременный доступ к общей разделяемой памяти непосредственно через сеть связи. Но, несмотря на всю привлекательность математической организации PRAM модели, реально не было создано ни одного процессора, поддерживающего эту модель в полном объеме с соответствующими функциональными требованиями.

Модель FITM используется в компьютерах с распределенной памятью. В таких компьютерах реализуется случайный доступ к собственной локальной памяти каждого процессора и косвенный доступ к памяти других процессоров с помощью транзитной пересылки через связи с соседними процессорами или с помощью общей сети связи.(29)

Табл. 1.3

Компьютер	Фирма производитель	Количество процессоров	Max. про-ИЗВОД. (Мфлопс)	Операц система
CRAY-1	Cray Reseach	1	160	COS
CRAY X-MP	Cray Reseach	2,4	210/пр	COS Unix
CRAY-2	Cray Reseach	4	2000	Unix
CRAY Y-MP	Cray Reseach	8	2666	COS, Unix
CRAYY-MPC-90	Cray Reseach	16	16000	Unix
CYBER 205	C DC	1	400	vsos
ETA'''	ETA Systems	2,4,6,8	800/пр	VSOS Unix
FACOM VP-400	Fuiitsu	1	1142	MVS
S-810/20	Hitachi	1	630	MVS
SX-2	NEC	1	1333	ATSS-AF
SX-3	NEC	1,2,4	5500/Пр	SXOS.Unix
FX/8	Alliant Сотр	S	94	Unix
C-l	Convex	1	60	Unix
SCS-40	ScientificComp	1	44	COS
FPS Series T	Floating Point	16-16384	t6/пр	Unix
VAX 9000VP	DEC	4	500	VMS, Unix
DECmpp 12000	DEC	1024-16384	1200	Unix
Amelec 2010	Symull Systems	4-512	215	React- Kernel
GP 1000	BBC AC1	2-256	•	Chrysalis
CM-2	Thinking Mach	65536	2400	Unix
CM-5	Thinking Mach	1024-16384	130000	Unix
ELXSl 6400	ELXSI	1-12	14	Unix

Mullimax	Encore	2-20	20	Unix
MPP	Goodvear	16384	470	•
Sequent	Sequent omp	2-30	*	Unix
iPSC/2	Intel	32-128	1000	Unix
iPSC/860	Intel	8-128	7680	NX/2
Paragon	Intel	64-409й	300000	Unix
NCube/10	NCube	4-1024	500	Vertex
NCubc/2	NCiibe	32-8192	27000	Vertex
SuperNode 1000	Parsys	16-1024	1600	1DR1S
VOLVOX	Arc hi pel	8-48	3000	Unix
GC	P.irsytec	64-16384	400000	Unix

(30)

Некоторые FITM машины были реализованы. Наиболее популярна топология связи процессорных элементов в виде сетки процессоров (llliac IV/NASA, Clip4/Univ. College of London, MPP/GoodYear, DAP/ICL, MPl/MasPar, SYMPATI,-). CM-1 компании Thinking Mashinc Corporation, iPCS — это пример машины с гипсркуби ческой топологией размерностью 12. ВС СМ вначале представлялась **SIMD** моделью, а позже как MIMD. Также достаточно хорошо была изучена пирамидальная структура; в компьютере Sphinx/Univ. В ParisXl— France реализована бинарная структура с мульти-SlMD управлением, остальные структуры -кватернарные (четверичные) только с SIMD управлением, например PAPIA/Univ. of Pavia-ltaly и GAM/Georges Mason Univ.-USA. Колумбийским Университетом (Columbia University/ New-York) были исследованы три вида топологии связей в параллельных не фон Нэймановских машинах, предназначенных для обработки баз данных.

Таким образом, изучение архитектуры, организации параллельных вычислительных систем, а так же тенденций развития показывает, что:

1. Параллелизм в настоящее время является вездесущим, он присутствует и в проектировании ВС и в организации вычислительных процессов.
2. Достижение современных технологии в коммуникации, таких как FDD1, HiPP или SONET позволяет строить мощные вычислительные системы распределенных ресурсов, где можно обрабатывать разнотипные задачи.
3. Задачи планирования играют особую роль в повышении эффективности вычислений систем любого типа.
4. Тенденции развития ВС требует нового осмысления понятия Заданий и Ресурсов (объекты задачи планирования), а так же новую схему для их распределения.
5. Масштаб современных ВС, особенности их конфигурации и управления, усложнил процесс планирования- Поэтому, для получении эффективные решения нужно рассматривать задачи планирования на уровне се составляющих (подзадач).

ГЛАВА 2(31)
ПРИНЦИПЫ ПОСТРОЕНИЯ И ФУНКЦИОНИРОВАНИЯ ОПЕРАЦИОННЫХ СИСТЕМ В ТРАДИЦИОННЫХ ВС
ВВЕДЕНИЕ

Данный раздел посвящен описанию принципов построения и структурной организации операционных систем (ОС) в традиционных вычислительных системах. Дана классификация поколений ОС, их функциональные особенности и современные виды. Показано, что основой любой современной вычислительной системы является операционная система, от которой зависит как функционирование вычислительной установки в целом, так и эффективность прохождения работ через нее.

В разделе рассматриваются следующие вопросы.

- Что такое операционная система ?
- История развития операционных систем.
- Концепция построения операционных систем (процесс, файлы, вызов, ядро).
- Классификация операционных систем и их структурные особенности.
- Понятие генерации, инсталляции, инициализации ОС.
- Ядро операционной системы, структура, состав, взаимодействие.
- Общая схема загрузки операционной системы.

- Общая схема функционирования операционной системы (на примере MS DOS).

2.1. ЧТО ТАКОЕ ОПЕРАЦИОННАЯ СИСТЕМА?

Операционная система ЭВМ или отдельной информационно - вычислительной системы представляет собой совокупность программ, выполняющих две основных функции.

- Эффективное управление ресурсами системы, обеспечение (в случае необходимости) их распределения между несколькими пользователями и контроль за выделением ресурсов для одновременного выполнения многих задач.
- Предоставление набора услуг, обеспечивающего пользователю удобный интерфейс, который приспособлен к его нуждам лучше, чем интерфейс физической машины; этим интерфейсом является интерфейс "виртуальная машина", который предоставляет набор средств для (32) управления информацией и реализацией прикладного программного обеспечения.

Программы, выполняемые вычислительной системой, по функциональному назначению подразделяются на два класса: прикладные и системные. Операционные системы составляют важную часть программ второго класса.

Операционная система — программная среда, обеспечивающая поддержку работы всех программ, их взаимодействие с аппаратными средствами вычислительной системы, а также предоставляющая пользователю возможности общего управления вычислительной системой и использования ресурсов ВС.

Операционные системы различаются архитектурой, возможностями, требуемыми ресурсами для их функционирования, набором сервисных функций, ориентацией на типы используемых микропроцессоров и другими особенностями. Развитие операционных систем неразрывно связано и в значительной степени определялось развитием технического обеспечения вычислительных систем, функциональными и структурными особенностями компонентов ВС и внедрением новых принципов организации вычислительного процесса на аппаратном и программном уровне.

2.2. ИСТОРИЯ РАЗВИТИЯ ОПЕРАЦИОННЫХ СИСТЕМ

Поколения операционных систем:

- *НУЛЕВОЕ ПОКОЛЕНИЕ* Для первых цифровых вычислительных машин не существовало операционных систем. Пользователь получал возможность полностью распоряжаться вычислительной системой и ее ресурсами на строго определенное время. Это были машины значительных размеров, сделанные на ненадежных электронных лампах. Программы, написанные на машинном языке, работали на "голой" машине без какого либо промежуточного системного обеспечения. Пользователи вводили свои программы, написанные на машинном языке, в компьютер с помощью машинных носителей (перфокарт, перфолент или, в самых первых компьютерах, с помощью коммутационной панели), загружали программы, ждали их выполнения и уносили распечатки с выходными данными. В некотором смысле пользователи-программисты выполняли функции операционной системы, так как они управляли всеми деталями действий машины и эффективность использования ее ресурсов целиком зависела от их действий. Прямое взаимодействие с машиной (пошаговое выполнение команд, непосредственный контроль и изменение состояния ячеек памяти) было главным средством отладки программы. Программист должны были сами писать программы выполнения процедур (33) ввода/вывода для своих программ. Печать символа, например, требовала много машинных команд, так как в компьютере не было нужной функции. Если этого не было в вашей программе, то этого не было нигде. Отсутствовало также управление памятью. Каждая программа имела доступ ко всему пространству памяти — она пользовалась тем, что было ей нужно, а остаток не был задействован.

Загрузка системы включала в себя запись программ в двоичном коде с помощью переключателей на передней панели в память машины, что занимало очень много времени и не гарантировало отсутствие ошибок в управлении. Такая работа оказалась крайне неэффективной, так как дорогостоящее оборудование использовалось неэкономно.

Первыми программными системными средствами, повышающими эффективность работы пользователя

на вычислительной установке, а отсюда и работу программиста, были, с одной стороны, средства автоматизации разработки программ (ассемблеры, компиляторы, отладчики), а с другой — подпрограммы ввода/вывода.

- *ПЕРВОЕ ПОКОЛЕНИЕ.* В середине 50-х годов в компьютерах стала использоваться новая элементная база — транзисторы. Это сделало машины более надежными, и сфера их применения расширилась. Однако принцип работы практически не изменился; программист или профессиональный оператор загружал программу, ждал пока она будет выполнена, затем снова ждал, когда будут напечатаны результаты — только потом загружалась следующая программа. Стало ясно, что такое использование дорогого быстродействующего оборудования является крайне неэффективным. Одна из проблем, которую должны были решить операционные системы, такова: ввод и вывод данных занимали время, но при этом не полностью использовались вычислительные возможности центрального процессора ЩП) т.к скорости ЦП и устройств ввода/вывода несоизмеримы. Ожидая, пока произойдет ввод/вывод, ЦП обычно работал вхолостую, растрачивая свой ресурс.

Первым решением проблемы повышения эффективности использования ЦП была простая пакетная обработка. При таком подходе перфокарты многих программ считывались дополнительным низкоскоростным компьютером на магнитную ленту, подготавливая входную очередь заданий — пакет. ЦП обрабатывал программы пакета одну за другой и выводил результаты на другую магнитную ленту. Так как накопители на магнитной ленте были быстрее, чем устройство перфоввода и принтеры, то положение дел улучшилось. Проблема простоя ЦП во время ввода/вывода была решена за счет ускорения этого процесса.

К концу 50-х годин появились первые пакетные мониторы. Эти программы создавались *с* целью выполнения всей последовательности работ (34) (организации данных и выполнения прикладных программ), подготовленных заранее, и автоматизации процесса перехода от одной работы к другой- Основной функцией такой системы было управление ресурсами: памятью, процессором, вводом/выводом. Автоматизация управления ресурсами ВС вызнала появление функций защиты ВС при выполнении работ от ошибок:

- ограничение времени доступа к процессору, чтобы устранить блокирование всей работы из-за зацикливания в одной программе;
- надзор за вводом/выводом, чтобы избежать циклического обращения к периферийным устройствам по ошибке;
- защита зоны памяти, постоянно занимаемой монитором, от ошибок адресации в пользовательских программах. Обеспечение указанных функций привело к расширению возможностей машин - измерение и учет времени, ограничения на использование некоторых инструкций, защита памяти.

Использование пакетных мониторов существенно повысило производительность вычислительных систем и уменьшило непроизводительные затраты машинного времени при смене заданий. Однако это повышение и значительной степени ограничивалось тем, что при выполнении операций ввода/вывода простаивал процессор. Стали использоваться два спаренных компьютера. Программы выполнялись на одном из них (главном), оснащенном устройством с повышенной скоростью передачи данных на магнитную ленту и считывания данных с магнитной ленты. Вспомогательный компьютер служил для записи данных с перфокарт на ленту и вывода результатов вычислений с магнитной ленты на печать- Операционная система этих. систем состояла из команд, расположенных между программами, подлежащими считыванию, и резидентной управляющей программы для интерпретации этих команд написанных с использованием JCL (язык управления заданиями).

Предварительное планирование очередности работ позволяло работать обеим машинам параллельно и тем самым более полно использовать возможности главного процессора. Недостаток такой организации — негибкость системы (временные ограничения для ввода заданий, долгое ожидание результатов). Описанные системы последовательной пакетной обработки получили широкое распространение в начале 60-х годов.

- *ВТОРОЕ ПОКОЛЕНИЕ.* В период с 1960 по 1970 г. значительный прогресс в области технологии материалов и в разработке принципов функционирования операционных систем позволил устранить недостатки систем пакетной обработки и перейти к системам с коллективным доступом.(35)

Введение автономных специализированных процессоров для передачи информации (каналов, или

устройств обмена) позволило освободить центральный процессор от управления вводом/выводом.

Мультипрограммирование, связанное с разделением памяти сразу для нескольких работ, повысило производительность центрального процессора за счет одновременной работы процессора и нескольких устройств ввода/вывода.

Для равномерной загрузки оборудования в таких системах требуется тщательный подбор задач, выполняемых в режиме мультипрограммирования. В большинстве случаев программы не будут подходить друг другу, и невозможно обеспечить равномерность загрузки. Однако при запуске более чем двух программ одновременно ЦП может работать с пользой большую часть времени.

Конечно, функционирование ВС в многозадачном режиме требует использования более сложного системного программного обеспечения (операционной системы), чтобы следить за тем, где расположена та или иная программа, в какой стадии выполнения и, совместно с аппаратным обеспечением, предотвращать взаимодействие между программами.

Управление памятью в первых мультипрограммных системах было достаточно простым; следить, чтобы каждая программа занимала только выделенную ей часть памяти и оставалась там вплоть до завершения и вывода.

Однако случалось, что программа была слишком большой, чтобы уместиться в отведенной ей части, или даже настолько большой, что не умещалась во всей памяти машины. В этом случае программы должны были делиться на оверлеи — отдельные части программы, которые вызывали друг друга, но это создавало дополнительные трудности программисту, так как именно ему приходилось делить программу на части подходящего размера и следить за бесконфликтным вызовом одной программы другой.

Лучшим решением этой проблемы была виртуальная память, где доступное пространство памяти ВС для каждой программы выглядело больше, чем имеющийся в действительности объем реальной оперативной памяти. Программа могла быть поделена на страницы (обычно одинакового размера). Те страницы программы, которые были активны (выполнялись команды или осуществлялся доступ к данным), сохранялись в оперативной памяти, тогда как те, которые не использовались, временно записывались на быстродействующий диск. Этот процесс полностью управлялся операционной системой и был незаметен программисту, который мог писать программы, занимающие больше памяти, чем было физически доступно.(36)

Разработчики также создали библиотеку стандартных программ ввода/вывода, чтобы пользователям не приходилось создавать их с нуля. После этого операционные системы имели собственные функции вывода символа на экран и выполняли другие задачи ввода/вывода. Было логичнее просто использовать эти функции, чем каждый раз повторять их в каждой программе. Так родилась идея системных функций, которые до сих пор являются частью большинства операционных систем.

Хотя мультипрограммирование заставляло ЦП работать большую часть времени, оно никак не помогало решить другую проблему: эффективное использование всей вычислительной установки в целом. Программист обычно приносил программу в виде колоды перфокарт в машинный зал и отдавал их оператору. Затем нужно было ждать, когда она выполниться, что занимало иногда несколько часов т.к. задача выполнялась в пакете. И. наконец, программист забирал распечатки с результатами из машинного зала " смотрел, что получилось. Времени обычно хватало на 2-3 таких прогона в день, так что отладка была очень длительной процедурой (поразительно, что какое-то программное обеспечение все-таки было написано).

Решением проблемы нехватки времени было введение систем разделения времени. При разделении времени несколько терминалов связывались с одним ЦП, и каждый получал серию маленьких промежутков (квантов) времени ЦП. Так как человек обычно работает медленнее процессора, то пользователь не замечал, что между нажатием "l" и "s" при набивке строки "1s" ЦП обслуживает еще нескольких пользователей. Результатом явилась интерактивная система, где кажется, что компьютер отзывается сразу после ввода. Это _режим_ кажущегося совмещения. Основной целью функционирования систем с разделением времени является улучшение работы пользователя. В наиболее развитых системах с разделением времени результаты пользователю отправлялись по мере их получения, что еще больше усиливало эффект "индивидуального" использования ресурсов ВС пользователем.

Первой широко используемой системой с разделением времени была операционная система UNIX, которая с тех пор получила широкое распространение и была перенесена на множество различных машин.

В среде с разделением времени операционная система имеет другие приоритеты, чем пакетная система. Она все еще остается мультипрограммной системой, но акцент делает не на оптимизацию использования

ЦП, а на минимизации времени отклика. Это позволяет пользователю нормально работать и видеть результаты через достаточно короткое время. Естественно, что при подключении к системе слишком большого количества пользователей, время отклика будет увеличивается и характеристики системы будут ухудшаться. Система должна попытаться (37) удовлетворить всех пользователей в равной мере, снабдив каждого из них одинаковой частью машинного времени.

Задача распределения времени ЦП в системах с разделением времени - намного более сложна, чем в пакетных системах. Она решается специальной частью операционной системы, называемой планировщиком Принципы, заложенные в планировщике, являются важными факторами в характеристике системы. В системе с разделением времени каждая программа работает в течение короткого отрезка времени: говорят, что программа получает квант времени. Когда промежуток времени одной программы заканчивается, начинает работать другая программа, несмотря на то, что первая программа не завершена. Это переключение и принципы выбора следующей программы для исполнения определяется приоритетной системой и применяемыми в ВС дисциплинами обслуживания заявок: каждый процесс либо является приоритетным, либо отключается и замещается другим.

Заметим, что когда мы говорим о двух программах, работающих одновременно в системе с разделением времени, мы не имеем в виду точную одновременность, так как ЦП отдает квант времени только одному пользователю. Пользователям кажется, что много программ работают параллельно, но в действительности ЦП запускает программу пользователя А на несколько миллисекунд, затем программу пользователя Б, и т;>к далее, до нескольких десятков. Затем он возвращается, чтобы дать пользователю А следующий квант времени. Только быстрота этого "карусельного" переключения времени ЦП создает иллюзию одновременности для пользователям.

Операционная система с разделением времени должна также управлять ресурсами. Как и и реальном мире, ресурсы — это то, чего всегда не хватает, то есть то, что нужно делить между многими пользователями. Примером ресурсов могут служить принтеры и память. Операционная система поддерживает порядок, осуществляя вывод программ пользователей на принтер и распределяя память так эффективно и справедливо, как это возможно.

Метод обращения с памятью весьма сложен. Один из общих подходов состоит в том, чтобы поделить каждую программу пользователя на сегменты. Сегменты могут соответствовать логическим частям программы, например один сегмент — для команд, один сегмент — для данных и один — для стека. Сегменты могут заноситься в память и удаляться из нее по необходимости. Они также могут быть защищены так, что только одна программа будет иметь к ним доступ. Или наоборот, они могут быть совместными, делая группу данных доступной многим пользователям.(38)

Так как в системе с разделением времени имеется множество пользователей, то возникают дополнительные сложности. Должна быть соблюдена конфиденциальность отдельных программ, так чтобы пользователь не мог случайно или преднамеренно изменить чужую программу либо данные или. что еще хуже, разрушить саму систему. Желательно, чтобы система имела простейший метод контроля, Должно регистрироваться, например, как долго каждый пользователь работал с системой и какие действия он совершал. Именно при разработке систем с разделением времени были введены понятия прерывания и логических устройств. Прерывания и логика обработки прерываний позволило перейти к внедрению операционных систем с принципиально новыми функциональными возможностями.

- *ТРЕТЬЕ ПОКОЛЕНИЕ.* На 70-е годы приходится начало расцвета операционных сисгем, В *у* о время были заложены принципы совместимости операционных систем "снизу вверх". Разрабатываются многорежимные операционные системы, предназначенные для разных конфигураций вычислительных систем. Однако для этих систем характерна труднодоступность пользователя к аппаратным ресурсам вычислительной системы.

70-е годы отмечены двумя событиями:

- Появлением микропроцессоров и постоянным ростом их возможностей. которые позволили обеспечить значительные вычислительные мощности при относительно низкой стоимости.
- Развитием техники передачи данных (телеобработки) и постоянно растущей степени интеграции средств связи в информационно - вычислительные системы. Одновременно с этим опыт эксплуатации вычислительных систем сформировал новые требования пользователей к ним:
- возможность разделения ресурсов, оправданное с экономической точки зрения, и доступ к удаленной информации (ресурсы и информация могут находиться в географически разных местах);

- возможность наилучшего соответствия и адаптации вычислительной системы к структуре прикладных задач, приводящая к децентрализации системы и географическому распределению ее элементов;
- возможность объединения прикладных задач (ранее разъединенных) в единую совокупность объектов на единых системных принципах использования и взаимодействия.

В результате появились новые типы вычислительных систем:

- Сети телеобработки данных, позволяющие организовать взаимосвязь между существующими системами, обмен данными между ними и доступ к удаленным устройствам информационного обслуживания.(39)
- Локальные сети ЭВМ с высокоскоростными каналами передачи информации (10 Мбит/с), географически сконцентрированные и выполняющие функции:
- систем управления;
- систем связи и управления документацией (учрежденческих систем), ориентированных на обмен информацией, создание и хранение документов;
- систем общего пользования, предназначенных, в частности, для создания программного обеспечения.

В тоже время предполагаемое развитие сетей и персональных компьютеров не исключает существование больших централизованных систем с доступом Р. режиме разделения времени, которые остаются экономически оправданными для многочисленных применений. Задачи вычислительной математики в ряде специальных областей требуют развития вычислительных систем очень большой мощности, включающих специализированные мультипроцессоры, для которых должна быть разработана своя архитектура операционных систем.

В 1965 году фирма Bell Telephone Laboratories, объединив свои усилия с компанией General Electric и проектом МАС Массачусетского технологического института, приступили к разработке новой операционной системы, получившей название Mukics. Перед системой Multics были поставлены задачи - обеспечить одновременный доступ к ресурсам ЭВМ большого количества пользователей, обеспечить достаточную скорость вычислений и хранение данных и дать возможность пользователям в случае необходимости совместно использовать данные. Многие разработчики, в последствии принявшие участие в создании ранних редакций системы UNIX, участвовали в работе над системой Multics в фирме Bell Telephone Laboratories, Хотя первая версия системы Multics и была запущена в 1969 году на ЭВМ GE 645, она не обеспечивала выполнение главных вычислительных задач, для решения которых она предназначалась и не было даже ясно когда цели разработки будут достигнуты. Поэтому фирма Bell Laboratories прекратила свое участие в проекте.

По окончании работы над проектом Multics сотрудники Исследовательского центра по информатике фирмы Bell Telephone Laboratories остались без "достаточно интерактивного вычислительного средства". Пытаясь усовершенствовать среду программирования, Кен Томпсон, Дэннис Ричи и другие набросали на бумаге проект файловой системы, получивший позднее дальнейшее развитие а ранней версии файловой системы UNIX. Томпсоном были написаны программы, имитирующие поведение предложенной файловой системы в режиме подкачки данных по запросу, им было даже создано простейшее ядро операционной системы для ЭВМ GE 645- Томпсон получил возможность изучить машину (40) PDP-7, однако условия разработки программ потребовали использования кросс-ассемблера дли трансляции программы на машине с системой GECOS и использования перфоленты для ввода в PDP-7. Для того, чтобы улучшить условия разработки. Томпсон и Ричи выполнили на PDP-7 свой проект системы, включившим первую версию файловой системы UNIX. подсистему управления процессами и небольшой набор утилит. В конце концов, новая система больше не нуждалась в поддержке со стороны системы (*ECOS в качестве операционной среды разраоотки и могла поддерживать себя сама. Новин система получила название UNIX, по сходству с Mullics его придумал ешс один сотрудник Исследовательского центра по информатике Брайан Керниган.

Несмотря на то что эта ранняя версия системы UNIX уже была многообещающей, она не могла реализовать свой потенциал до тех пор, пока не получила применение в реальном проекте. Так, для того, чтобы обеспечить функционирование системы обработки текстов для патентного отдела фирмы Bell Telephone Laboratories, в 1971 году система UNIX была перенесена на ЭВМ PDP-11. Система отличалась небольшим объемом- 16 Кбайт для системы, 8 Кбайт для программ пользователей, обслуживала диск объемом 512 Кбайт и отводила под каждый файл не более 64 Кбайт. После своего первого успеха Томпсон

собрался было написать Ш1Я новой системы транслятор с Фортрана, но вместо этого занялся языком Би (B), предшественником которого явился язык BCPL. Би был языком интерпретирующего типа со всеми недостатками, присущими подобным языкам, поэтому Ричи переделал его в новую разновидность, получившую название Си (C) и разрешающую генерировать машинный код, объявлять типы данных и определять структуру данных. В 1973 году система была написана заново на Си. Это был шаг, неслыханный для того времени, но имевший огромный резонанс среди широкого круга пользователей. Количество машин фирмы Bell Telephone Laboratories, на которых была инсталирована система, возросло до 25, в результате чего была создана группа по системному сопровождению UNIX внутри фирмы.
В то время корпорация AT&T не могла заниматься продажей компьютерных продуктов в связи с соответствующим соглашением, подписанным ею с федеральным правительством в 1956 году, и распространяла систему UNIX среди университетов, которым она была нужна в учебных целях. Следуя букве соглашения, корпорация AT&T не рекламировала, не продавала и не сопровождала систему. Несмотря на это, популярность системы устойчиво росла.

В 1974 году Томпсон и Ричи опубликовали статью, описывающую систему UNIX, в журнале Communications of the ACM, что дало еще один импульс к распространению системы. К 1977 году количество машин, на которых функционировала система UNIX,(41) увеличилось до 500, при чем 125 из них работали в университетах. Система UNIX завоевала популярность среди телефонных компаний, поскольку обеспечивала хорошие условия для разработки программ, обслуживала работу в сети в режиме диалога и работу в реальном масштабе времени (с помощью системы MERT), Помимо университетов, лицензии на систему UNIX были переданы коммерческим организациям. В 1977 году корпорация Interactive Systems стала первой организацией, получившей права на перепродажу системы UNIX с надбавкой к цене за дополнительные услуги, которые заключались в адаптации системы к функционированию в автоматизированных системах управлении учрежденческой деятельностью. 1977 год также был отмечен "переносом" системы UNIX на машину, отличную от PDP (благодаря чему стал возможен запуск системы на другой машине без изменений или с небольшими изменениями), а именно на interdaia 8/32.

С ростом популярности микропроцессоров другие компании стали переносить систему UNIX на новые машины, однако ее простота и ясность побудили многих разработчиков к самостоятельному развитию системы, в результате чего было создано несколько вариантов базисной системы. За период между 1977 и 1982 голом фирма Bell Laboratories объединила несколько вариантов, разработанных в корпорации AT&T. в один. получивший коммерческое название UNIX версия III. В дальнейшем фирма Bell Telephone Laboratories добавила в версию III несколько новых особенностей, назвав новый продукт UNIX версия V, и эта версия стала официально распространяться корпорацией AT&T с января 1983 года. В то же время сотрудники Калифорнийского университета в Бэркли разработали вариант системы UNIX, получивший название BSD 4.3 для машин серии VAX и отличающийся некоторыми новыми, интересными особенностями.

К началу 1984 года система UNIX была уже инсталлирована приблизительно на 100000 машин по всему миру, причем на машинах с широким диапазоном вычислительных возможностей — от микропроцессоров до больших ЭВМ — и разных изготовителей. Ни о какой другой операционной системе нельзя было бы сказать того же. Популярность и успех системы UNIX объяснялись несколькими причинами;

Система написана на языке высокого уровня, благодаря чему ее легко читать, понимать, изменять и переносить на другие машины. По оценкам, сделанным "или, первый вариант системы на Си имел на 20-40 % больший объем и работал медленнее по сравнению с вариантом на ассемблере, однако преимущества использования языка высокого уровня намного перевешивают недостатки;(42)

- Наличие довольно простого пользовательского интерфейса, в котором имеется возможность предоставлять все необходимые пользователю услуги.
- Наличие элементарных средств, позволяющих создавать сложные программы из более простых.
- Наличие иерархической файловой системы, легкой *в* сопровождении и эффективной в работе.
- Обеспечение согласования форматов в файлах, работа с последовательным потоком байтов, благодаря чему облегчается чтение прикладных программ.
- Наличие простого, последовательного интерфейса с периферийными устройствами.

- Система является многопользовательской, многозадачной; каждый пользователь может одновременно выполнять несколько процессов.

Архитектура машины скрыта от пользователя, благодаря этому облегчен процесс написания программ, работающих на различных конфигурациях аппаратных средств.
Простота и последовательность вообще отличают систему UNIX и объясняют большинство из вышеприведенных доводов в ее пользу.
Хотя операционная система и большинство команд написаны на Си, система UNIX поддерживает ряд других языков, таких как Фортран, Бейсик, Паскаль, Ада, Кобол, Лисп и Пролог. Система UNIX может поддерживать любой язык программирования, для которого имеется компилятор или интерпретатор, и обеспечивать системный интерфейс, устанавливающий соответствие между пользовательскими запросами к операционной системе, и набором запросов, принятых в UNIX.

- *ЧЕТВЁРТОЕ ПОКОЛЕНИЕ* В конце 70-х годов технология изготовления БИС понизила стоимость и размеры одного транзистора до таких пределов, что стало целесообразным производить персональные компьютеры. Это вызвало поворот к ситуации с одной программой и одним пользователем, как во времена первых компьютеров, и возможность полного владения и управления ресурсами системы. Первый массовый рынок персональных компьютеров включал такие модели, как IMSA1 8080, Radio Sliach TRS-80 Model I, Apple II и позднее IBM PC.

Операционные системы персональных компьютеров (ПК) могли быть достаточно простыми. Им нужно было обслуживал, действие только одной программы, гак что их основной задачей стало управление файлами и обеспечение ряда полезных для программиста функций. Для первого поколения ПК пользователем был программист, работавший на языке ассемблера, так как первые машины не имели возможностей для вызова средствами языков высокого уровня (обычно BASIC) (43) системных функций, встроенных в операционную систему. Первыми операционными системами на оснащенных процессорами Intel компьютерах были CP/M и — с появлением IBM PC — MS-DOS MS-DOS до сих пор наиболее распространенная операционная система и замечательно сохранилась за десятилетие—достаточно большое время в компьютерном мире.

Однако становилось все более ясно, что однозадачный персональный компьютер не является окончательным вариантом. Операционная система, которая может загружать и запускать только одну программу, ограничивает производительность труда пользователей.

Разработчики попытались обеспечить ограниченные формы мультипрограммности в MS-DOS с помощью использования TSR — программ (программ которые завершаются и остаются резидентными в памяти), но это решение далеко от совершенства. С одной стороны, TSR— программы не обеспечивают реальную многозадачность: если вы выведете TSR — программу на экран, то программа, с которой вы работали до того, остановится. С другой стороны, TSR— программы в связи с ограничениями в системе прерываний конфликтуют друг с другом, так как архитектура MS-DOS не создавалась для их использования.

Термин "многозадачность" аналогичен термину " мультипрограммирование". Оба относятся к системе, в которой несколько программ могут находиться в памяти и выполняться одновременно. Однако "задача" означает не совсем то же самое, что "программа", как мы увидим позднее, когда будем рассматривать процессы и цепочки.

Можно запустить версию многозадачной многопользовательской операционной системы на персональном компьютере, используя такие системы как Unix или Xenix. Но они в действительности используют персональный компьютер как миникомпьютер, создавая систему с разделением времени и принося с этим всю оболочку многопользовательской системы. Однако такие системы не будут оптимальными с позиций одного пользователя.

Многие компании предлагали операционные системы, которые имели некоторые черты многозадачной системы, например, DESQView фирмы QuarterDeck Office System, Windows фирмы Microsoft и PC-MOS фирмы Software Link. Среди систем такого класса следует выделить операционную систему OS/2, имеющую ряд отличительных особенностей, позволяющих определить ее такими фирмами как IBM и Microsoft как наиболее перспективною ОС.

Таким образом, OS/2 является основной однопользовательской многозадачной операционной системой, разработанной специально для персонального компьютера 80-е годы УТО период интенсивной разработки и распространения дружественных операционных систем.(44)

- *ПЯТОЕ ПОКОЛЕНИЕ.* Современные операционные системы обеспечивают дружественный пользовательский интерфейс для различных категории пользователей. Операционные системы реализуют такие элементы человеко-машинного взаимодействия, как многооконный интерфейс с пользователем, переключение между программами (и их окнами) и т.д. Операционные системы содержат мощный набор графических примитивов, обеспечивающих стандартизацию работы на различных типах графических устройств.

Таким образом, в течение последних 30 лет был разработан ряд удачных операционных систем, некоторые из них приведены в таблице табл. 2.1,

2.3. КОНЦЕПЦИЯ ПОСТРОЕНИЯ ОПЕРАЦИОННЫХ СИСТЕМ

Часть операционной системы, постоянно находящаяся *в* основной памяти, называется *ядром системы пли резидентной* частью операционной системы. Программы, вызываемые в основную память для выполнения определенных функций и не хранящиеся там постоянно, называются *транзитными программами* или просто *транзитами.*

Транзиты вызываются в участок основной памяти, который называется транзитной областью управляющей программы. Вся область основной намяти, занимаемая ядром и транзитами, называется областью управляющей программы. Остальная часть основной памяти образует область проблемных программ. Разделение управляющей программы на ядро и транзиты позволяет минимизировать область управляющей программы и, соответственно, увеличить область памяти выделяемой для проблемных программ.

В силу того, что транзиты вызываются на одно и тоже место основной памяти, перекрывая друг друга, они иногда называются еще перекрывающимися программами или про1раммами с запланированным перекрытием,

Операционная система спроектирована для работы в различных режимах. Поэтому на вход системы может одновременно поступать большое количество заявок, выполнить которые сразу невозможно просто из-за того, что нет достаточного колпчсс1ва физических ресурсов. К физическим ресурсам системы относятся центральный процессор, место в основной и вспомогательной памяти, отдельные программы, устройства управления вводом/выводом, каналы, таймер, консоль оператора. Все заявки, поступающие на вход системы, делятся на независимые задания, являющиеся внешней единицей работы для операционной системы. Любая выполняемая работа в системе, называется процессом, являющимся автоматическим объектом системы, которому она выделяет ресурсы.(45)

Табл. 2.1

Операционная система	Год разработки	Разработчик
Atlas	1959	Манчестерский университет, Англия
CTSS	1963	Масачусетский технологический институт (MT1),США
OS/360 (I)	1964	IBM. США
THE	1967	Эйндховенский технологический университет, Голландия
RC 4000	1970	Университет города Орхуса, Дания
OS/360 (21)	1970	IBM. США
UN1X(1)	1972	Bell Laboratories, США
Multics	1975	Bell Laboratories, MTI, США
MVS	1976	IBM, США
VMS	1980	Digital Equipment Corp., США
CP/M	1983	Digital Research Inc., США
MS-DOS	1983	Microsoft Corp., США

UNIX (7.5)	1983	Bell Laboratories, США
UNIX(V»	1985	Bel! Laboratories, США
OS/2 Warp	1994	IBM. США
WINDOWS NT	1993-94	Microsoft Corp., США

Выполнение пользовательских процессов в "операционной системе осуществляется на двух уровнях: уровне пользователя и уровне ядра. Когда процесс производит обращение к операционной системе, режим выполнения процесса переключается с режима задачи (пользовательского) на режим ядра; операционная система пытается обслужить запрос пользователя, возвращая код ошибки в случае неудачного завершения операции. Даже если пользователь не нуждается в каких-либо определенных услугах операционной системы и не обращается к ней с запросами, система еще выполняет учетные операции, связанные с пользовательским процессом, обрабатывает прерывания, планирует процессы, управляет распределением памяти и т.д. Большинство вычислительных систем разнообразной архитектуры (и соответствующие им(46) операционные системы) поддерживают большее число уровней, чем указано здесь, однако уже двух режимов: режима задачи и режима ядра, вполне достаточно для операционной системы.

Основные различия между этими двумя режимами.

- В режиме задачи процессы имеют доступ только к своим собственным инструкциям и данным, но не к инструкциям и данным ядра (либо других процессов).
- В режиме ядра процессам уже доступны адресные пространства ядра и пользователей. Например, виртуальное адресное пространство процесса может быть поделено на адреса, доступные только в режиме ядра, и на адреса, доступные в любом режиме.

Некоторые машинные команды являются привилегированными и вызывают возникновение ошибок при попытке их использования в режиме задачи. Например, в машинном языке может быть команда, управляющая регистром состояния процессора; процессам, выполняющимся в режиме задачи, она недоступна.

Проще говоря, любое взаимодействие с аппаратурой описывается в терминах режима ядра и режима задачи и протекает одинаково для всех пользовательских программ, выполняющихся в этих режимах. Операционная система хранит внутренние записи о каждом процессе, выполняющемся в системе. На Рисунке 2.!. показано это разделение: ядро делит процессы A, B, C и D, расположенные вдоль горизонтальной оси, аппаратные средства вводят различия между режимами выполнения, расположенными по вертикали. Несмотря на то, что система функционирует в одном из двух режимов, ядро действует от имени пользовательского процесса. Ядро не является какой-то особой совокупностью процессов, выполняющихся параллельно с пользовательскими, оно само выступает составной частью любого пользовательского процесса.

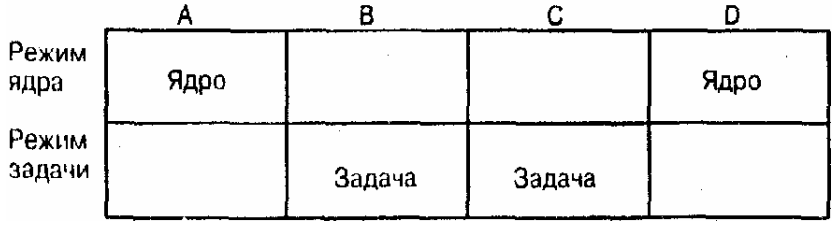


Рис 2.1. Процессы и режимы их выполнения(47)

2.4. КЛАССИФИКАЦИЯ ОПЕРАЦИОННЫХ СИСТЕМ И ИХ СТРУКТУРНЫЕ ОСОБЕННОСТИ

Различают следующие типы операционных систем:

- Однопрограммные ОС.
- Многопрограммные ОС.
- Однопроцессорные ОС.
- Многопроцессорные ОС.
- Распределенные ОС.
- Виртуальные ОС.

Система работает в *однопрограммном* режиме если в системе находится одна или несколько задач, но

все ресурсы системы отданы одной задаче, и она не может быть прервана другой. Задача возвращает ресурсы только при нормальном или аварийном завершении.

Система работает в *многопрограммном* режиме, если в ней находится несколько задач в разной стадии исполнения, и каждая из них может быть прервана другой с последующим возвратом.

Многопрограммный режим в однопроцессорной системе - это организация вычислительного процесса с планированием во времени. При этом операционная система обеспечивает выполнение активизированных процессов на одном и том же оборудовании, разделяя (синхронизируя) их работу во времени. В функции такой операционной системы также входит защита влияния одного процесса на другой. Основные заботы о синхронизации взаимодействующих процессов возложены на программиста.

Многопрограммный режим в многопроцессорной системе — это планирование во времени и "в пространстве. В этом режиме операционная система, выполняя одну из своих основных функций (обеспечение эффективности работы ресурсов), должна распределять активизированные процессы по процессорам (в пространстве) и синхронизировать их работу во времени. В том случае, если количество задач больше количества процессоров, задача планирования, диспетчеризации усложняется. В связи со сложностью решения задач распределения процессов по процессорам, их решение выполняется или до активизации процессов в многопроцессорной системе (статическое планирование), или решаются простыми способами, не дающими оптимального решения.

В настоящее время применение систем массового распараллеливания и распределенные систем обработки информации, когда вычислительная система может в большинстве случаев выделить столько вычислительных ресурсов, сколько требуется, временную координату планирования можно исключить, что значительно облегчает задачи операционной системы по планированию и организации вычислительных процессов.(48) 48 'Организация вычислительных процессов в ЭВМ, комплексах, сетях и системах

Режимы эксплуатации вычислительной системы

Бурный рост потребностей в высокопроизводительных и надежных средствах вычислений привел к появлению и развитию вычислительных систем (ВС). Пока рано говорить о достаточно полной и точной классификации ВС, так как развитие их структур и способов функционирования продолжается, и почти каждая новая разработка ВС вносит новые идеи, заставляющие изменять принятую классификацию. Тем не менее, можно указать некоторые, уже установившиеся признаки, по которым удается классифицировать существующие и проектируемые ВС (рис.2.2).

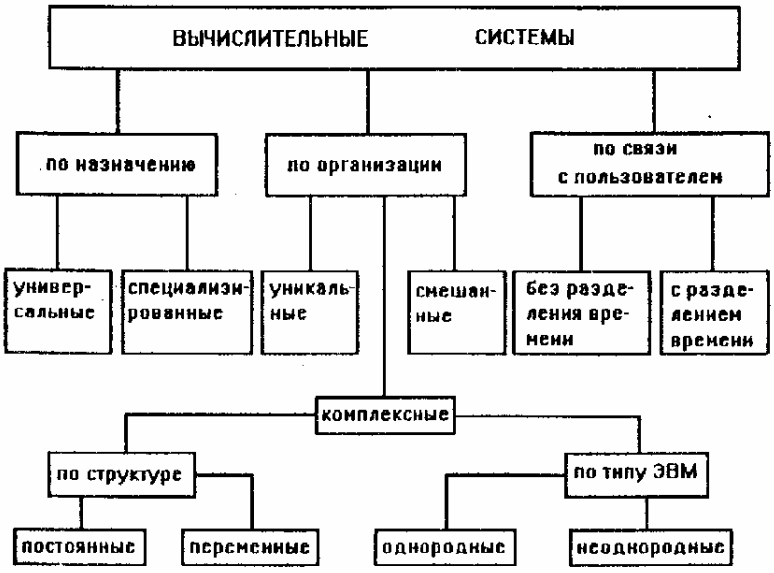


Рис 2.2

В прицеленной классификации не нашли отражения способы использования ВС при решении задач и организации вычислительного процесса и системе машин. Первый опыт эксплуатации НС показал, что большие возможности этих систем могут быть полностью раскрыты и использованы только при

применении эффективных методов и средств организации работы. Обзор литературы по применению вычислительной техники позволяет обощить различные формы эксплуатации ВС и представить их в (49) виде определенной классификационной схемы (рис. 2.3). При этом следует заметить, что в настоящее время наибольшее внимание уделяется организации работ в параллельных системах.

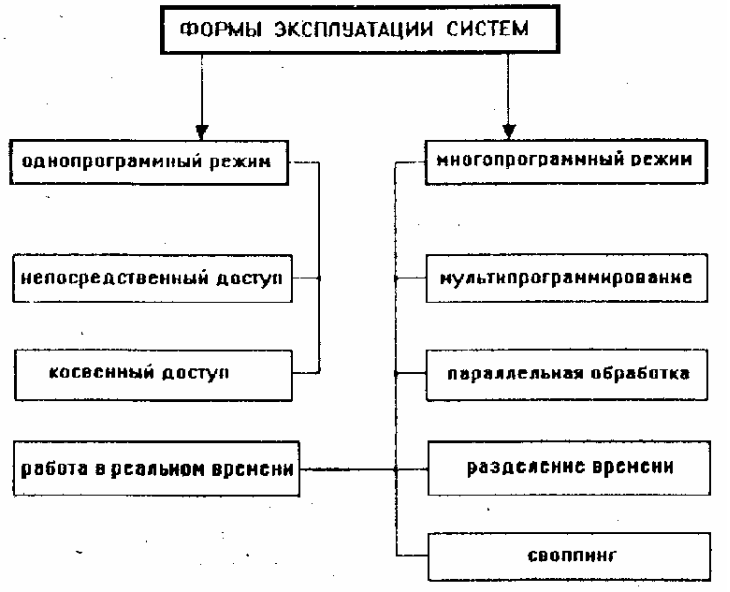


Рис. 2.3. Однопрограммный режим работы

Как уже отмечалось, вычислительная система (машина) работает в однопрограммном режиме, если в системе находится одна или несколько задач, но все ресурсы отданы одной задаче, и она не может быть прервана для выполнения другой.

В начальной стадии применения ЭВМ пользователь сам работал за пультом управления, осуществляя ввод своей программы, ее запуск и наблюдал за выходом результатов по мере их получения.

Время реакции пользователя при непосредственном доступе велико по сравнению со временем реакции машины, что чрезвычайно невыгодно с точки зрения эффективного использования мощности последней. Действительно, пользователь определенное время набирает с клавиатуры информацию, или обдумывает свое очередное действие, в это время (50) ЭВМ простаивает. При пакетной обработке доступ к ЭВМ осуществляется косвенно: посредством управляющей программы, обеспечивающей переход от одной задачи к другой и контроль за их выполнением. При этом значительно сокращаются непроизводительные потерн машинного времени, повышается эффективность использования ресурсов ЭВМ, но, в то же время, полностью исключается возможность диалога человек — машина.

При пакетном режиме предусматривается последовательная обработка задач. Однако в некоторый момент времени только одна из задач пакета выполняется и занимает все ресурсы, а остальные задачи (программы) пакета находятся в состоянии ожидания своей очереди. Во время выполнения одной из программ пакета запрещается ее прерывание с целью перехода к другой программе пакета. Переход к очередной программе разрешается либо после завершения текущей программы, либо в случае ее аварийного останова.

Для внесения малейшего изменения в программу необходимо ожидать ввода в машину очередного пакета задач. Таким образом, при косвенном доступе к ЭВМ время реакции машины в подавляющем большинстве случаев превышает время реакции пользователя, а это снижает эффективность работы пользователя.

При пакетной обработке задач в однопрограммном режиме функционирования ВС возникает необходимость выбора оптимальной последовательности выполнения задач с заданными сроками реализации на одном обслуживающем устройстве. В случае, если возможности ВС являются постоянными, а заявки (задачи пакета), которые должны быть обслужены, уже имеются в наличии, необходимо определить оптимальную, в некотором смысле, очередность обслуживания заявок.

Многопрограммный режим работы

Система работает в многопрограммном режиме, если в системе находится несколько задач на разной стадии выполнения, и каждая из них может быть прервана другой с последующим восстановлением.

Основной проблемой многопрограммной работы является организация защиты программ от взаимного влияния как на уровне оперативной памяти, так и на разных уровнях внешней памяти. Необходимость организации защиты возникает вследствие того, что различные программы пишутся независимо друг от друга и расчете на единоличное использование ресурсов системы. Это создает опасность взаимного влияния программ друг на друга и должно быть учтено при реализации системы.

Разделение аппаратных и программных ресурсов системы ставит сложные задачи управления перед СУПЕРВИЗОРОМ, которые он решиет (51) благодаря наличию специальных алгоритмов распределения ресурсов системы.

Если количество задач превышает количество процессоров, то возникает проблема определения (планирование) очередности их решения во времени./Планирование во времени — это решение множества задач на ограниченных ресурсах. При организации (планировании) работы параллельной вычислительной ВС и распределении задач между множеством процессоров (ресурсов) планирование осуществляется в пространстве и во времени (рис. 2.4),

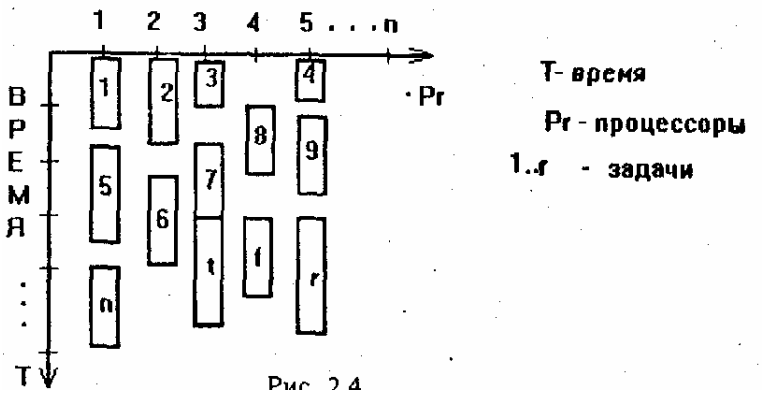


Рис. 2.4

Как только количество процессоров становится больше, чем количество независимо решаемых задач, надобность во временной координате планирования пропадает.

Принято различать пять способов реализации многопрограммного режима работы:

- классическое мультипрограммирование;
- параллельная обработка;
- разделение времени;
- свопинг.

Классическое мультипрограммирование — это режим истинного совмещения, когда параллельно исполняемые задачи занимают различное оборудование.

Режим мультипрограммирования характеризуется тем, что правила перехода от программы к программе устанавливаются из соображений достижения максимального использования ресурсов машины путем более (52) широкого использовании совмещения их действий. Дорогостоящий ЦП все-таки простаивает, когда данные вводятся или выводятся, несмотря на то, что процесс ввода/вывода все время ускоряется. Тогда было предложено другое решение — поместить в память одновременно несколько программ. Память поделена на участки, и каждая программа располагается в своем участке. Теперь, при выполнении ввода/вывода для одной программы, вторая программа может производить вычисления, т.е. процессор переключается на выполнение другой программы (рис. 2.5), Этот подход, при котором несколько программ делят память и по очереди используют ЦП и другие ресурсы, работающие автономно, называется мультипрограммированием.

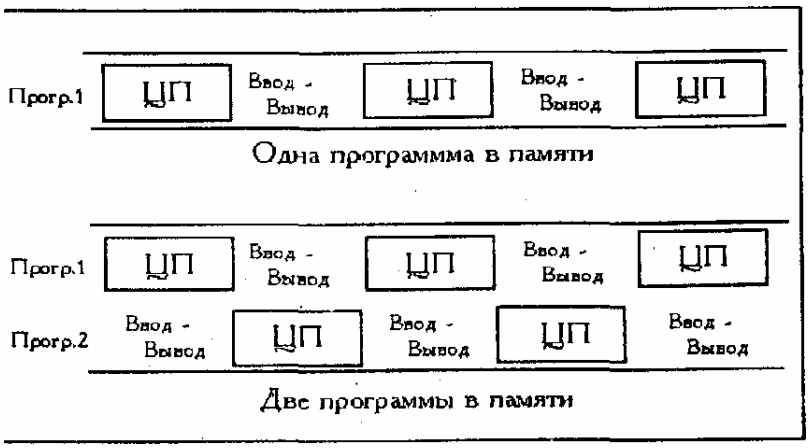


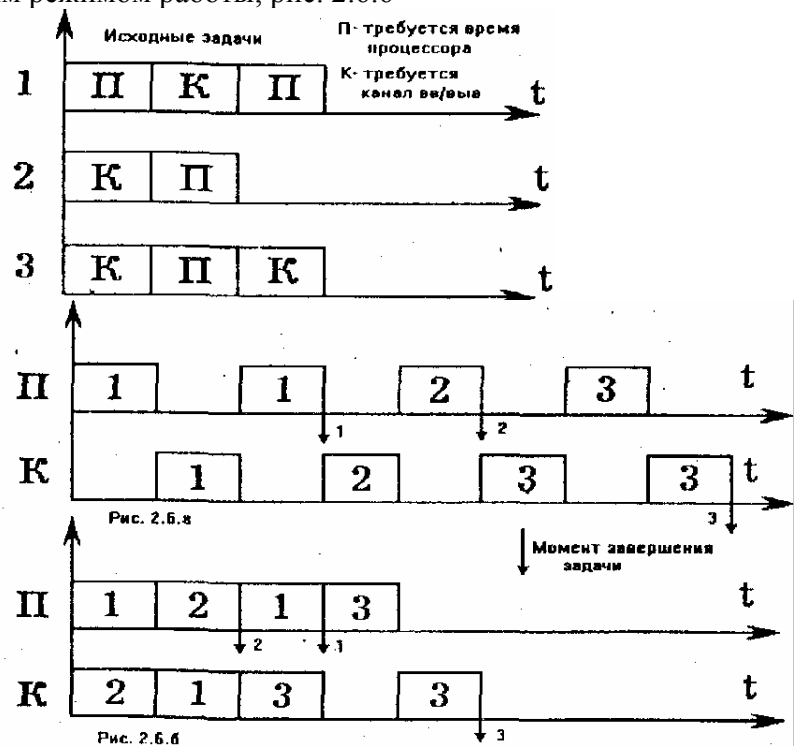
Рис. 2.5. Многозадачный и последовательный подход

При мультипрограммировании улучшение качества обслуживания пользователя непосредственно не предусматривается. Может случится так, что одна из программ, захватив один из ресурсов, например процессор, "заблокирует" выполнение других программ. Кроме того, мультипрограммирование, в чистом виде, не совместимо с непосредственным доступом, так как это один из видов пакетной обработки задач, то есть все пользователи получают результаты одновременно, как и в режиме косвенного доступа. Тем не менее, общее время обработки пакета задач при мультипрограммировании оказывается меньше, чем при последовательной пакетной обработке, за счет лучшего использования оборудования.(53)

Производительность системы, работающей в мультипрограммном режиме. во многом зависит от состава группы программ, исполняемых совместно. Действительно, почти неизбежно возникновение отдельных моментов времен, когда все программы ожидают ввод или вывод данных, а центральное устройство простаивает- Отсюда следует задача целенаправленного формирования пакета программ с целью сокращения непроизводительных простоев оборудования.

На рис. 2.6,6 показан пример выполнения трех задач в мультипрограммном режиме. При этом необходимо учитывать приоритеты задач при выборе задачи, которой необходимо дать предпочтение в случае множественных требований к некоторым ресурсам.

При уменьшении абсолютного времени решения, время решения отдельных заявок может увеличиваться по сравнению с однопрограммным режимом работы, рис. 2.6.6



(54)

Если количество однотипных устройств увеличить, то и количество задач, обрабатываемых одновременно, будет увеличиваться. Для такой системы вводится критерий - степень мультипрограммирования. Степень мультипрограммирования, определяет количество заявок при которых система работает наиболее эффективно, а дальнейшее увеличение количества заявок не приводит к повышению эффективности работы вычислительной системы (рис. 2.6.6).

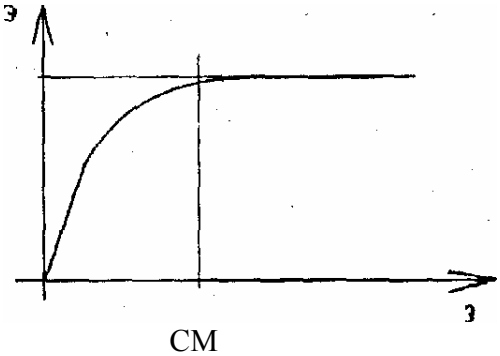


Рис.2.6.б. Э — эффективность, 3 — задачи, СМ — степень мультипрограммирования

Под режимом *параллельной обработки* данных нескольких задач понимается такой многопрограммный режим, в котором переход от одной задачи к другой происходит через достаточно короткие промежутки времени (кванты), сравнимые с тактом машины, чтобы создать у пользователя впечатление одновременности исполнения нескольких программ (режим кажущегося совмещения).

Основной целью данного режима является улучшение обслуживания пользователей, выражающееся в том, что у последних создается впечатление отсутствия очереди, так как решение их задач не прерывается на длительные отрезки времени, Кроме того, если пользователю предоставляются некоторые средства прямого доступа (хотя бы для вывода информации), то это впечатление еще более усиливается выдачей результатов по мере их получения. Параллельная обработка основана на значительном отличии времени реакции пользователя и машины следствии высокой скорости работы последней.

Однако, в общем случае, время отпета при этом режиме не оптимально, так как требуется тонкий механизм учета приоритета задач, учитываемый при определении очередности их обработки. Наибольшее применение получили алгоритмы, основанные на принципе сканирования (55) очереди. Время выполнения пакета задач при параллельной обработке значительно превышает время их выполнения при единоличном использовании машины. Тем не менее, параллельная обработка задач, при которой пользователь может получать результаты, не ожидая конца обслуживания всего пакета, уменьшает время ожидания ответа по сравнению с режимом последовательной обработки.

В большинстве случаев параллельная обработка сочетается с мультипрограммированием, что обеспечивает наличие двух типов прерываний, вызывающих переход к другой программе: естественное прерывание во время ожидания ввода/вывода (мультипрограммирование); вынужденное прерывание в конце отрезка времени, отводимого каждой программе (параллельная обработка).

Разделение времени — наиболее развитая форма многопрограммной работы, совмещающая мультипрограммирование с непосредственным доступом и обеспечением доступа некоторых привилегированных пользователей к ресурсам системы. Время ответа в такой системе близко к тому, которое было при единоличном использовании машины.

Как и при параллельной работе, задачи выполняются поочередно в течение некоторого отрезка времени (кванта), по завершении которого происходит вынужденное прерывание. Длительность этих отрезков времени (квантов) не является, как правило, фиксированной, а изменяется в зависимости от рассматриваемой задачи, а также от конкретных условий эксплуатации в момент передачи управления от одного пользователя другому. Выбор пользователя, задаче которого будет передано управление, и выделение этой задаче кванта времени осуществляет управляющая программа, являющаяся составной частью программ — диспетчеров, предназначенных для работы с разделением времени. Диспетчер функционирует в соответствии с выбранной для данной системы дисциплиной обслуживания заявок, которые будут рассмотрены ниже.

Существует не менее интересный способ реализации многопрограммного режима работы на однопроцессорной машине — "свопинг". Он используется при недостаточной памяти, для хранения всех параллельно выполняемых программ.

Свопинг характеризуется тем, что после определенного времени (кванта) программы пользователей, находящиеся в оперативной памяти, переписываются на внешний носитель информации (диск) в специально выделенные для этого своп-файлы т.е. сохраняется состояние этих задач. Следующая совокупность, параллельно выполняемых программ занимает освободившееся место в оперативной памяти, и система исполняет их в многопрограммном режиме до завершения следующего кванта,(56)

Формы взаимодействия человека с машиной

Учитывая вышесказанное, можно выделить следующие формы вза-НМОДСЙСТУИЯ пользователя с машиной.

1) Непосредственный доступ (НД).
2) Косвенный доступ (Косв.Д).
3) Коллективный доступ (КД).

Непосредственный доступ (НД) подразумевает, что все ресурсы вычислительной системы находятся в полном распоряжении пользователя.

Косвенный доступ (Косв.Д) — режим работы пользователя с вычислительной системой с помощью управляющей программы (косвенно), для которой пользователь должен подготовить информацию, определяющую ее действия для выполнения его работы.

Коллективный доступ (КД) подразумевает работу многих пользователей с ресурсами системы, те, разделение ресурсов (оборудования, времени, программ, данных) между многими. В этом режиме, как правило, пользователь общается с ЭВМ посредством терминального устройства, имеющего ограниченные возможности по вводу и выводу информации и отсутствие вычислительных.

В настоящее время принципы коллективного доступа значительно усилились за счет использования интеллектуальных терминалов — персональных ЭВМ. Сочетание вычислительных возможностей персональной ЭВМ и использование, в случае необходимости, всей вычислительной мощности вычислительной среды, к которой этот терминал подключен, определяет практически неограниченные возможности выполнения задач пользователя. Такую форму взаимодействия (НД1) можно определить как непосредственный доступ на новом качественном уровне.(рис. 2.7) Несомненно эта форма взаимодействия человека с машиной (или лучше с вычислительной средой) требует совершенно новых принципов организации вычислительной системы и ее функционирования.

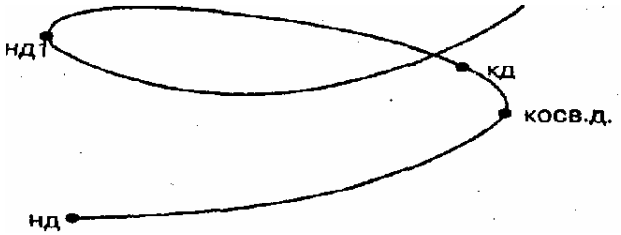


Рис. 2-7. Эволюционное развитие формы взаимодействия человек — машина(57)

Для достижения большей производительности ВС в систему включают несколько независимых процессоров, взаимодействующих через оперативную память или специальный канал связи. Подобная организация ВС привела к мультипроцессорным вычислительным системам, способным параллельно выполнять несколько независимых программ или несколько независимых ветвей одной программы. Высокие требования к надежности и живучести вычислительной системы предопределили наличие в ней как минимум двух процессоров н возможность двухмаршрутной связи между модулями системы Свойства мультипроцессорных вычислительных систем отражают две основные тенденции современного развития вычислительной техники:

- Модульный принцип построения технического обеспечения, при котором каждое устройство выполняется в виде независимого модуля, что позволяет: во первых, наращивать структуру и, во-вторых, достигается возможность сделать систему менее уязвимой к отказам в силу

взаимозаменяемости однотипных модулей.

- Распараллеливание программ (выделение параллельных участков) и параллельное выполнение независимых ветвей одной программы или выполнение полностью независимых программ, что позволяет уменьшить общее время реализации задач и повысить эффективность работы оборудования.

Второе свойство отличает мультипроцессорные системы от мультипрограммных, в памяти которых хранится несколько программ, совмещаются работы внешних и центральных устройств и выполнение отдельных операций центральных устройств. Основная цель мультипрограммирования — наиболее полное использование оборудования — лучше всего достигается при непрерывном функционировании в системе большого количества программ, когда несмотря на прерывания, и, следовательно, некоторое увеличение времени выполнения одной программы, время решения многих задач уменьшается по сравнению с временем их решения на машине без совмещения операций.

Многопроцессорная архитектура включает в себя два и более ЦП, совместно использующих общую память и периферийные устройства (рис. 2.8), располагая большими возможностями в увеличении производительности системы, связанными с одновременным исполнением процессов на разных ЦП. Каждый ЦП функционирует независимо от других, но все они работают с одним и тем же ядром операционной системы. Поведение процессов в такой системе ничем не отличается от поведения в однопроцессорной системе — с сохранением семантики обращения к каждой системной функции — но при этом они могут открыто перемещаться с одного процессора на другой. Хотя, к сожалению, это не приводит к (58) снижению затрат процессорного времени, связанного с выполнением процесса.

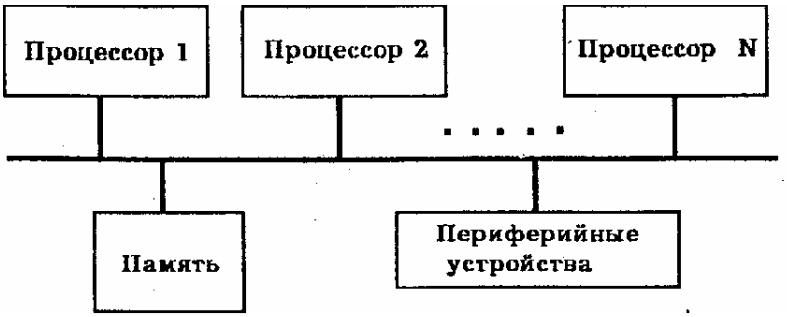


Рис.2.8. Многопроцессорная конфигурация

Отдельные многопроцессорные системы называются системами с присоединенными процессорами, поскольку в них периферийные устройства доступны не для всех процессоров.

Параллельная работа нескольких процессоров в режиме ядра при выполнению различных процессов создает ряд проблем, связанных с сохранением целостности данных и решаемых благодаря использованию соответствующих механизмов защиты

Если вопрос об опасности возникновения нарушения целостности оставить открытым, как бы редко подобные нарушения ни случались, ядро утратит свою неуязвимость и его поведение станет непредсказуемым. Избежать этого можно тремя способами:

- Исполнять все критические операции на одном процессоре, опираясь на стандартные методы сохранения целостности данных в однопроцессорной системе;
- Регламентировать доступ к критическим участкам программы, используя элементы блокирования ресурсов;
- Устранить конкуренцию за использование структур данных путем соответствующей переделки алгоритмов,

Главный процессор несет ответственность за обработку всех обращений к операционной системе и всех прерываний. Подчиненные процессоры ведают выполнением процессов в режиме задачи и информируют главный процессор о всех производимых обращениях к системным функциям.(59)

Мультипроцессорная система, в отличии от мультипрограммной, позволяет уменьшить время решения каждой отдельной задачи за счет одновременного выполнения независимых участков ее программы и достигнуть высокого коэффициента использования оборудования при непрерывной обработке большого количества программ.

Распределенные операционные системы представляют собой совокупность вычислительных средств и периферийных устройств, объединенных между собой с обеспечением возможности взаимного обмена информацией.

Свойства распределенных операционных систем:

- Отсутствие обшей памяти приводит к тому, что нельзя определить общее состояние системы с помощью множества совместных переменных, а невозможность совместного обращения к памяти и различия в задержке передач сообщений приводят к тому, что при определении состояния какого-либо элемента системы из двух различных точек можно получить разные результаты, в зависимости от порядка предшествующих событий;
- Распределенная система распределяет выполняемые работы в узлах системы, исходя из соображений повышения пропускной способности всей системы;
- Распределенные системы имеют высокий уровень организации параллельных вычислений-

Два фактора увеличивают вероятность отказов отдельных элементов (по сравнению с централизованными системами):

1. Большое число элементов системы.
2. Надежность систем связи обычно меньше, чем надежность информационных систем.

Однако надежность всей системы в целом в распределенных операционных системах весьма высока за счет специфических свойств, позволяющих исключить или ослабить эффект отказа отдельных элементов системы (исправление вышедших из строя элементов, переконфигурация системы и т.д.).

Архитектура распределенной системы представлена на рис. 2.9. Каждый компьютер, показанный на рисунке, является автономным модулем (вычислительным узлом), состоящим из ЦП, памяти и периферийных устройств. Соответствие модели не нарушается даже несмотря на то, что компьютер располагает только локальной файловой системой. В случае необходимости, он должен иметь периферийные устройства для связи с другими машинами, а все необходимые ему файлы могут располагаться и на ином компьютере. Физическая память, доступная каждой машине, не зависит от процессов, выполняемых на других машинах. Этой особенностью распределенные системы отличаются от сильносвязанных (60) многопроцессорных систем. Соответственно, и ядро системы на каждой машине функционирует независимо от внешних условий эксплуатации распределенной среды.

Распределенные системы традиционно делятся на следующие категории.

- *Периферийные системы,* представляющие собой группы машин, отличающихся ярко выраженной общностью и связанных с одной (обычно более крупной) машиной. Периферийные процессоры делят свою нагрузку с центральным процессором и переадресовывают ему все обращения к операционной системе. Цель периферийной системы состоит в увеличении обшей производительности сети и в предоставлении возможности выделения процессора одному процессу в операционной системе. Система запускается как отдельный модуль; в отличие от других моделей распределенных систем, периферийные системы не обладают реальной автономией, за исключением случаев, связанных с диспетчеризацией процессов и распределением локальной памяти.

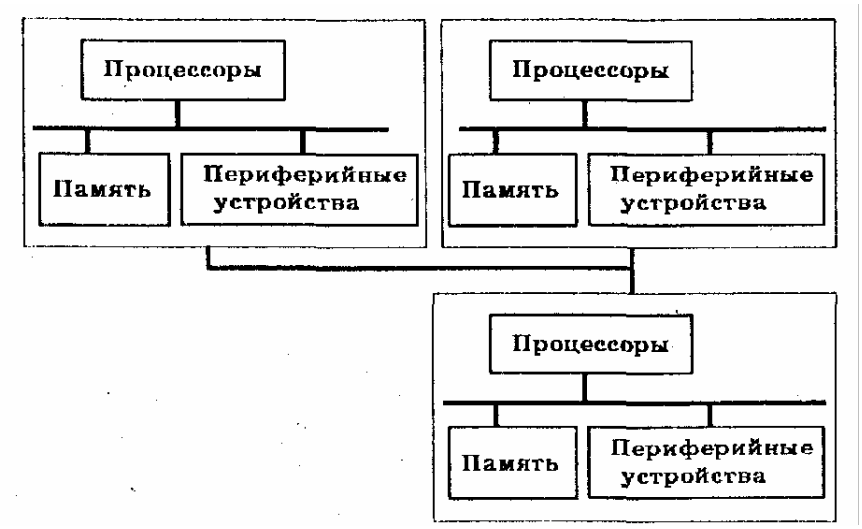


Рис. 2.9 Модель системы с распределенной архитектурой

- *Распределен системы* типа "Newcastle",' позволяющие осуществлять дистанционную связь но именам удаленных файлов в (61) библиотеке. Удаленные файлы имеют спецификацию (составное имя), которая в указании пути поиска содержит специальные символы или дополнительную компоненту имени, предшествующую корню файловой системы. Реализация этого метода не предполагает внесения изменений в ядро системы, вследствие этого он более прост, чем другие методы, но менее гибок.
- *Абсолютно "прозрачные" распределенные системы,* в которых для обращения к файлам, расположенным на других машинах, достаточно указания их стандартных составных имен; распознавание этих файлов как удаленных входит в обязанности ядра. Маршруты поиска файлов, указанные в их составных именах, пересекают машинные границы в точках монтирования, сколько бы таких точек ни было сформировано при монтировании файловых систем на дисках. Виртуальная операционная система характеризуется обеспечением многопрограммного режима работы, когда каждый пользователь может работать в собственной операционной среде.

2.5. ПОНЯТИЕ ГЕНЕРАЦИИ. ИНСТАЛЛЯЦИИ. ИНИЦИАЛИЗАЦИИ

При покупке вычислительной системы приобретается и программное обеспечение для выполнения тех функций, ради которых эта ВС и покупалась, Как правило, и это ПО включены все необходимые программы для полной поддержки всех соответствующих частей ВС. Таким образом, покупается исходный вариант операционной системы или *дистрибутив.* В него входят различные варианты драйверов для различных аппаратных компонент, а также программы для обеспечения различных режимов работы ВС. Это драйверы CD-ROMa, сетевых Ethernet карт, SCSI-устройств, Sound Blastera, принтера и т.д. Причем в состав дистрибутива могут входить варианты драйверов для поддержки различных локальных шин (VESA, PCI) и других аппаратных платформ. Кроме этого, дистрибутив имеет все необходимые программы, поддерживающие работу вычислительной системы в различных режимах работы, например, однопрограммном или многопрограммном, многопроцессорном, сетевом.

Для нормального функционирования вычислительной системы в реальных условиях эксплуатации совокупность программ всего исходного ПО не нужна и из всего их множества выбирается некое подмножество программ, обеспечивающих работу вычислительной установки в установленных пользователем режимах и на имеющемся оборудовании. Процесс отбора из дистрибутива тех программных модулей, которые будут использоваться называется *генерацией* операционной системы. Процесс генерации выполняется либо с использованием специального "языка (62) генерации", либо с помощью организации диалога пользователя с системой в результате которого определяются желания и требования пользователя к функционированию системы, а система в соответствии с этим определяет совокупность необходимых программ, поддерживающих эти требования.

Место, где находится резиденция системы, называется "резидентным" гомом, а сама система "резиденцией". Из всех программных модулей в резидентном томе часть используется по мере необходимости, а некоторые нз них обеспечивают базовое функционирование ВС и составляют *резидентные* программы, находящиеся в системной области оперативной памяти. Совокупность программ, обеспечивающих функционирование ВС и находящихся в системной области оперативной памяти, составляет ядро супервизора.

Некоторые программы, связанные с выполнением функций ОС, но не находящиеся постоянно в оперативной памяти называются *транзитными.* Эти программы вызываются в память по мере необходимости. Кроме этого, пользователь может по своему желанию определить некоторые программы как резидентные. Все остальные программы являются транзитами, вызываются динамически и могут затирать друг друга.

При функционировании вычислительной системы пользователь может менять свои представления о ее функционировании. Кроме этого может меняться и состав оборудования. Процесс локальной настройки ОС по желанию пользователя называется *инсталляцией.* Признаком, по которому можно отличить инсталляцию от генерации является то, что, если при настройке операционной системы не используется дистрибутив – это инсталляция, если требуется дистрибутив — генерация.

При загрузке ОС необходимо выполнить связывание, размещение всех частей, входящих У ядро супервизора, т.е. размещение резидентных программ на своих местах, формирование специальных

системных структур данных в области данных операционной системы, формирование постоянной области, активизацию процессов для начала работы операционной системы. Этот процесс называется *инициализацией* операционной системы.

26 КОНЦЕПЦИЯ РЕСУРСОВ В СОВРЕМЕННОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЕ

Под ресурсом абстрактно можно понимать "нечто" (реально существующее), которое может понадобиться для выполнения программы (задачи, процесса). Все ресурсы можно разделить условно на две категории:

1. Физические устройства, входящие в состав современной ЭВМ.(63)
2. Внутренние ресурсы пользовательских программ (данные, подпрограммы и т. д.).

В качестве физических устройств можем рассматривать следующие:

- оперативная память (или ОЗУ), где хранятся данные, программы, различные системные объекты управления, векторы прерываний, информация о внешних устройствах, таблицы оверлеев, системные буферы, блоки управления активизированных процессов (PCB) и т.д.
- процессор(ы), который выполняет работу, определяемую операционной системой (выполняет обработку данных);
- клавиатура как устройство ввода;
- принтер как устройство печати (вывода);
- терминал как устройство вывода информации;
- магнитные, жесткие или гибкие диски как устройства ввода/вывода.

Чаше всего при описании ресурсов рассматривают процессор и оперативную память (ОП). ОП выделяется пользователю для записи данных, программ, размещения блоков управления процессами и так далее.

Если ОП достаточно велика для того, чтобы разместить все необходимые программы, данные и вспомогательные таблицы, то особых затруднений в размещении и использовании пространства памяти нет.

Иначе обстоит дело с процессором. Это самое динамичное, с точки зрения управления и планирования его работой, устройство.

Под выделением процессора понимается выделение его времени для обработки различных задач, находящихся в системе.

Наибольшую трудность для ОС может представлять планирование процессов для выполнения на процессоре. Для этого существует множество дисциплин управления очередями, но обязательным является использование приоритетов, которое позволяет отсортировать имеющиеся в системе и готовые к выполнению процессы.

Наиболее приоритетный процесс и получает в свое распоряжение время процессора (квант) для выполнения. Разработчики ОС стремятся достигнуть максимально возможной загрузки процессора (наиболее эффективной).

К внутренним ресурсам системы можно также отнести различные данные (константы, переменные), подпрограммы или даже целые программы (библиотеки программ, используемые пользовательскими программами в процессе выполнения).

В многопользовательских или многопроцессорных системах к некоторым программам часто обращаются или они могут использоваться несколькими процессами одновременно. Поэтому ДЛЯ повышения эффективности работы системы стремятся выделить из этих программ ядро – наиболее используемые данные и процедуры программы, — для использования этими процессами.(64)

Некоторый ресурс может потребоваться сразу нескольким процессам, однако, в любой момент времени он может быть выделен только одному из них. В таком случае остальные процессы выстраиваются в очередь (отсортированную по приоритету) на выделение ресурса. Когда процесс освободит ресурс, он может быть выделен первому процессу в очереди ожидающих, а если такового нет, то ресурс считается свободным.

Такая задача согласования работы нескольких процессов при использовании общею ресурса называется *задачей взаимного исключения,* т.е. первый процесс в очереди ожидания к общему ресурсу получает его, а остальным доступ к ресурсу запрещен.

2.7. ПРОЦЕСС — ОСНОВА ФУНКЦИОНИРОВАНИЯ ВС

Процесс— это теоретическое понятие, на основании которого можно описать то, что происходит в системе при выполнении некоторых действий (программы).

Для пользователей знать, что происходит в системе при выполнении программы, не обязательно — наглядно показывает результат. Однако, для разработчиков систем необходимо знать, что происходит при выполнении той или иной операции, т.е. изменение состояния машины на физическом уровне. Понятие процесса, его развитие во времени и в пространстве в распределенных системах обработки информации помогают создать некоторую условную модель выполнения программы.

Состояние машины определяется состоянием процессора (содержимым адресуемых и внутренних регистров) и состоянием памяти (содержимым ячеек памяти), Это состояние меняется при выполнении процессором некоторых операции, выполнение операции — это некоторое действие, результатом которого является переход за конечное время из начального состояния машины (до выполнения операции) в конечное (после ее выполнения). В глобальном масштабе такой операцией может быть и вся программа. Однако, на таком уровне мы не можем цвести понятие процесса. Тогда рассмотрим каждую такую операцию как неделимую, т.е. выполняемую процессором за один шаг (единицу времени, машинный такт).

С этой точки зрения программу можно рассматривать как последовательность элементарных (неделимых) операций. Будем отслеживать выполнение программы в течение времени T (от начала до конца выполнения). За это время процессор успеет выполнить N операций (N сколь угодно большое), которые будем идентифицировать по номеру (1,…,N).

В таких элементарных операциях можно выделить два момента: начало операции - H и конец операции - K. В эти моменты, времена которых обозначили tH и tK, будем отмечать состояния (65) машины. Эти два момента считаются *событиями.* Таким образом, события позволяют отмечать изменения состояния машины.

При выполнении программы имеем последовательность операций 1,2,..,N, причем $tH(1) < tK(1) < tH(2) < ... < tK(N)$ (хотя конец предыдущей операции и начало следующей — это в принципе одно и то же). Такая последовательность и называется *последовательным процессом* (или просто *процессом).* События H(l), K(1), H(2),..., K(N) образуют *временной след* (трассу или историю) процесса.

Таким образом, можно выделить три наиболее употребляемых определения процесса.

Процесс :

- это любая выполняемая работа в системе;
- это динамический объект системы, которому она выделяет ресурсы;
- это траектория процессора в адресном пространстве ВС.

2.7. Классификация процессов

По способу создания процессы делятся на:

- Системные процессы. Они создаются при загрузке системы, имеют оверлейную или динамически последовательную структуру программ.
- Проблемные (пользовательские) процессы. Создаются при активизации работы (задачи пользователя) операционной системой. Под активизацией процесса здесь понимается начало выполнения программы, подчиненной этому процессу, т.е. начало процесса. Пользовательский процесс может быть активизирован только другим процессом, для этого и существуют системные процессы, активизируемые ОС при запуске.

По способу существования процессы делятся на:

- Последовательные. Это процессы, которые выполняются друг за другим. т.е. когда закончится выполнение первого процесса, начинается второй и т.д.
- Параллельные. Процессы, которые могут выполняться одновременно. *Параллельные процессы* могут быть:

а) Независимые, т.е. выполняют разные операции и не имеют общих частей.

б) Асинхронные — это процессы, которые должны синхронизироваться и взаимодействовать друг с другом. Взаимодействие — это передача данных одним процессом другому. Также процессы могут иметь общие части — процедуры, данные, требуемые (но еще не выделенные) ресурсы. При этом возникает

задача взаимного исключения. (66)

При инициализации ядра и системы активизируются процессы, подчиненные ОС — системные процессы, обеспечивающие выполнение основных функции ОС.

Особое значение для функционирования вычислительной системы имеет процесс, связанный с обработкой прерываний. Он является первым процессом, который активизируется при загрузке и инициализации ядра.

Должен быть активизирован процесс администратора памяти (содержащийся в системном файле msdos.sys) — это процесс, который управляет эффективным распределением памяти для размещения различных программ. Также необходима активизация процесса для работы с внешними устройствами и управления данными (управления файловой системой).

Выделяется отдельно процесс, подчиненный часам, который активизируется на аппаратном уровне.

2.7.2. Состояния процессов

Во время своего существования процесс может находиться в четырех состояниях: *подготовленном, готовом, активном и заблокированном* (рис.2.10). В каждый конкретный момент времени процесс находится в одном из этих состояний, которое фиксируется в блоке управления процессом (PCB).

Процесс находится в подготовленном состоянии, если он находится в системе (как правило на внешнем носителе информации), но ему не выделены ресурсы системы.

Процесс находится в готовом состоянии или активизирован, если ему уже выделены ресурсы (программа, необходимая для выполнения, находится в оперативной памяти), однако ему не выделено время процессора для выполнения (процессор занят другим процессом).

Процесс, которому выделяется время процессора, переводиться в активное состояние или состояние выполнения.

Во время выполнения процесса ему могут понадобиться дополнительные ресурсы, но для их получения необходимо некоторое время. Т.е. процесс должен ожидать наступления некоторого события или окончания выполнения конкретной операции (например, ввода/вывода для получения/выдачи данных). Такой процесс переводится в заблокированное состояние.

Процессы при поступлении в систему находятся в подготовленном состоянии и накапливаются во входных очередях заданий. Следует различить процессы, поступающие в систему и требующие безусловного выполнения в соответствии с одной из дисциплин обслуживания, и (67) процессы, находящиеся в подготовленном состоянии после загрузки операционной системы. Процессы первого вида, как правило, принадлежат пользователям, а второго вида — ОС. Такие процессы находятся в подготовленном состоянии до тех пор, пока для выполнения функций системы они не будут активизированы. Пользовательские процессы активизируются или делается попытка их активизации при наличии в системе ресурсов. Попытка активизации процесса производится системой при наличии в системе оперативной памяти, достаточной для размещении программы, подчиненной процессу.

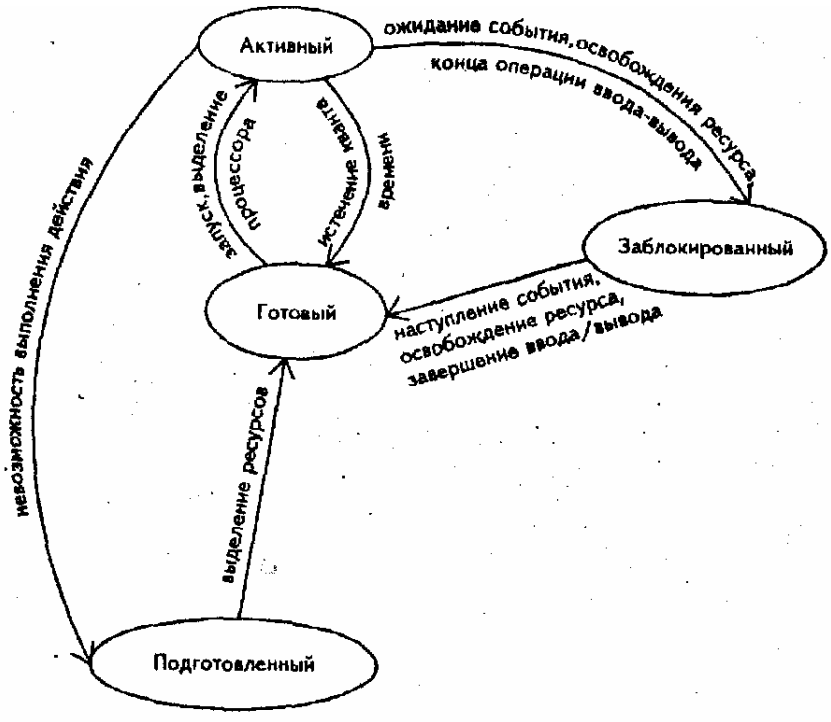


Рис. 2.10. Переход процесса из состояния в состояние

Поэтому в системе может находиться одновременно довольно много подготовленных и готовых процессов, для которых организуются очереди (одна для готовых и одна для подготовленных процессов). Дисциплины (68) управления ими могут быть различными и выбор дисциплины обслуживания определяется требованиями, предъявляемыми к системе планирования.

Если система определила необходимость активизации процесса и выделяет нужные ему ресурсы, кроме времени процессора, то она переводит его в готовое состояние. При этом система определяет (если может) имеются ли в системе ресурсы, необходимые для выполнения данного процесса. Процесс, который переводится в готовое состояние, должен иметь больший приоритет, чем процессы, требующие таких же, как и он, типов ресурсов *(т.е.* он находится в начале очереди подготовленных процессов). При этом сортировку по приоритетам процессов в очереди и планирование перехода в готовое состояние (следя за освобождением ресурсов) должен осуществлять планировщик данной очереди.

Если процессор освободился, то первый (наиболее приоритетный) процесс из очереди готовы-процессов получает время процессора и переходит в активное состояние Выделение времени процессора процессу осуществляет диспетчер.

Если активный процесс не выполнился за выделенный ему квант времени, то он переходит снова в готовое состояние.

Процесс, который требует дополнительных ресурсов, кроме времени процессора, либо выполнения некоторой операции другим процессом, переводится в заблокированное состояние и ожидает наступления события, которое он потребовал. Как только произошло ожидаемое событие, процесс переводится в готовое состояние.

Если процесс требует действия или ресурса, которых в данный момент операционная система не может выполнить, он переводится в подготовленное состояние и считается приостановленным.

Задача ОС заключается в слежении за состоянием всех процессов, находящихся внутри системы, особенно в активизированном состоянии, поскольку в активном состоянии процесс является монополистом по отношению к процессору. Поэтому ему и выделяется квант времени, чтобы процесс не блокировал процессор или не занимал его на длительное время, иначе могут возникнуть тупиковые ситуации. В некоторых ОС процесс. требующий наименьшего времени обслуживания на процессоре, имеет наибольший приоритет. Тогда быстрые заявки быстрее покинут систему, а процессы, требующие на обслуживание довольно большого кванта времени при большой загрузке системы вообще могут не обслужиться. С этой точки зрения наиболее приоритетными оказываются системные процессы, так как в ОС все заложено для их быстрого выполнения.

Приоритеты программ обработки прерываний зависят от употребляемости соответствующих ресурсов. Наибольший приоритет имеет (69) программа обработки прерывания по таймеру (вызывается 18,2 раза в секунду) и наименьший — прерывание по вводу/выводу. Пользовательские процессы имеют различные приоритеты в зависимости от порождающих их процессов и от времени выполнения на процессоре. Также необходимо следить за очередью блокированных процессов, с тем, чтобы среди них не было "бесконечно" ожидающих процессов. Вполне возможно, что событие, которое они ожидают, вообще может не произойти - тогда их надо вывести из системы. Также может оказаться, что блокированный процесс имеет слишком низкий приоритет и он может бесконечно долго находиться в режиме бесконечного откладывания в очереди блокированных процессов —тогда необходима система динамического, адаптивного изменения приоритетов.

В общем случае, система должна следить за процессами в очередях готовых и заблокированных процессов, чтобы не было бесконечно ожидающих процессов или процессов, монопольно удерживающих выделенные им ресурсы. Из сказанного следует, что эффективность системы управления процессами в значительной степени влияет на характерно гики ВС, связанные с пропускной способностью. Плохая организация управления процессами может привести к тому, что некоторые процессы могут так и остаться в подготовленном состоянии.

2.7.3. Блок управления процессом (PCS —process control block)

Выполнение функций ОС, связанных с управлением процессами, осуществляется с помощью блока управления процессом (PCB). Вход в процесс (фиксация системен процесса) — это создание его блока управления (PCB), а выход из процесса — это его уничтожение, т.е. уничтожение его блока управления.

Таким образом для каждого активизированного процесса система создает PCB, в котором в сжатом виде содержится информация о процессе, используемая при управлении. PCB — это системная структура данных, содержащая определенные сведения о процессе и имеющая следующие поля.

1. Уникальный и идентификатор процесса (имя).
2. Текущее состояние процесса.
3. Приоритет процесса.
4. Указатели участка памяти выделенного программе, подчиненной данному процессу.
5. Указатели выделенных ему ресурсов.
6. Область сохранения регистров.
7. Права промесса (список разрешенных операций).(70)
8. Связи зависимости ц иерархии процессов (список дочерних процессов, имя родительского процесса).
9. Пусковой адрес программы, подчиненной данному процессу.

Когда ОС переключает процессор с процесса на процесс, она использует области сохранения регистров в PCB для запоминания информации, необходимой для рестарта (повторного запуска) каждого процесса с точки прерывания, когда он в следующий раз получит в свое распоряжение процессор. Количество процессов в системе ограничено и определяется самой системой, пользователем но время генерации ОС или при загрузке. Неудачное определение количества одновременно исполняемых программ может привести к снижению полезной эффективности работы системы, т.к. переключение процессов требует выполнения дополнительных операций по сохранению н восстановлению состояния процессов. Блоки управления системных процессов создаются при загрузке системы. Это необходимо, чтобы система выполнила свои функции достаточно быстро, и время реакции ОС было минимальным. Однако, количество блоков управления системными процессами меньше, чем количество самих системных процессов. Это связано с тем, что структура ОС имеет либо оверлейную, либо динамически — последовательную структуру иерархического типа, и нет необходимости создавать для программ, которые никогда не будут находиться одновременно в оперативной памяти, отдельные PCB. При такой организации легко учитывать приоритеты системных процессов, выстроив их по приоритетам заранее при инициализации системы. Блоки управления проблемными (пользовательскими) процессами создаются в процессе активизации процессов динамически. Все PCB находятся в выделенной системной области памяти.

В каждом PCB есть поле состояния процесса. Все блоки управления системными процессами располагаются в порядке убывания приоритетов и находятся в системной области памяти. Если приоритеты системных блоков можно определить заранее, то для проблемных процессов необходима

таблица приоритетов проблемных программ. Каждый блок PCB имеет стандартную структуру, фиксированный размер, точку входа, содержит указанную выше информацию и дополнительную информацию для синхронизации процессов. Для синхронизации в PCB имеются четыре поля:

1-2. Поля для организации цепочки связи.

3-4. Поля для организации цепочки ожидания.

В цепочке связи указывается адрес PCB вызываемого (поле 1) и вызывающего (поле 2) процесса.
В цепочке ожидания, в поле 3 указывается адрес PCB вызываемого процесса, если вызываемый процесс запит. В поле 4 занятого процесса находится число процессов, которые ожидают (71) данный.

Если процесс А пытается вызвать процесс В, а у процесса В в PCB занята цепочка связей, то есть он является вызываемым по отношению к другим процессам, тогда адрес процесса В записывается в цепочке ожидания PCB процесса А, а в поле счетчика ожидания PCB процесса В добавляется 1. Как только процесс В завершает выполнение своих функций, он передает управление вызывающему процессу следующим образом; В проверяет состояние своего счетчика ожидания, и, если счетчик больше 0, то среди PCB других процессов ищется первый (по приоритету или другим признакам) процесс, в поле 3 PCB которого стоит имя ожидаемого процесса, в данном случае В, тогда управление передастся этому процессу.

Пример: Имеем связь процессов X и B, и процесс A вызвал процесс B (рис.2.11).

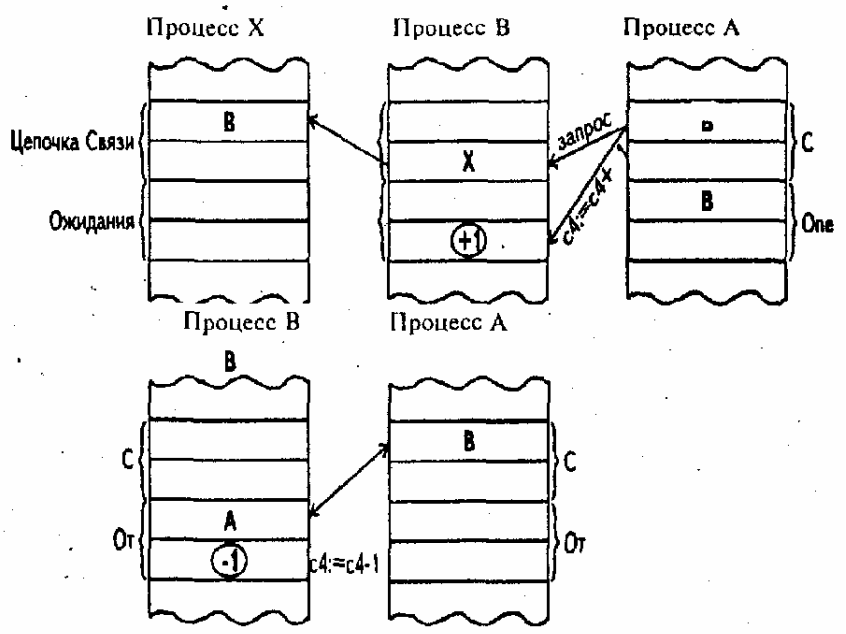


Рис. 2.11. Состояние после выполнения процесса X(72)

2.7.4. Операции над процессами

Над процессами можно производить следующие операции; создание; уничтожение; возобновление; изменение приоритета; блокирование; пробуждение; запуск (выбор).

1. Создание пронесся включает в себя:

а) присвоение имени процессу.

б) включение этого имени в список имен процессов, известных системе.

в) определение начального приоритета.

г) формирование блока управления процессом.

д) выделение начальных ресурсов.

2. Уничтожение процесса означает его удаление из системы. Ресурсы, выделенные этому процессу, возвращаются системе, имя в системных списках стирается и блок управления процессом освобождается.
3. Изменение приоритета означает просто модификацию значения приоритета в блоке управления данным процессом. В основном увеличивают приоритет у процессов, когда предполагают, что они будут находиться в состоянии бесконечного откладывания или уменьшают приоритет процессу, надолго захватившему процессор.

4. Запуск (выбор) процесса, осуществляемый планировщиком, который выделяет процессу нремя процессора.
5. Приостановленный процесс не может продолжить свое выполнение до тех пор, пока его не активизирует любой другой процесс (кратковременное исключение определенных процессов в периоды пиковой нагрузки). В случае длительной приостановки процесса, его ресурсы должны быгь освобождены. Решение об освобождении ресурса зависит от его природы. Основная память должна освобождаться немедленно, а вот принтер может и не быть освобожден- Приостановку процесса обычно производят в следующих случаях:

а)если система работает ненадежно и может отказать, то текущие процессы надо приостановить, исправить ошибку и затем активизировать приостановленные процессы;

б) если промежуточные результаты работы процесса вызвали сомнение, то нужно его приостановить, найти ошибку и запустить либо сначала, либо с контрольной точки;

в) если ВС перегружена, что вызвало снижение ее эффективности;

г) если для продолжения нормального выполнения процессу необходимы ресурсы, которые ВС не может ему предоставить.

Инициатором приостановки может быть либо сам процесс, либо другой процесс. В однопроцессорной ПС при однопрограммном режиме работы выполняющийся процесс может приостановить сам себя (73) или быть принудительно остановиться с пульта управления без возможности восстановления, т.к. ни один другой процесс не может выполняться одновременно с ним, В мультипроцессорной системе или при многопрограммном режиме работы любой выполняющийся процесс может быть приостановлен другим процессом.

6. Возобновление (или активизация) процесса — операция подготовки процесса к повторному запуску выполняемая операционной системой, с той точки, в которой он был приостановлен.

2.7.5. Иерархия процессов

Процесс может породить новый процесс. При этом первый, порождающий процесс, называется родительским, а второй, порожденный процесс, - дочерним. Для создания некоторого процесса необходим только один родительский процесс (рис. 2.12).

Таким образом, в системе создается иерархическая структура процессов, На самом верху иерархии (рис. 2.12) находятся системные процессы (процесс А на рисунке), которые в свою очередь порождают дочерние процессы и т.д. Получаем некоторое дерево процессов, которое может расти вниз и иметь столько ярусов, сколько позволяет система. Причем у дочерних процессов может быть только один родительский, но у родительского может быть несколько дочерних.

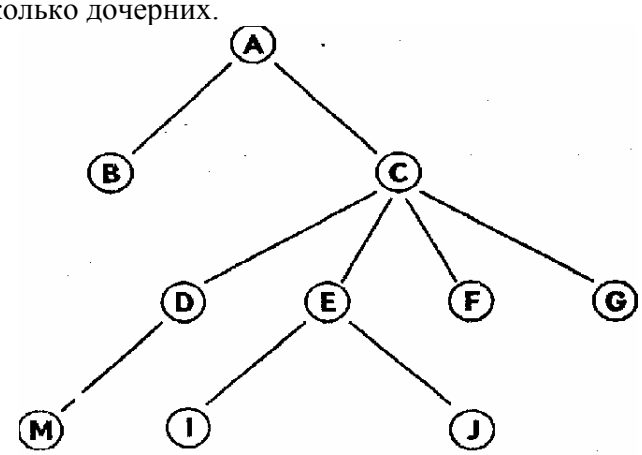


Рис 2.12. Иерархия процессов(74)

Уничтожение процессов на нижнем ярусе иерархии не вызывает затруднений, т.к. они не имеют дочерних процессов. При уничтожении родительского процесса в некоторых системах удаляются и дочерние процессы, а в некоторых дочерние процессы начинают существовать независимо от родительского, уничтоженного. При создании процесса в системе UNIX текущий процесс разветвляется на два независимых параллельных процесса: родительский и дочерний. Они не имеют общей первичной

памяти, но могут совместно использовать все открытые файлы. Для дочерних процессов создаются копии всех сегментов данных, в которые разрешена запись. Создается процесс примитивом fork, значение которого позволяет затем узнать какой из процессов является дочерним, а какой — родительским. Родительский процесс может быть уничтожен раньше дочерних в двух случаях:

1. Его выполнение не зависит от результатов, формируемых в дочерних процессах, и он имеет все ресурсы, которые необходимы для его завершения. Тогда родительский процесс завершается независимо от завершения дочерних.
2. 2. ОС считает, что он создаст тупиковую ситуацию и удаляет его из системы. Однако, обычно родительский процесс ожидает завершения дочерних, получает от них результаты и завершается сам.

*2.7.*6. Прерывание. Переключение контекста процесса

Прерывание — это нарушение последовательности выполнения действий (команд), т.е. после текущего действия (команды) выполняется не следующее (команда), а некоторое другое действие.

При появлении сигнала прерывания управление передается ОС, которая запоминает состояние прерванного процесса *в* области сохранения регистров PCB. Далее ОС анализирует тип прерывания и передает управление соответствующей программе обработки этого прерывания. Инициатором прерывания может быть выполняющийся процесс или оно может быть вызвано некоторым событием, связанным или даже не связанным с этим процессом.

Рассмотрим механизм передачи управлений программе обработки прерывания (IH). Как было сказано выше, ОС запоминает состояние прерванною процесса и передает управление. Эта операция называется переключением контекста. При реализации переключения используются слова состояния программы (PSW), с помощью которых осуществляется управление порядком выполнения команд. В PSW содержится информация относительно состояния процесса, обеспечивающая продолжение прерванной программы на момент прерывания.(75)

Существуют три типа PSW: текущее, новое и старое PSW (рис.2,13,). Адрес следующей команды (активной программы), подлежащей выполнению, содержится в текущем PSW, в котором также указываются и типы прерываний, разрешенных и запрещенных в данный момент.

Процессор реагирует только на разрешенные прерывания, а запрещенные игнорирует или задерживает их выполнение. Адрес следующей команды, хранящийся в текущем PSW, в командном цикле передается счетчику команд, а адрес новой команды записывается в PSW (по первым двум битам команды можно определить ее длину). Кроме этого, в PSW находятся: признак результата предыдущей команды, поля масок некоторых прерываний (например, каналов и прерываний исключительных ситуаций).

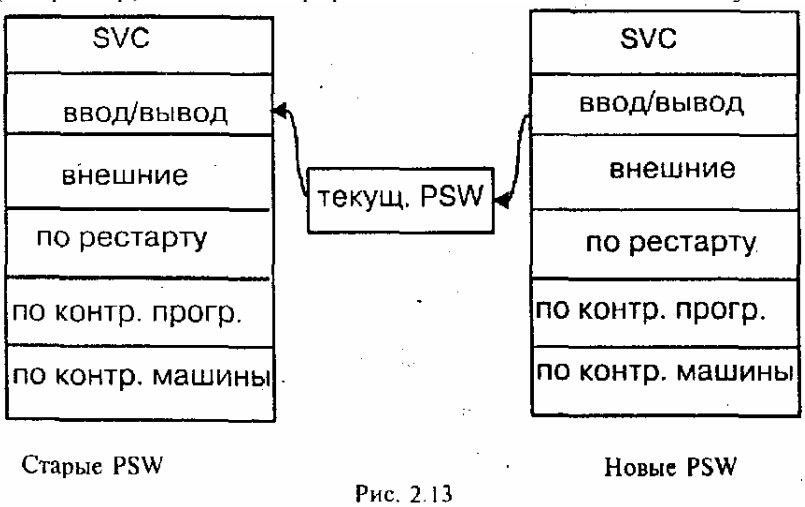


Рис. 2.13

В новых PSW содержатся адреса, по которым резидентно размешаются программы обработки прерываний. При возникновении разрешенного прерывания, в одно из старых PSW (соответствующем типу прерывания) система сохраняет содержимое текущего PSW — адрес следующей команды данного процесса, которая должна выполниться по окончании обработки прерывания н передаче управления данному прерванному процессу (т.е. адрес команды. которая следует за текущей в данном процессе и

которая должна была бы выполниться при отсутствии сигнала прерывания). Таким образом, обеспечивается корректное возвращение в прерванный процесс после обработки прерывания.(76)

Одно из новых PSW, которое содержит адрес обработчика прерывания, соответствующего поступившему сигналу прерывания, записывается *в* текущее PSW, ч происходит переход на соответствующую программу обработки прерывания. Таким образом с помощью механизма смены PSW организуется вход в прерывающую программу. Старые PSW могут храниться а постоянной области памяти вместе с векторами прерываний в стеке той программы, которая прерывается или в PCB прерванного процесса. Если старое PSW находится н постоянной области памяти, то количество выделенных для этого PSW равно количеству классов прерываний, В этом случае глубина прерываний определяется количеством старых PSW.

При выполнении процедуры выхода из прерывания, ОС может передать управление процессору для продолжения выполнения прерванного процесса, если ОС не допускает перехвата процессора более приоритетному процессу (первому в очереди готовых процессов), тогда данный процесс переводится в готовое состояние.

2.7.7. Ядро операционной системы

Операции, связанные с процессами, входят в базовое обеспечение машины. Они включаются в комплекс используемых средств с помощью программ и микропрограмм, совокупность которых составляет ядро ОС. Таким образом, все операции, связанные с процессами, осуществляются под управлением ядра ОС, которое представляет лишь небольшую часть кода ОС в целом. Поскольку эти программы часто используются, то резидентно размещаются в ОП. Другие же части ОС перемещаются в ОП по мере необходимости. Ядро ОС скрывает от пользователя частные особенности физической машины, предоставляя ему все необходимое для организации вычислений:

- само понятие процесса, а значит и операции над ним, механизмы выделения времени физическим процессам;
- примитивы синхронизации, реализованные ядром, которые скрывают от пользователя физические механизмы перестановки контекста при реализации операций, связанных с прерываниями.

Выполнение программ ядра может осуществляться двумя способами:

1. Вызовом примитива управления процессами (создание, уничтожение, синхронизация и т.д.); эти примитивы реализованы в виде обращений к супервизору.
2. Прерыванием: программы обработки прерываний составляют часть ядра, т.к они непосредственно связаны с операциями синхронизации и изолированы от высших уровней управления.(77)

Таким образом, во всех случаях вход в ядро предусматривает сохранение слова состояния (при переключении контекста в PSW) и регистров процессора (в PCB), который вызывает супервизор или обрабатывает прерывание.

2.7.8. Функции ядра ОС

Ядро ОС содержит программы для реализации следующих функций:

- обработка прерываний;
- создание и уничтожение процессов;
- переключение процессов из состояния в состояние;
- диспетчеризацию заданий, процессов и ресурсов;
- приостановка и активизация процессов;
- синхронизация процессов;
- организация взаимодействия между процессами;
- манипулирование PCB;
- поддержка операций ввода/вывода;
- поддержка распределения и перераспределения памяти;
- поддержка работы файловой системы;
- поддержка механизма вызова— возврата при обращении к процедурам;
- поддержка определенных функций по ведению учета работы машины;

Одна из самых важных функций, реализованная в ядре — обработка прерываний.

В систему поступает постоянный поток прерываний и требуется быстрая реакция на них с тем, чтобы эффективно использовались ресурсы системы, и пользователь как можно быстрее получал ответ (на запрос в виде прерывания). При обработке (дешифрации) текущего прерывания ядро запрещает другие прерывания и разрешает их только после завершения обработки текущего. При большой интенсивности прерываний может сложиться такая ситуация, когда ядро будет блокировать прерывания в течение значительной части времени, т.е. не будет иметь возможности эффективно реагировать на прерывания. Поэтому, ядро ОС обычно осуществляет лишь минимально возможную предварительную обработку каждого прерывания, а затем передает это прерывание на дальнейшую обработку соответствующему системному процессу, после начала работы которого ядро могло бы реагировать на последующие прерывания. Таким образом улучшается реакция системы на прерывания (сигнал прерывания).(78)

2.7.9. Синхронизация процессов

Активизированные в системе процессы могут выполняться параллельно. если им для выполнения не, требуется одинаковых ресурсов. Однако, это довольно редкая ситуация в системе. В общем случае несколько процессов могут выполняться параллельно лишь некоторое малое время. В остальное время они синхронизируются или взаимодействуют. Взаимодействие — это передача данных одного процесса другому. Синхронизация необходима для корректности передачи данных или для обращения к общему ресурсу (ОР).

Синхронизация *при передаче данных* заключается в следующем: пока процесс — передатчик не сформирует передаваемые данные и не перешлет их, процесс — приемник не должен выполнять никаких действий, т.е. должен ожидать пересылки данных, и тогда он может продолжить своё выполнение. Процесс — приемник может и сам создать процесс — передатчик, например, при возникновении необходимости ввода данных, При этом процесс — приемник блокируется и передает управление дочернему процессу, по окончании выполнения которого он может продолжить (возобновить) свое выполнение. При реализации такого вида синхронизации необходимо определить порядок предшествования во времени для некоторые точек трасс процессов и определить условие, разрешающее переход некоторых точек трасс на определенные процессы. Эти выделенные точки называются точками синхронизации (когда два выполнявшихся параллельно процесса подошли к точке, в которой необходимо определить, кто из них передатчик и кто приемник; либо точка фиксирует момент, когда создается и запускается дочерний процесс (передается управление от родительского).

Синхронизация *при обращении к ОР* необходима, чтобы исключить одновременный доступ к нему сразу нескольких процессов. Здесь возникает задача взаимного исключения, когда только один процесс может в данный момент "захватить" ОР, а остальные должны ожидать- Критический участок (КУ) — это часть процесса, где происходит обращение к ОР. Процесс, который н данный момент обладает ОР, находится в КУ. В период нахождения н КУ процесс является монополистом по отношению к ОР. Поэтому необходимо, чтобы процесс как можно быстрее проходил свой КУ и не блокировался в нем, иначе может сложиться ситуация, когда несколько (очевидно, наименее приоритетных) процессов будут бесконечно долго ожидать выделения ОР. При выходе процесса из КУ, вход туда может быть разрешен одному из ожидающих в очереди к ОР процессов. Это тоже своего рода синхронизация. Таким образом, процессы, обращающиеся к ОР могут выполняться лишь последовательно.(79)

2.7.10. Примитивы синхронизации

Для синхронизации процессов используются примитивы. Некоторые из них (в основном, семафоры и мониторы) реализованы в ядре (для синхронизации ввода/вывода, переключения контекста (переход по прерыванию) и т.д.).

Примитивы синхронизации могут быть реализованы следующим образом:

1) Самые простые примитивы — флажки (примитивы взаимоисключения). Если флажок равен 0, т.е. условие ложно, то процесс не может войти в КУ, т.к. там уже находится другой процесс; если флажок равен 1 (условие истинно), то процесс входит в КУ, обнуляя флажок (если имеется очередь процессов к ОР, то в КУ входит наиболее приоритетный из них). На основе флажков реализованы алгоритмы синхронизации Деккера. Флажки — программная реализация решения задачи взаимного исключения. Это

самый низкий уровень.

2) Команда test_and_set — аппаратное средство синхронизации (более высокий уровень). Это специальная команда, выполняющая вместе (неделимо) 3 действия:

- чтение значения переменой;
- запись ее значения в область сохранения;
- установка нового значения этой переменной.

2. Семафор — некоторая (защищенная) переменная, значение которой можно считывать и изменять только при помощи специальных операций P и V. Преимущество по сравнению с test_and_set — разделение действий на отдельные.

Операция P(S): проверяет значение семафора S; если S>0, то S:=S-1, иначе (5=0) ждать (по S).

Операция V(S): проверяет, есть ли процессы, приостановленные по данному семафору; если есть, то разрешить одному из них продолжить свое выполнение (войти в КУ); если нет, то S=S+1.

Операция P должна стоять до входа в КУ , а операция V — после выхода из КУ.

Недостатки:

1) громоздкость — в каждом процессе каждый КУ должен окаймляться операциями P и V;
2) невозможность решения целого ряда задач с помощью таких примитивов.

3. Монитор — примитив высокого уровня. Здесь "забор" ставится вокруг ОР, что удобнее (чем семафоры), т.к. ресурсов в несколько раз меньше, чем процессов (их КУ) и ставить "заборы" вокруг КУ слишком(80) громоздко. Можно сказать, что монитор — это некоторый программный модуль, в котором объединены ОР и подпрограммы для работы с ними. При обращении к ОР, т.е. соответствующей процедуре монитора для работы с ОР, осуществляется вход в монитор. В каждый конкретный момент времени может выполняться только одна процедура монитора — это и есть решение задачи взаимного исключения. Если процесс обратился кг процедуре монитора, а другой процесс уже находится внутри монитора, то обратившийся процесс блокируется и переводится во внешнюю очередь процессов — блокированных по входу в монитор. Процесс, вошедший в монитор (т.е. выполняющий его процедуру), также может быть блокирован им, если не выполняется некоторое *условие* X для выполнения процесса. Условие X в мониторе — это некоторая переменная типа condition. С ней связывается некоторая внутренняя очередь процессов, блокированных по условию X. Внутренняя очередь приоритетнее внешней. Для блокирования процесса по переменной X и помещения его во внутреннюю очередь по этой переменной используется операция монитора WAIT(X). При выполнении операции SIGNAL(X) из очереди, связанной с переменной X, извлекается и разблокируется первый процесс. Если внутренняя очередь пуста, то в монитор разрешается войти первому процессу из внешней очереди.

2.7.11. Классические задачи, связанные с доступом к ОР

Производители — потребители. Понятие кольцевого буфера Рассмотрим пару "производитель—потребитель" (рис. 2.14).

Рис. 2.14

В качестве ОР выступает некоторый буфер , куда заносятся и откуда считываются данные. Один процесс—источник или производитель, генерирует информацию, которую другой процесс—получатель или потребитель использует. Это взаимодействие производится при помощи буфера Процесс—производитель генерирует данные либо осуществляет некоторые вычисления и результат заносит в буфер, откуда процесс потребитель считывает данные и печатает их. Если оба процесса (81) работают с примерно одинаковыми скоростями, то проблем передачи/приема практически, не существует; в противном случае (резко различные скорости) возникают трудности синхронизации.

Если процесс—производитель работает быстрее, то он может несколько раз перезаписывать данные в буфере, прежде чем потребитель считает их. Происходит потеря информации, чего нельзя допустить. Тогда процесс—производитель должен дожидаться, пока потребитель не считает данные, только тогда он сможет снова заносить данные в буфер. При этом не будет потери информации (скорость обмена процесса—

потребителя). Если же процесс—потребитель быстрее, то он будет несколько раз считывать одни и те же данные до замены их производителем. Имеет место дублирование информации. В этом случае потребитель должен дожидаться замены информации в буфере производителем.

В ОС предусматривается выделение некоторого количества ячеек памяти для использования в качестве буфера передач. Этот буфер можно представить в виде массива заданного размера. Процесс — производитель помещает передаваемые данные в последовательные элементы этого массива. Потребитель считывает данные в порядке их поступления. Производитель может опережать потребителя на несколько шагов. Со временем, процесс—производитель заполнит последний элемент выделенного буфера—массива. Когда он сформирует очередные данные для передачи, то должен будет возвратиться к началу буфера и продолжить запись, начиная с первого элемента буфера. Такой массив работает как замкнутое кольцо и носит название кольцевого буфера. Так как размер буфера ограничен, процесс—производитель может столкнуться с тем, что все элементы массива заняты. В этом случае производитель должен подождать, пока потребитель не считает и не освободит хотя бы один элемент массива. Может возникнуть ситуация, когда потребитель хотел бы прочитать данные, а массив пуст. Тогда потребитель должен будет ожидать, пока производитель не начнет помещать данные в буфер. Таким образом, такой буфер хорошо использовать для передачи данных от процесса—производителя процессу—потребителю, если они имеют различные скорости. Примером реализации такой схемы является буфер клавиатуры ПК.

Читатели—писатели

Процессы—писатели записывают информацию в некоторый ресурс, например, порт (рис. 2.15).(82) 82

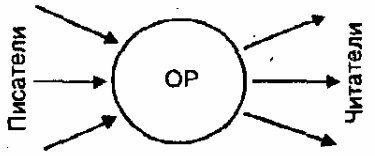


Рис. 2.15

Процессы—читатели должны считывать эту информацию, но записывать не имеют права. Может сложиться ситуация, когда имеется много процессов—писателей, которые поочередно записывают информацию в порт. При этом процессы—читатели могут находиться в ситуации бесконечного откладывания, т.е. ожидания, пока процессы—писатели закончат запись и читатели получат доступ к ОР (порту) для считывания. Поэтому, при появлении нового процесса—писателя приоритет отдается имеющимся в системе процессам—читателям. Однако, теперь уже процессы—читатели могут надолго заблокировать доступ к ОР писателям. В этом случае, при появлении новых процессов—читателей, больший приоритет имеют находящиеся в системе процессы—писатели.

2.7.12. Тупики. Тупиковые ситуации и методы борьбы с ними

Тупик — это:

1. Ситуация, из которой система не может выйти;
2. Ситуация, когда процесс ждет события, которое никогда не произойдет.

Примером из жизни может служить транспортная пробка на перекрестке.

Второй пример; круговая цепь ожидания (рис. 2.16). Здесь ресурс 2 выделен процессу 1, а ресурс 1 — процессу 2. В какой-то момент процесс 1 затребовал дополнительно ресурс 1, а процесс 2 — ресурс 2. Естественно, ни один из этих ресурсов не может быть выделен затребовавшему его процессу. Получаем тупиковую ситуацию, при которой ни один из двух процессов не может, закончить свое выполнение, пока не будет освобожден требуемый ресурс, чего в данном случае никогда не сможет произойти. Выход заключается в удалении одного из процессов попавших в тупиковую ситуацию из системы.

Тупик *неизбежен,* если присутствует четыре условия его возникновения:

1. Условие *взаимного исключения.* Процесс обладает монопольным правом владения ресурсами во время всего существования процесса.(83)
2. Условие *ожидания.* Процесс, обладая монопольным правом, пытается захватить новые ресурсы.
3. Условие *перераспределения.* Ресурс нельзя отнять до полного завершения процесса.
4. Условие: *круговая цепь ожидания.* Каждый из процессов цепочки требует дополнительный ресурс,

который уже выделен процессу данной цепочки.

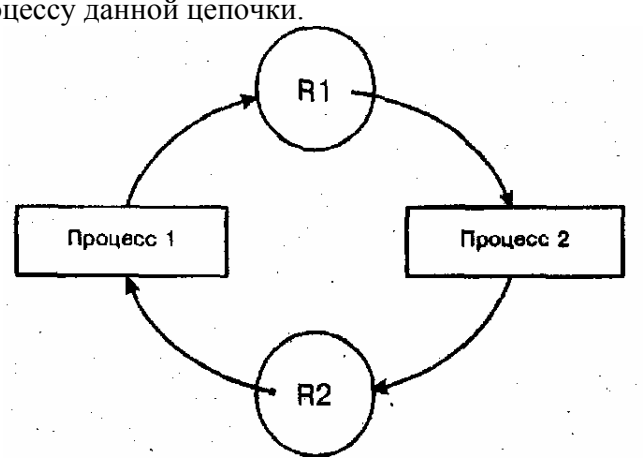


Рис. 2.16 Тупики в системе спулинга

При вводе заданий используется режим спулинга — режим буферизации для выравнивания скоростей при вводе и считывании информации из буфера- Система, использующая режим спулинга, сильно подвержена тупикам.

В таких системах буфер может быть полностью заполнен до того, как данные из него станут поступать на печатающее устройство, а его размера недостаточно для буферизации всех данных вывода. При этом возникает тупиковая ситуация, т.к. размер буфера недостаточен для помещения всех поступивших данных, а принтер не сможет начать вывод. В данном случае единственным выходом является рестарт системы, но при этом теряется вся информация.(84)

В таких системах хорошим выходом было бы неполное заполнение буфера (на 75-80%) и передача после этого управления печатающему устройству. По освобождении буфера можно его снова неполностью заполнить, затем снова перейти к распечатке и т.д.

Во многих современных ОС принтер начинает распечатку до заполнения буфера (по мере заполнения). При этом часть буфера (распечатанная) освобождается и туда можно заносить новую информацию. В таких системах вероятность возникновения тупиковой ситуации значительно меньше.

2.7.13. Способы борьбы с тупиками

Последствия тупиков равносильны ситуации бесконечного откладывания. Тупики и ситуация бесконечного откладывания — это ситуации равносильные "потери процесса".

Можно выделить 4 стратегии борьбы с тупиками:

1. Предотвращение тупика. Требуется создать условия, исключающие возможность возникновения тупиковых ситуаций. Однако этот метод часто приводит к нерациональному использованию ресурсов.
2. Обход тупика. Этот метод предусматривает менее жесткие ограничения, чем первый и обеспечивает лучшее использование ресурсов. Метод учитывает возможность возникновения тупиков, и при увеличении вероятности тупиковой ситуации принимаются меры по обходу тупика.
3. Обнаружение тупиков. Требуется установить сам факт возникновения тупиковой ситуации, причем, точно определить те процессы и ресурсы которые в нее включены.

Восстановление после тупиков. Необходимо устранить тупиковую ситуацию, чтобы система смогла продолжить работу, а процессы, попавшие а тупиковую ситуацию, смогли завершиться и освободить занимаемые ими ресурсы. Восстановление — это серьезная проблема, так что в большинстве систем оно осуществляется путем полного выведения из решения одного или более процессов, попавших в тупиковую ситуацию. Затем выведенные процессы обычно перезапускаются с самого начала, так что вся или почти вся проделанная процессами работа теряется.(85)

2.7.13.1. Предотвращение тупиков

Возникновение тупика невозможно, если нарушается одно из четырех необходимых условий возникновения тупиковых ситуаций. Первое условие при этом можно и не нарушать, т.е. считаем что

процесс может обладать монопольным правом владения ресурсами. Нарушая остальные три условия получаем следующие три способа предотвращения тупиков:

1. способ. Процесс запрашивает и получает все ресурсы сразу. Если процесс не имеет возможности получить все требуемые ресурсы сразу, т.е. некоторые из них на данный момент заняты, то он должен ожидать освобождения требуемых ресурсов. При этом он не должен удерживать за собой какие-либо ресурсы. Нарушается условие "ожидания дополнительных ресурсов" (2-е) и тупиковая ситуация невозможна. При этом способе имеем простой процессов, ожидающих выделения ресурсов, и работу вхолостую значительной части ресурсов. Можно прибегнуть к разделению программы на несколько программных шагов, работающих независимо друг от друга. При этом выделение ресурсов можно осуществлять для каждого шага программы, однако увеличиваются расходы на этапе проектирования прикладных программ. Этот способ имеет два существенных недостатка:

1. Снижается эффективность работы системы.
2. Усиливается вероятность бесконечного откладывания для всех процессов.

2 способ. Если процесс, удерживающий за собой определенные ресурсы, затребовал дополнительные и получил отказ в их получении, он должен освободить все ранее полученные ресурсы. При необходимости процесс должен будет запросить их снова вместе с дополнительными. Здесь нарушается условие "перераспределения" (3-е). При этом способе каждый процесс может несколько раз запрашивать, получать и отдавать ресурсы системе. При этом система будет выполнять массу лишней работы — происходит деградация системы с точки зрения полезной работы. Недостатки этого способа:

1. Если процесс в течение некоторого времени удерживал ресурсы, а затем освободил их. он может потерять имевшуюся информацию.
2. Здесь также возможно бесконечное откладывание процессов — процесс запрашивает ресурсы, получает их, однако не может завершиться, т.к. требуются дополнительные ресурсы; далее он, не завершившись, отдает имеющиеся у него ресурсы системе; затем он снова запрашивает ресурсы и т.д. Имеем бесконечную цепочку откладывания завершения процесса.(86)

III способ. Стратегия линейности. Все ресурсы выстроены в порядке присвоенных приоритетов и захват новых ресурсов может быть выполнен только в порядке возрастания приоритетов. При этом нарушается условие "круговая цепь ожидания" (4-е). Трудности и недостатки данного способа:

1. Поскольку запрос ресурсов осуществляется в порядке возрастания приоритетов, а они назначаются при установке машины, то в случае введения новых типов ресурсов может возникнуть необходимость переработки существующих программ и систем;
2. Приоритеты ресурсов должны отражать нормальный порядок, в котором большинство заданий используют ресурсы. Однако, если задание требует ресурсы не в этом порядке, то оно будет захватывать и удерживать некоторые ресурсы задолго до того, как появится необходимость их использования —теряется эффективность.
3. Способ не предоставляя ет удобства пользователям.

2.7.13.2. Обход тупиков

Алгоритм "банкира". При использовании алгоритма "банкира" подразумевается, что:

1) системе заранее известно количество имеющихся ресурсов;
2) система знает, сколько ресурсов может максимально потребоваться процессу;
3) число пользователей (процессов), обращающихся к ресурсам, известно и постоянно;
4) система знает, сколько ресурсов занято в данный момент времени этими процессами (в общем и каждым из них);
5) процесс может потребовать ресурсов не больше, чем имеется в системе.

В алгоритме "банкира" введено понятие надежного состояния. Текущее состояние вычислительной машины называется надежным, если ОС может обеспечить всем текущим пользователям (процессам) завершение их заданий в течение конечного времени, Иначе состояние считается ненадежным. Надежное состояние — это состояние, при котором общая ситуация с ресурсами такова, что все пользователи имеют возможность со временем завершить свою работу. Ненадежное состояние может со временем привести к тупику.

Система удовлетворяет только те запросы, при которых ее состояние остается надежным, т.е. при

запросе ресурса система определяет сможет ли она завершить хотя бы один процесс, после этого выделения.

Пример решения задачи выделения ресурсов по алгоритму "банкира".(87)

Имеем 6 ресурсов и 6 процессов (рис. 2.17.). В таблице показано состояние занятости ресурсов на данный момент (1 свободный ресурс).

Процесс	Выделенное число ресурсов	Требуемое число ресурсов
A	2	3
B	1	3
C	0	2
D	1	2
E	1	4
F	0	5

Рис. 2.17

Эта таблица отражает надежное состояние, т.к. существует такая последовательность выделений - освобождения ресурсов, при которой вес процессы за конечное время будут завершены.

Недостатки алгоритма банкира:

1) Если количество ресурсов большое, то становится достаточно сложно вычислить надежное состояние.

2) Достаточно сложно спланировать, какое количество ресурсов может понадобиться системе.

3) Число работающих пользователей (процессов) остается постоянным.

4) Пользователи должны заранее указывать максимальную потребность в ресурсах, однако, потребность может определяться динамически по мере выполнения процесса.

5) Пользователь всегда заказывает ресурсов больше, чем ему может потребоваться.

6) Алгоритм требует, чтобы процессы возвращали выделенные им ресурсы в течение некоторого конечного времени. Однако для систем реального времени требуются более конкретные гарантии. Т.е. в системе должно быть задано максимальное время захвата всех ресурсов процессом (через это время ресурсы должны быть освобождены). Графический метод обхода тупика (рис. 2.18). Имеются 2 процесса Р"1 и Р2 и каждому из них для выполнения требуются по 2 ресурса; Д—диск , П — процессор. Если Р1 захватил процессор , а Р2 — диск, то возникает тупиковая ситуация. Однако, если Р1(