Обзорная лекция по курсу «Архитектура вычислительных систем» для специальности Т10.02

1. Архитектура как набор взаимодействующих компонентов

Ранее область применения вычислительных систем определялась ее быстродействием. Однако существует достаточно большое количество ВС, обладающих равным быстродействием, но имеющих совершенно разные способы представления данных, методы организации памяти, режимы работы, системы команд, набор ВнУ и т. д. Таким образом, ВС имеет, кроме быстродействия, ряд других характеристик, необычайно важных в той или иной области применения. Это стало особенно заметно при переходе к ВС четвертого и пятого поколений. Совокупность таких характеристик и легла в основу понятия архитектуры ВС.

Архитектура ВС определяет основные функциональные возможности системы, сферу применения (научно-техническая, экономическая, управление и т. д.), режим работы (пакетный, мультипрограммный, разделения времени, диалоговый и т. д.), характеризует параметры ВС (быстродействие, набор и объем памяти, набор периферийных устройств н т. д.), особенности структуры (одно-, многопроцессорная) и т. д. Составные части понятия "архитектура" можно определить следующей схемой (рис. 1.1).

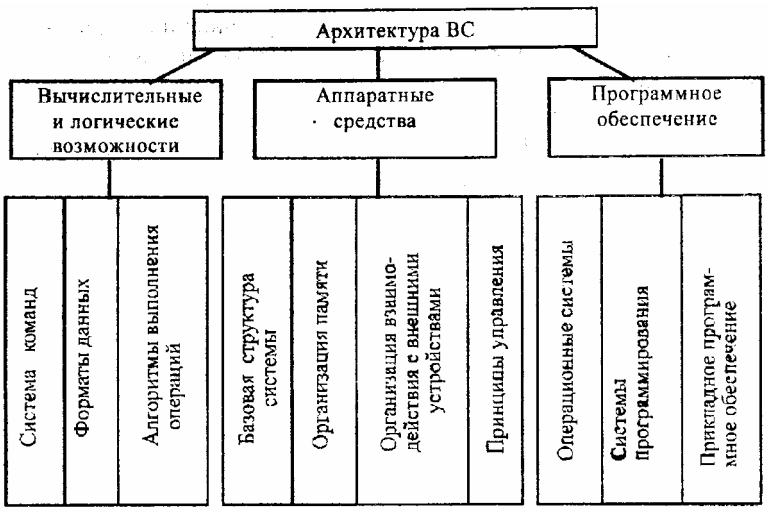


Рис. 1.1. Функциональные возможности ВС

Вычислительные и логические возможности ВС. Они обусловливаются системой команд (СК), характеризующей гибкость программирования, форматами данных и скоростью выполнения операций, определяющих класс задач, наиболее эффективно решаемых на ВС. Система команд ВС, базирующихся на архитектуре фон Неймана, сегодня мало чем отличается от СК ЭВМ 50-х годов. Большинство достижений в этой области остались незамеченными проектировщиками и соответственно не нашли адекватного воплощения в архитектуре современных компьютеров.

Анализ показывает, что в различных программах чаще всего встречаются достаточно простые команды: команды пересылки и команды процессора с использованием, регистров и простых режимов адресации. Не нашли широкого применения и нетрадиционные способы кодирования данных, несмотря на значительные возможности их в плане разработки быстродействующих алгоритмов арифметических операций. Среди них знакоразрядные системы, системы в коде вычетов и др.

Рассмотрим структуру системы команд в зависимости от класса решаемых задач (рис. 1.2).

К командам управления мы относим команды ввода-вывода данных и команды управления состоянием процессора, памяти и каналов.

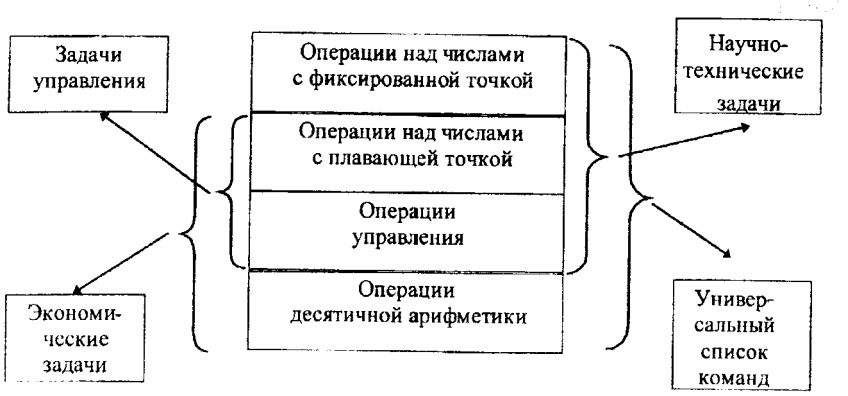


Рис. 1.2. Классификация СК по назначению

Как видно из рис. 1.2, для решения задач любого класса необходимы команды типов 2 и 3. Следовательно, эти типы команд должны присутствовать в любом компьютере.

Большое влияние на точность выполнения операций оказывают форматы данных. Современные компьютеры имеют развитую систему форматов.

Алгоритмы выполнения операций достаточно полно отражают производительность только однопроцессорных ВС.

Аппаратные средства. Простейшая ВС включает модули пяти типов:

центральный процессор, основная память, каналы, контроллеры и внешние устройства.

Процессор (УУ + АЛУ + намять) управляет работой системы и обеспечивает вычисления непосредственно по программе. Выполнение машинных команд, команд ввода-вывода (I/O), обращение к памяти, управление состоянием устройств инициализируются или выполняются с помощью процессора.

Основная память предназначена для хранения команд и данных и обеспечивает адресный доступ к ним от процессора. Современная память работает со скоростью, близкой к скорости работы процессора.

Каналы - спецустройства, управляющие обменом данных с внешними устройствами. Каналы инициируют свою работу с помощью процессора и затем переходят в автономный режим работы. Это, по сути, спецпроцессор ввода - вывода, обеспечивающий работу внешних устройств, контроль информации и т. д.

Контроллеры ввода-вывода служат для подсоединения внешних устройств (ВнУ) к каналам и обеспечивают обмен управляющей информацией с внешними устройствами, присвоение приоритетов и выдачу информации о состоянии ВнУ для канала, т. е. это устройства управления ВнУ.

ВнУ служат для ввода-вывода информации с различных носителей. Память может быть организована как многоуровневая с различным объемом и временем доступа к ней - сверхоперативная (СОЗУ), оперативная (ОП), внешняя (ВнП) (рис. 1 3), так и одноуровневая, виртуальная. Почти всегда виртуальная память есть переупорядоченное подмножество реальной памяти.

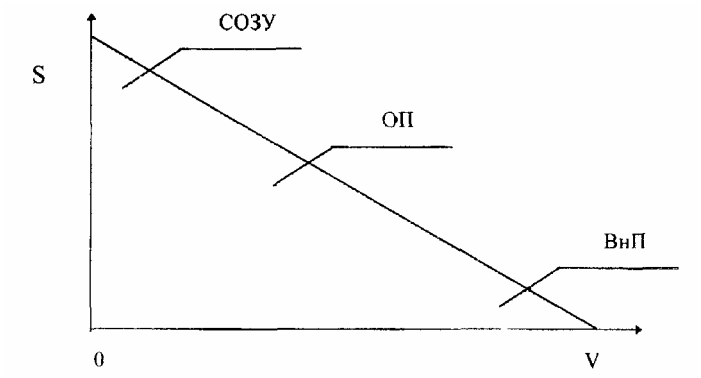


Рис 1 3 Типы памяти (V - объем, S - быстродействие)

Уровни иерархии памяти взаимосвязаны между собой, все данные одного уровня могут быть найдены на более низком уровне.

Успешное или неуспешное обращение к уровню памяти называют соответственно

попаданием (hit) или промахом (miss), а соответствующее время - временем обращения (hit time или miss penalty).

Существенное влияние на производительность ВС оказывают каналы ввода-вывода. Мультиплексный канал обеспечивает работу группы медленных устройств, блок-мультиплексный - группы быстрых устройств, селекторный - монополизирует информационную магистраль только одним быстродействующим устройством.

Для повышения пропускной способности каналов используют некоторые дополнительные меры, например буферизацию ВнУ путем введения памяти в состав самого устройства или контроллера.

Аппаратные средства защиты памяти служат для управления доступом к различным областям памяти в соответствии с имеющимися у пользователя полномочиями.

Программное обеспечение. Оно является составной частью архитектуры компьютера и существенно влияет на весь вычислительный процесс, в частности позволяет эффективно эксплуатировать аппаратные средства системы.

Операционная система (ОС) управляет ресурсами, разрешает конфликтные ситуации" оптимизирует функционирование системы в целом.

Широкий спектр языков программирования позволяет описывать практически любые задачи, а разнообразие компиляторов - их эффективно реализовывать.

Роль прикладного программного обеспечения (ПО) необычайно велика для решения тематических задач.

2. Архитектура как интерфейс между уровнями физической системы

Применительно к ВС термин "архитектура" может быть определен как распределение функций, выполняемых системой, по различным уровням и установление интерфейса между этими уровнями. На рис. 2.1 представлен набор уровней абстракции как специфического свойства архитектуры ВС Остановимся лишь на ключевых уровнях.

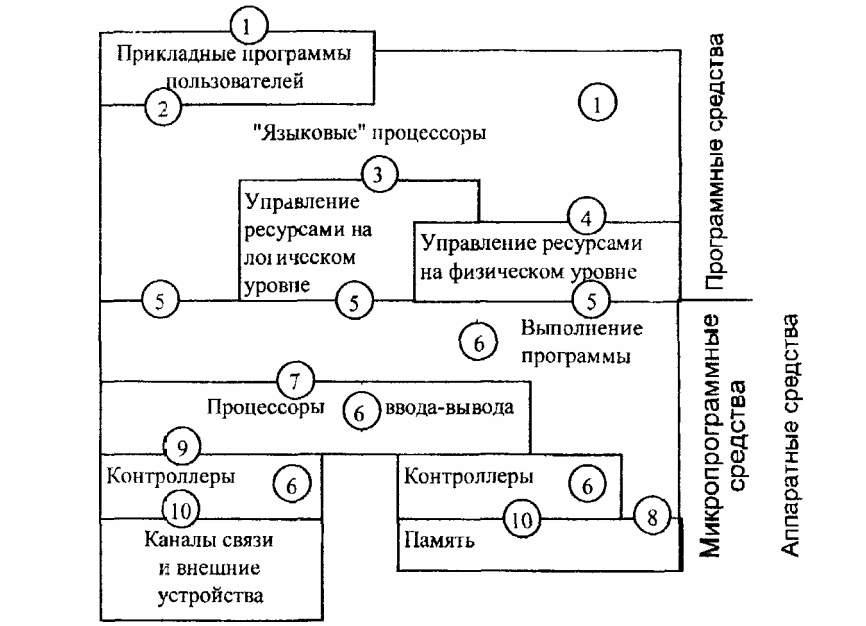


Рис 2.1 Многоуровневая организация архитектуры вычислительной системы

Архитектура первого уровня определяет, какие функции по обработке данных решаются системой, а какие передаются внешнему миру: пользователю, оператору ЭВМ, администратору баз данных и т.д. Система взаимодействует с внешним миром через два набора интерфейсов: языки (язык программирования, язык оператора терминала, язык управления заданиями, язык общения с базой данных, язык оператора ЭВМ) и системные программы (программы редактирования, связи, оптимизации, восстановления и обновления информации, интерпретации, управления и т.д., т.е. программы, созданные разработчиком системы). Оба интерфейса должны быть созданы при разработке архитектуры системы.

Уровни, определяемые интерфейсами внутри программного обеспечения, могут быть представлены как архитектура программного обеспечения. К примеру, если прикладные задачи реализованы на языках программирования, которые не входят в набор языков, предоставляемых системой пользователю, то здесь речь может идти об архитектуре уровня, позволяющего определить указанные языки.

Уровень 5 является одним из центральных уровней архитектуры и проводит разграничение между системным программным и аппаратным (т.е. схемным и микропрограммным) обеспечением. Он позволяет представить физическую структуру системы независимо от способа реализации.

Остальные уровни отражают интерфейсы и распределяют функции между отдельными частями физической системы.

Таким образом, можно сказать, что архитектура компьютера это абстрактное представление физической системы с точки зрения программиста.

3 Концепция виртуальной памяти

Каждая часть среды компьютера имеет собственное обозначение: ячейка - адрес, периферийное устройство - номер и т. д. В простейших компьютерах собственные обозначения указываются непосредственно в программе. В более сложных компьютерах программа отделена от среды "аппаратом преобразования собственных обозначений". Рассмотрим один элемент среды - память. Аппарат преобразования адреса (АПА) не находится под прямым управлением программы и связь с ним осуществляется только через процедуры, работающие в управляющем режиме.

Если программисту безразлично существование АПА, то он работает с набором ячеек и периферийных устройств, образующих "виртуальную (математическую, мнимую) среду". Почти всегда виртуальная среда есть переупорядоченное подмножество реальной среды. Каждому виртуальному элементу соответствует реальный элемент, но обратное не всегда верно.

Рассмотрим один из элементов виртуальной среды - виртуальную память (ВП).

Задачи, решаемые виртуальной памятью

Виртуальный адрес - адрес, по которому ссылаются на ячейку виртуальной памяти. Область виртуальных адресов - это множество всех виртуальных адресов.

Использование виртуальной адресации обусловливается следующими обстоятельствами.

1. Однородность области адресов. Представим себе реальный компьютер без виртуальной памяти. Пусть на нем выполняется параллельно несколько процессов. У каждого процесса будет отдельная локальная среда и каким-то образом распределяемые элементы общей среды. Программисту требуется заранее знать, к каким конкретно частям общей среды его процедура может обращаться. Это затруднительно для пользователей ЭВМ, составляющих свои собственные программы. Отвести наперед фиксированную область среды для каждого процесса невозможно, ибо положение каждой конкретной программы определяется положением всех других программ.

При виртуальной адресации каждый процесс может выполняться в памяти начиная с фиксированной (обычно нулевой) ячейки, имеющей необходимы размеры области ЗУ. Автору безразлично, в каком участке памяти выполняется его программа, так как каждое обращение к виртуальной памяти во время выполнения посредством АПА преобразуется в реальное обращение.

Замечание. АПА работает не во время ассемблирования, а непосредственно во время выполнения обращения,

2. Защита памяти. Общеизвестно, что основная цель защиты памяти состоит в том, чтобы не дать возможности некорректному процессу испортит часть среды, относящуюся к другому процессу. Особенно это важно при защите сред управляющих процедур. Виртуальная адресация здесь используется следующим образом: при каждой ссылке процессом на память проверяется, принадлежит ли она к области виртуальных адресов, отведенных для данного процесса.

3. Изменение структуры памяти. При проектировании больших программ структура памяти машины с малой ОП явно усложняет проектируемую программу. Применение виртуальной адресации позволяет преобразовать память на разных ступенях иерархии в "одноуровневую память" с одинаковым доступом ко всем элементам и отобразить ее на реальную память.

Для удовлетворения пунктов 1-3 требуется аппарат "страничной" организации памяти, для пунктов 1,2 достаточно иметь регистры "настройки" регистры "базы" и "границы".

1.6.2. Страничная организация памяти

Отображение виртуальной памяти в реальную обычно осуществляется с помощью страничной организации памяти.

Виртуальную память в системе со страничной организацией памяти делят на ряд "блоков" фиксированной длины, равной 2k где k - целое натуральное число. Так как первая ячейка блока N + 1 примыкает к последней ячейке блока N, то программисту факт разбиения ВП на блоки учитывать не требуется.

Оперативная память компьютера делится на "страницы", а вспомогательная - на "сегменты" такого же размера.

Виртуальную память пользователя можно разделить на три типа:

•"активные" блоки, которые содержат программу и данные, используемые в текущий момент;

•"пассивные" блоки, содержащие программу и данные, которые будут использоваться при выполнении программы;

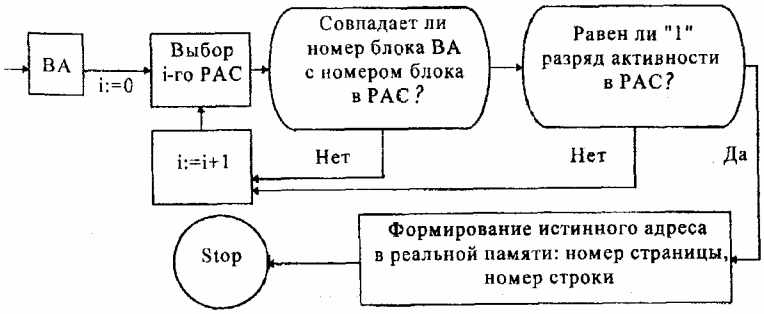
•"мнимые" блоки, к которым не обращаются на протяжении выполнения программы.

Соотношения между первым и вторым типами блоков ВП в процессе выполнения программы изменяются. Третий тип блоков возможен лишь при наличии очень большой области ВП,

Аппарат виртуальной адресации должен отображать виртуальную среду на реальную, причем так, чтобы "активные" блоки находились в оперативной памяти, "пассивные" - по возможности в оперативной или вспомогательной, а мнимые - нигде.

Преобразование виртуального адреса в реальный происходит с помощью регистров адресов страниц (РАС). Аппарат виртуальной адресации отображает виртуальный адрес в реальный следующим образом; виртуальный адрес сравнивается одновременно с содержимым всех РАС. Единственным сравнимым с ним РАС будет тот, который содержит гот же номер блока и "1" в разряде активности. РАС определяет номер страницы, с которой он связан. Для получения реального адреса памяти к номеру страницы данного РАС присоединяется номер строки из виртуального адреса (ВА). В последовательной интерпретации процесс отображения ВА в реальный можно представить следующей схемой.

Разряд записи в РАС служит для экономии времени перезаписи "страницы" в ВнП. Когда блок переносится из ВнП в оперативную память, в разряд записи пишут "О". Если какая-то строка данной страницы ОП изменяется в процессе обращения к ней, то пишут "1". И пока в разряде записи "0", эта страница является точной копией соответствующего блока в ВнП.



В разряд использования "1" посылается при очередном обращения к данной странице. Через равные промежутки времени содержимое регистров использования сканируется и записывается в определенную ячейку памяти тем самым создавая статистику использования данной страницы.

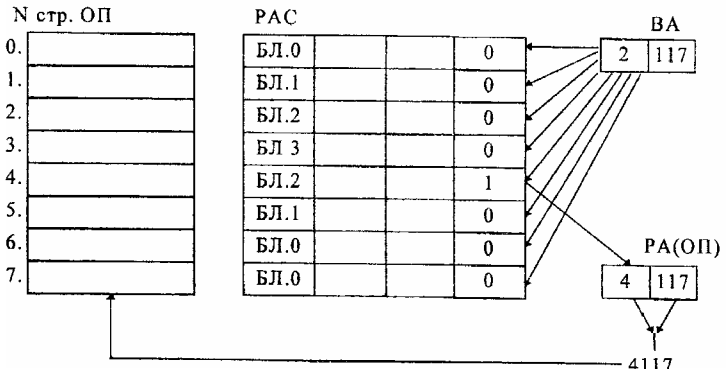
Полный процесс отображения ВП в реальную выполняется за три этапа. Д

1. Если происходит обращение к блоку ВП, который должен отобрав жаться на страницу ОП, РАС для которой является одним из ассоциативных регистров, то процесс отображения выполняется согласно выше приведённой схеме.

2. Если обращение происходит к блоку ВП, для которого условия п. 1 не выполняются, то номер виртуального блока служит указателем для обращения к таблице блоков (ТБ) в оперативной памяти. Если из ТБ видно, что искомый блок находится в оперативной памяти, то номер страницы также может быть выбран из ТБ.

Если же нужный блок находится во вспомогательной памяти или отсутствует вообще, то происходит прерывание, и тогда переходим к управляющей процедуре (см. п. 3)

3. Обработка прерывания. Для эффективной работы системы необходимо, чтобы ассоциативные регистры содержали РАС для наиболее часто используемых страниц. Основная цель методов оптимизации распределения реальной памяти состоит в минимизации числа обменов между оперативной и вспомогательной памятью. Отсюда следует, что необходимо корректно выбирать стратегию замещения страниц.



Основные стратегии замещения страниц, наиболее часто используемые на практике: циклическое изгнание, случайная выборка, наименьшее число обращений с момента последнего прерывания. Результаты экспериментов показали, что ни в одном случае разница в числе обменов, требуемых конкретной задачей при использовании указанных стратегий, не превышала 10 %.

4. Взаимодействие и управление процессами

Понятие "программа" - недостаточно мощное понятие для описания «рационных систем, однако детальное исследование программ позволяет выделить ряд важных концепций, которые проясняют принципы построения операционных систем.

При выполнении программ явно выделяются три объекта:

• последовательность команд или процедура, которая определяет программу;

• процессор, который выполняет процедуру;

• среда, т.е. та часть окружающею мира, которую процессор может непосредственно воспринимать или изменять.

К среде относятся, например, память, универсальные и управляющие регистры ЭВМ, поскольку они могут и восприниматься, и изменяться программой.

Можно выделить следующие свойства программы:

• операции, заданные процедурой, выполняются строго последовательно, т. е. следующий шаг не начинается, пока предыдущий полностью не выполнится (из определения программы мы исключаем любой процесс, содержащий совмещение операций);

• среда полностью управляется программой, а следовательно, и изменяется только в результате шагов, выполненных программой;

• время выполнения операций, а также временной интервал между выполнением операций не имеют отношения к выполнению программы. Естественно, здесь мы не учитываем, что вся программа должна быть выполнена за разумный интервал времени;

• совершенно не имеет значения, выполняется ли программа целиком на одном процессоре, лишь бы не изменялась среда программы.

Программа в силу указанных свойств предполагает отсутствие внешнего воздействия на её выполнение. Хорошим приближением такого рода программ являются программы, написанные на языках высокого уровня. Важное достоинство таких программ - возможность точного повторения их работы, если только она не имеет доступа к часам реального времени.

Программы, удовлетворяющие указанным свойствам, не могут быть операционными системами, ибо:

• предполагается, что операционная система эффективно использует ресурсы компьютера, а это требует совмещения операций различных компонентов;

• ОС отвечает на поступающие запросы за определенное время, а так как они поступают произвольным образом, то последовательность операций определяется не только самой системой.

5. Понятие процесса и состояния

Создавать современные операционные системы без теоретической базы стало затруднительно, так как невозможно повторить условия, приведшие к ошибке во время работы, а следовательно, выявить ее источник. Вес это привело к появлению понятия процесса. Понятие "процесс" в последнее время встречается в литературе по операционным системам довольно часто, однако вкладываемый в это понятие смысл иногда сильно различается. Мы будем пользоваться следующим определением.

Процесс есть тройка (Q, f, g), где Q - множество состояний процесса, f - функция действия f: Q -> Q; g - начальное состояние процесса.

Действия, реализуемые процессом, будем рассматривать как результат выполнения некоторой программы на реальном (виртуальном) процессоре.

Перечислим некоторые свойства процесса:

• процесс не является закрытой системой и может взаимодействовать с другими процессами, воспринимая или изменяя часть среды, которую он с ними разделяет;

• каждый процесс живет лишь временно. Имеется и "главный" процесс выполняемый вручную при включении компьютера, который начинает всю цепочку процессов;

• в любой момент процесс может быть описан его состоянием. Все параметры (переменные), характеризующие текущее состояние процесса, объединяются в "вектор состояний" или "слово состояний", которые и позволяют возобновить процесс после его прерывания, в том числе и локальную среду.

Цикл жизни процесса может быть представлен как переход его из одного состояния в другое.

Пример. Пусть заданы три процесса (задания пользователя), одновременно присутствующие в системе с мультипрограммированием. И пусть каждый из них может находиться в трех состояниях: ожидание, готовность, выполнение.

Опишем эти состояния.

ОЖИДАНИЕ. Процесс ожидает выполнения какого-либо события (например, завершения операции I/O).

ГОТОВНОСТЬ. Процесс готов к выполнению, но процессов больше чем процессоров, и он должен ждать своей очереди.

ВЫПОЛНЕНИЕ. Процессору выделены все ресурсы, в том числе и процессор, и его программы выполняются в настоящее время.

Схема на рисунке предполагает, что процессы уже существовали в памяти компьютера и будут существовать всегда. На практике пользователь должен эти процессы (задания)

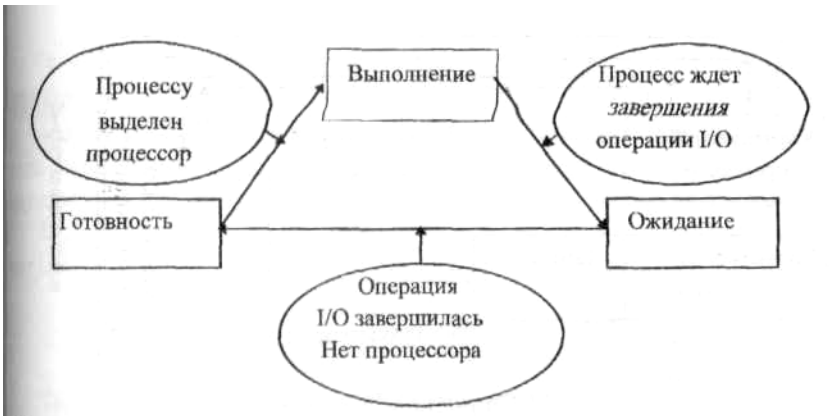


Рис. 7.1. Схема перехода процессов из одного состояния в другое

Рис. 7.2. Полная схема обработки процесса

предоставить системе. Тогда реальная схема будет выглядеть так.

Здесь дополнительно введены состояния: т доставление, хранение и завершение. Поясним их.

ПРЕДОСТАВЛЕНИЕ. Пользователь предоставляет задание системе, на которое она должна отреагировать.

ХРАНЕНИЕ. Задание преобразовано во внутреннюю форму, понятную лишь компьютеру, но ресурсы еще не выделены (например, задание записано на диск). Процесс, в случае выделения ресурсов, переходит в состояние готовности.

ЗАВЕРШЕНИЕ. Процесс завершил свои вычисления и все выделенные процессу ресурсы могут быть освобождены и возвращены системе.

6. Организация системы прерывания.

Появление системы прерывания в компьютерах конца 50-х годов позволило существенно продвинуться в разработках ЭВМ, по новому переосмыслив их возможности и среды применения. Действительно, уже на первых ЭВМ, обладающих этим свойством, появилась возможность автономной работы периферийных устройств после их запуска центральным устройством правления. По окончании работы или в связи с другой причиной периферийное устройство смогло через прерывание заставить центральный процессор обратить на себя внимание. Это открыло путь к широкому совмещению операций, что повлекло за собой естественное усложнение математического обеспечения. Современные операционные системы полностью разрабатываются и базируются на развитой системе прерывания.

Основные определения и характеристики

Работу вычислительной системы можно представить как последовательность программно-определяемых (порождаемых программой и возможные моменты появления которых известны) и программно-независимых (вызванных посторонними от программы источниками или моменты возникновения которых неизвестны) событий.

События, происходящие вне процессора, как правило, программно-независимые (выход параметров объекта за дозволенные пределы, запросы оператора и т. д.) и происходят асинхронно. То же относится и к периферийным устройствам, работающим одновременно с выполнением программы в процессоре, хотя начало их работы и задает процессор, однако её окончание и последовательность операций неизвестны.

События, происходящие внутри процессора, могут быть двух типов. Последовательность арифметических операций определяется программой, в то время как особые ситуации (переполнение, попытка деления на нуль и т.д.) зависят от сочетания операндов и предусмотреть их во время программирования практически невозможно. ВС, вообще говоря, должна реагировать на любые события, которые могут повлиять на процесс вычислений. В случае программно-определяемых событий для этого достаточно иметь специальный набор команд (переход по нулю, по знаку и т. д.). Если же события программно-независимые, то, как правило, неэффективно использовать обычные программные методы для их опознания.

Чтобы ВС могла реагировать на программно-независимые события при минимальных усилиях программиста и максимально возможном быстродействии, ее надо снабдить дополнительными аппаратно-логическими средствами, совокупность которых называют системой прерывания программ (СПП)

Определение. Прерывание программы это свойство ВС при возникновении особых событий временно прекратить выполнение текущей программы и передать управление программе, специально предусмотренной для обработки данного события.

В системе с прерыванием каждое программно-независимое событие (источник прерывания) должно, если оно может повлиять на ход обработки сопровождаться сигналом, говорящим о его возникновении. Назовем эти сигналы запросами прерывания. Программы, затребованные запросами прерывания, назовем прерывающими программами, в отличие от прерванных грамм, выполнявшихся компьютером до появления запросов прерывания.

Так как функции по сохранению и восстановлению состояния прерванной программы возлагаются на саму прерывающую программу, то последняя должна состоять из трех частей: подготовительной и восстановительной, обеспечивающих переход к нужной программе, и собственно прерывающей программы.

По окончании работы прерывающей программы переход может быть осуществлён либо к прерванной программе, либо к другой прерывающей программе.

Так как всевозможные запросы на прерывание вырабатываются независимо и асинхронно, то возможны такие ситуации:

• приход запросов последовательный;

• одновременный приход нескольких запросов;

• приход запроса во время выполнения прерывающей программы. Следовательно, должен быть организован порядок, в котором поступающие запросы удовлетворяются. Если в ВС имеются средства для обслуживания запросов в порядке присвоенного им приоритета, то такие системы срывания называются приоритетными.

СПП, как правило, выполняют следующие основные функции:

• организуют вход в прерывающую программу;

• осуществляют приоритетный выбор между запросами прерывания;

• обеспечивают возврат к прерванной программе и программное изменение приоритетов программ.

Вход в прерывающую программу

Система прерывания программ должна определить допустимый момент прерывания текущей программы и начальный адрес прерывающей программы. Наиболее простыми являются три следующих способа определения допустимого момента прерывания.

Метод помеченного оператора (опорных точек). Суть метода состоит в следующем. В специальные разряды команд, после которых допускается прерывание, записывается определенный знак, разрешающий или запрещающий прерывание. Тогда вдоль программы можно расставить её опорные точки. Здесь желательно так расставить точки прерывания, чтобы информация, находящаяся в регистрах процессора после выполнения данной команды, дальше не использовалась. Это уменьшает время обслуживания и увеличивает время реакции.

Покомандный способ. Здесь прерывание допускается после выполнения любой команды. Способ прост в реализации.

Метод быстрого реагирования. Прерывание допускается во время выполнения любой команды, т.е. после выполнения её очередного такта.

Из-за простоты и удачного сочетания характеристик прерывания наибольшее распространение получил второй способ, хотя в последних ВС используется и третий. Так, если обнаружено, что адрес операнда сформирован неверно, то целесообразно сразу же прервать выполнение операции, чтобы ошибка не распространилась на другие такты. Это необходимо при мультипрограммной работе, если адресованный операнд в команде принадлежит внешней памяти. Здесь текущая операция не может быть продолжена, пока итеративная память не получит данные из ВнП.

Распознавание начального адреса прерывающей программы можно осуществлять как программным, так и аппаратным способом.

Суть программного распознавания: все линии связи, по которым приходят запросы прерывания, объединяются в схему ИЛИ, формирующую на выходе один и тот же сигнал, который в допустимый момент прерывания поступает в прерывающую программу. Последняя распознает запросы и разветвляется для их выполнения.

Вход системы прерывания, обладающей способностью формировать собственный адрес начала прерывающей программы, принято называть уровнем прерывания.

Таким образом, система с аппаратным распознаванием причин прерывания может быть названа многоуровневой системой прерывания.

Простейший способ указания начальных адресов состоит в следующем. Каждому уровню присваивается номер и в памяти отводится ячейка, адрес которой, к примеру, равен номеру уровня. В такой ячейке памяти может храниться команда перехода к остальной части прерывающей программы, которая может находиться в любом месте памяти. Набор фиксированных по отношению к своим уровням ячеек памяти образует таблицу входов в прерывающие программы, содержание которых может менять программист.

Приоритетное обслуживание прерываний

Аппарат приоритетов предназначен для повышения эффективности использования всех ресурсов ВС. Так, неэффективно одновременное выполнение двух заданий, каждое из которых требует большой загрузки устройств ввода-вывода и незначительно использует центральный процессор. Приоритет задания может назначаться исходя из: времени его выполнения ("короткие" задания имеют более высокий приоритет по сравнению с "длинными" заданиями); объема используемой оперативной памяти (задания, требующие большого объема памяти, не должны иметь одинаковый приоритет); интенсивности и объема использования других ресурсов ВС; срочности выполнения и т.д.

При программном распознавании причин прерывания прерывающая программа начинает анализ с тех запросов на прерывание, которые имеют колее высокий приоритет. При этом в процессе работы мы можем программным путем изменить приоритет запросов. При аппаратном распознавании причин прерывания указанные функции возлагаются на специальное оборудование.

Понятие приоритета в прерывании программ имеет два смысла. Предположим, что никаких ограничений на время поступления запросов прерывания не накладывается. При одновременном поступлении нескольких запросов для немедленного удовлетворения может быть принят только один. Этот тип приоритета, определяющий очередность рассмотрения запросов прерывания, называют приоритетом между запросами прерываний. Прерывающие программы, однако, могут иметь относительно текущей программы различную степень важности, и не любым запросом может быть прервана выполняющаяся программа. Чтобы иметь возможность прервать текущую программу, принятый СПП запрос должен соответствовать программе более важной, нежели выполняемая в данный момент. Этот тип приоритета определяет старшинство программ и обычно его называют приоритетом между прерывающими программами.

В случае простейшей аппаратной реализации приоритета между запросами прерывания может быть использован метод "последовательного поиска".

Суть метода состоит в последовательном изменении содержимого счётчика от 0 до 2^n - 1 и просмотре всех 2^n уровней прерывания до совпадения содержимого счетчика с номером уровня.

При одновременном появлении нескольких запросов жестко закрепляйся запрос с уровня с меньшим номером. Метод "последовательного поиска" прост в реализации, но время реакции велико, так как в общем случае необходимо прохождение счетчиком всех 2^n позиций, что при большом n может выходить за допустимые временные пределы. Особенно это важно при работе

в режиме реального времени.

Приоритет между прерывающими программами определяет, какие программы могут прервать данную программу, а какие нет. Этот вид приоритета для многоуровневых систем с достаточной глубиной прерывания имеет гораздо большее значение, чем приоритет между запросами прерывания.

Таким образом, приоритет между запросами прерывания нужен лишь для выбора одного запроса из многих, а приоритет между прерывающими программами определяет фактический порядок выполнения программ.

Так как степень важности программ, их объем, требуемые ресурсы т. д. могут изменяться в ходе вычислительного процесса, то только приоритеты между запросами прерывания могут быть строго зафиксированы. Приоритеты же между прерывающими программами должны быть программно управляемыми.

Маска прерывания. Маска - шаблонная последовательность знаков, управляющая сохранением или исключением отдельных частей другой последовательности знаков. В простейшем виде маска - это двоичное число, каждый разряд которого соответствует одному из уровней прерывания и разрешает (например, состояние "1") или запрещает (состояние "0") прерывание запросов, относящихся к данному уровню. Управление приоритетом находятся полностью в распоряжении программы. Для каждой прерывающей программы может быть установлена своя маска, указывающая, какие программы способны ее прерывать. Каждый разряд маски соответствует отдельной программе. Маски всех программ хранятся в памяти. Если какая-нибудь программа вызывается для выполнения, то её маска засылается в регистр маски. Физически маска реализуется обычно в виде триггерного регистра, состояние которого можно изменить программным путем. При формировании его состояние "1" получают лишь те триггеры, которые соответствуют программам с более высоким, чем у данной программы, приоритетом.

Организация возврата к прерванной программе

Для осуществления возврата к прерванной программе необходимо полностью восстановить её начальное состояние. Информацию, которую следует сохранять при прерывании программы, можно разделить на основную (которая запоминается всегда) и дополнительную (необходимость запоминания второй зависит от содержания прерывающей программы).

В основную информацию можно включить:

• содержимое счетчика адреса команд, т. е. адрес первой невыполненной команды прерванной программы;

• триггер состояния системы: "рабочее" или состояние "ожидания";

• маска прерывания, устанавливаемая каждой новой программой;

• код прерывания - двоичное число, отдельное для каждого уровня объединяющего прерывание от нескольких источников, по которому прерывающая программа опознает конкретный источник прерывания.

Так как код прерывания обычно находится в общем для всех регистре, то, если предыдущий код полностью себя не исчерпал (а это почти всегда так), с приходом новой прерывающей программы его надо запомнить.

Указанная информация образует так называемое "слово состояния программы" (ССП)

(иногда его называют вектором состояния программы), которое хранится в некотором поле памяти компьютера. В момент прерывания старое ССП, относящееся к прерванной программе, заменяется ССП прерывающей программы, а в конце прерывания старое ССП восстанавливается прерывающей программой. Для ускорения процесса замены ССП эту процедуру выполняют обычно аппаратным путем.

Замечание. В период сохранения и восстановления ССП прерывания любого уровня запрещены.

К дополнительной информации относят содержимое:

- арифметических регистров;
- индексных регистров;
- прочих программно-доступных регистров, общих для всех программ, и т.п.

Сохранение дополнительной информации увеличивает время обслуживания. Поэтому программисту надо тщательно продумать, что из дополнительной информации следует запоминать в каждом конкретном случае. Более того, момент прерывания следует выбирать так (если это возможно), чтоб дополнительной информации для сохранения было как можно меньше.