**БИЛЕТ № 16**

1. Особенности задач, решаемых в ОС в системах массового распараллеливания.

В данной разделе мы рассмотрим основное программное обеспечение, необходимое для построения многопроцессорных систем. Вначале остановимся на тех отличиях, которые присущи ПО многопроцессорных систем по сравнению с ПО, используемых в многозадачной однопроцессорной среде. Данные отличия обусловлены двумя главными причинами: во-первых, архитектурными особенностями, присущими многопроцессорной системе, и, во-вторых, особыми проблемами, возникающими для параллельно выполняемых приложений. Следует подчеркнуть, что главная проблема ⎯ это параллельное выполнение приложений.

Многозадачная однопроцессорная система может эмулировать многопроцессорную среду, создавая многочисленные "виртуальные процессоры" (virtual processors) для пользователей. Примерами такой системы могут служить ОС Unix, OS/2, WINDOWS. Пользователь данной системы может организовать параллельное выполнение программ, при котором данные с выхода одной программы будут являться входными данными другой программы. Однако заметим, что параллельность выполнения задач в однопроцессорной системе чисто условная. В каждый момент времени, в течение кванта, выполняется какая-либо одна задача, при этом все активные ресурсы предоставляются только этой задаче.

В этом случае можно лишь абстрактно предположить одновременное выполнение программ на "виртуальных процессорах", хотя использование мультипрограммирования и позволяет реализовать в какой-то степени истинное совмещение (рис.3.13).



Рис.3.13. Структура многозадачной ОС в однопроцессорной системе

На уровне виртуальных процессоров имеются незначительные различия между многозадачной однопроцессорной средой и многопроцессорной системой. Однако наличие нескольких процессоров усложняет системное ПО, необходимое для такой системы.

Одной из архитектурных особенностей, которая может усложнить ПО многопроцессорных систем, является их неоднородность. Наличие функционально различных процессоров требует разнородного ПО. В данном случае "менеджер ресурсов" ОС СМР должен обеспечить динамическую диспетчеризацию поступающих задач, т.е. спланировать место их выполнения (рис.3.14).

Второй архитектурной особенностью, усложняющей ПО системы, является асимметрия памяти, т.е. различная степень доступа каждого процессора к общим участкам памяти системы, различный объем памяти у каждого процессора.

Как третью особенность архитектуры многопроцессорой системы, особенно влияющей на универсальность ПО, отметим различные способы конфигурации процессоров в системе.



Рис.3.14. Структура многозадачной ОС в многопроцессорной системе

Рассмотрим вторую причину различия многозадачного ПО однопроцессорных и многопроцессорных систем. Это проблемы, связанные с параллельно выполняемыми приложениями. Базовым исполнительным элементом системы является процесс, т.е. независимая часть программы, которая выполняется процессором и использует программные и аппаратные ресурсы системы. В многопроцессорной среде процесс может выполнятся параллельно с другими процессами, приостанавливаясь для взаимодействия с другими процессами.

Исходя из этого, параллельно выполняемые приложения можно определить как программы, имеющие более двух процессов, между которыми возможно взаимодействие.

Параллельные процессы могут быть указаны в программе явно и неявно. Для указания явного параллелизма пользователь должен обладать средствами программирования, которые позволяли бы ему указывать параллельные места в программе. При неявном параллелизме такие места выявляются компилятором. В этом случае компилятор сканирует программу с целью выявления процессов, допускающих распараллеливание, т.е. осуществляется поиск фрагментов программы, которые можно идентифицировать как параллельные подпроцессы.

Для параллельно выполняемых приложений большую роль играют вопросы синхронизации. Эта проблема может послужить причиной приостановки работы системы, если механизмы синхронизации неэффективны и алгоритмы, их использующие, недостаточно хорошо спланированы.

Во многих системах аппаратная поддержка механизма синхронизации отсутствует. Поэтому его поддержку осуществляет программное обеспечение. Введение аппаратно поддерживаемого механизма синхронизации значительно снижает программные затраты на его обеспечение.

Для упрощения проектирования сложных параллельно выполняемых приложений в настоящее время применяются специальные управляющие программные структуры (Program ⎯ Control Structures).

Первый тип такой структуры ⎯ это посылочно-основной тип (message-based). При такой организации вычисления представляются многочисленными однородными (гомогенными) процессами, которые выполняются независимо и взаимодействуют через посылки. Минимальный размер процесса зависит от системы. Второй тип управляющих структур ⎯ это процедурные (chore) структуры. В этой структуре вся программа разбивается на маленькие разделы. Процесс, который выполняется разделом программы, называется процедурой (chore). Важной характеристикой процедуры в процессе ее выполнения является ее завершенность и отсутствие взаимодействия между ней и другими процессами. При этом система избегает длительных ожиданий. Размер базовых процедур выбирается минимальным. Они имеют относительно малый объем вводимой исходной информации, и количество используемых объектов минимально. Поэтому при их взаимодействии не возникает задержек на ожидание информации от других процедур. Для своего решения одна процедура может потребовать выполнения малого числа добавочных процедур. Примером ОС, построенных с помощью подобных управляющих структур, является ВСС - 500, Plurilus.

В качестве примера рассмотрим часть ОС СМР, которая управляет обменом между главной памятью и жестким диском. Основная процедура может включать:

1. Дисковые команды, обеспечивающие передачу "страницы данных" из памяти на диск и обратно.
2. Команды проверки готовности памяти и диска к обмену данными.

В состав главной процедуры входят также дополнительные процедуры, реализующие действия на низком уровне.

Третий тип управляющих структур, которые называются " производящие системы" (production system), сейчас часто используется в "системах искусственного интеллекта". Необходимость выполнение действий в них выражается в форме причинно-следственной связи. Всякий раз, когда определяется булевское выражение, истинное следствие может быть выполнено. В отличие от процедурных структур программа не разбивается на подпрограммы.

В "производящих системах" присутствует 4 ступени действий:

1. Управление отбором утверждений, которые подлежат оценке;
2. Приказ на проверку отобранных утверждений;
3. Отбор подмножества действительных следствий, которые выполнились;
4. Приказ на выполнение отобранных следствий.

Отметим, что все три управляющие структуры допускают параллельное выполнение приложений. Достоинством использования таких структур является то, что программист разрабатывает отдельные участки программы, а всеобщий контроль осуществляется системой при их выполнении.

Высокая степень конкуренции процессов на захват ресурсов в многопроцессорных системах может увеличить сложность решения задач управления, особенно при наличии многих процессоров.

В однопроцессорной системе в многозадачном режиме всегда можно избежать конфликтов путем тщательного планирования процессов, а в случае конфликта ⎯ приостановить активность одного из процессов. Программное обеспечение в этом случае должно только эффективно отслеживать возможные ошибки. При этом программное обеспечение, поддерживаемое аппаратно, более надежно.

Поведение исполняемых процессов в многопроцессорных системах более сложное, чем в однопроцессорных. Хотя параллельные программы могут быть и не очень сложными для выполнения, тем не менее возникают некоторые проблемы. С помощью определенных усилий программист может добиться правильной работы параллельных программ, однако обеспечение правильности совместной работы больших программных комплексов является трудной задачей.

2) Проблема управления памятью.

Функции системы управления памятью

Чтобы обеспечить эффективный контроль использования памяти, ОС должна выполнять следующие функции:

• отображение адресного пространства процесса на конкретные области физической памяти;

• распределение памяти между конкурирующими процессами;

• контроль доступа к адресным пространствам процессов;

• выгрузка процессов (целиком или частично) во внешнюю память, когда в оперативной памяти недостаточно места;

• учет свободной и занятой памяти.

В следующих разделах лекции рассматривается ряд конкретных схем управления памятью. Каждая схема включает в себя определенную идеологию управления, а также алгоритмы и структуры данных и зависит от архитектурных особенностей используемой системы. Вначале будут рассмотрены простейшие схемы. Доминирующая на сегодня схема виртуальной памяти будет описана в последующих лекциях.

Простейшие схемы управления памятью

Первые ОС применяли очень простые методы управления памятью. Вначале каждый процесс пользователя должен был полностью поместиться в основной памяти, занимать непрерывную область памяти, а система принимала к обслуживанию дополнительные пользовательские процессы до тех пор, пока все они одновременно помещались в основной памяти. Затем появился "простой свопинг" (система по-прежнему размещает каждый процесс в основной памяти целиком, но иногда на основании некоторого критерия целиком сбрасывает образ некоторого процесса из основной памяти во внешнюю и заменяет его в основной памяти образом другого процесса). Такого рода схемы имеют не только историческую ценность. В настоящее время они применяются в учебных и научно-исследовательских модельных ОС, а также в ОС для встроенных (embedded) компьютеров.

3) Модель непротиворечивости - поэлементная непротиворечивости

Репликация данных

Важным вопросом для распределенных систем является репликация данных. Данные обычно реплицируются для повышения надежности и увеличения производительности. Одна из основных проблем при этом — сохранение непротиворечивости реплик. Говоря менее формально, это означает, что если в одну из копий вносятся изменения, то нам необходимо обеспечить, чтобы эти изменения были внесены и в другие копии, иначе реплики больше не будут одинаковыми.

Вопросы, связанные с репликацией

1. как должны расходиться обновления по копиям

2. поддержка непротиворечивости

3. проблема кеширования

Основные проблемы репликации

Первая проблема - обеспечение взаимного исключения при доступе к объекту. Существует два решения:

• объект защищает себя сам (к примеру, доступ к объекту совершается при помощи синхронизированных методов в Java - каждому клиенту будет создано по потоку и виртуальная Java-машина не даст воспользоваться методом доступа к ресурсу обоим потокам в один момент времени)

• система защищает объект (ОС сервера создает некоторый адаптер, который делает доступным объект, только одному клиенту в один момент времени)

Вторая проблема - поддержка непротиворечивости реплик. Здесь снова выделяют два пути решения:

• решение основанное на осведомленности объекта о том, что он был реплицирован. По сути, обязанность поддерживать непротиворечивость реплик перекладывается на сам объект. То есть в системе нет централизованного механизма поддержки непротиворечивости репликаций. Преимущество в том, что объект может реализовывать некоторые специфичные для него методы поддержки непротиворечивости.

• обязанность поддержки непротиворечивости накладывается на систему управления распределенной системой. Это упрощает создание реплицируемых объектов, но если для поддержки непротиворечивости нужны некоторые специфичные для объекта методы, это создает трудности.

**Модели непротиворечивости, ориентированные на данные**

По традиции непротиворечивость всегда обсуждается в контексте операций чтения и записи над совместно используемыми данными, доступными в распределенной памяти (разделяемой) или в файловой системе (распределенной).

Модель непротиворечивости (consistency model), по существу, представляет собой контракт между процессами и хранилищем данных. Он гласит, что если процессы согласны соблюдать некоторые правила, хранилище соглашается работать правильно. То есть, если процесс соблюдает некоторые правила, он может быть уверен, что данные которые он читает являются актуальными. Чем сложнее правила - тем сложнее их соблюдать процессу, но тем с большей вероятностью прочитанные данные действительно являются актуальными.

• поэлементная

Еще одна модель непротиворечивости, созданная для применения в критических областях, — поэлементная непротиворечивость (entry consistency). Как и оба варианта свободной непротиворечивости, она требует от программиста (или компилятора) вставки кода для захвата и освобождения в начале и конце критической области. Однако в отличие от свободной непротиворечивости, поэлементная непротиворечивость дополнительно требует, чтобы каждый отдельный элемент совместно используемых данных был ассоциирован с переменной синхронизации — блокировкой или барьером. Если необходимо, чтобы к элементам массива имелся независимый параллельный доступ, то различные элементы массива должны быть ассоциированы с различными блокировками. Когда происходит захват переменной синхронизации, непротиворечивыми становятся только те данные, которые ассоциированы с этой переменной синхронизации. Поэлементная непротиворечивость отличается от ленивой свободной непротиворечивости тем, что в последней отсутствует связь между совместно используемыми элементами данных и блокировками или барьерами, потому при захвате необходимые переменные определяются эмпирически.

4) Принципы управления процессами в многозадачной ОС.

*Процесс*:

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

***2.7.1. Классификация процессов***

По способу создания процессы делятся на:

1. Системные процессы. Они создаются при загрузке системы, имеют оверлейную или динамически-последовательную структуру программ.
2. 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:

1. Последовательные. Это процессы, которые выполняются друг за другом, т.е. когда закончится выполнение первого процесса, начинается второй и т.д.
2. Параллельные. Процессы, которые могут выполняться одновременно. *Параллельные процессы* могут быть:
3. Независимые, т.е. выполняют разные операции и не имеют общих частей.
4. 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.

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 различать процессы, поступающие в систему и требующие безусловного выполнения в соответствии с одной из дисциплин обслуживания, и 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.

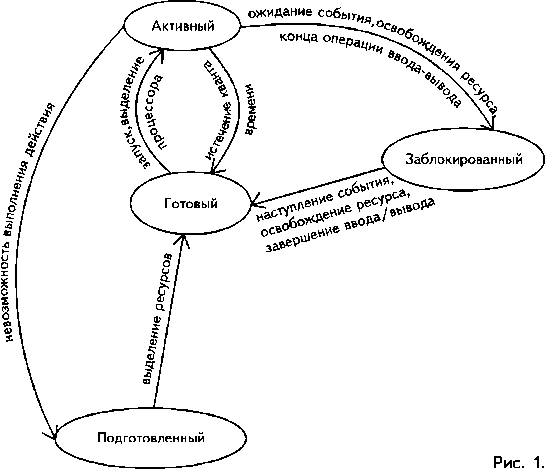


Рис. 2.10. Переход процесса из состояния в состояние

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 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 готовых 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 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 характеристики 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 (РСВ*** ⎯ ***process control block)***

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

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

1. Уникальный индентификатор 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 процесса (список разрешенных операций)
8. Связи зависимости в 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 РСВ 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 во 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 и восстановлению состояния процессов. Блоки управления системных процессов создаются при загрузке системы. Это необходимо, чтобы система выполняла 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 РСВ. 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 РСВ 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 РСВ имеет стандартную структуру, фиксированный размер, точку входа, содержит указанную выше информацию и дополнительную информацию для синхронизации процессов. Для синхронизации в РСВ имеются четыре поля:

1-2. Поля для организации цепочки связи.

3-4. Поля для организации цепочки ожидания.

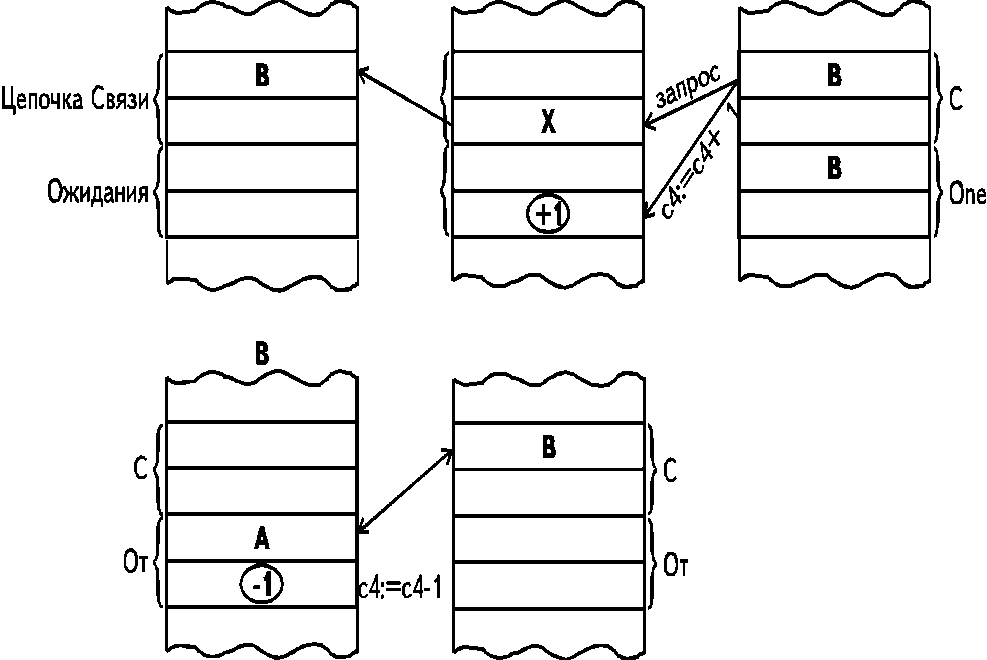
В цепочке связи указывается адрес РСВ вызываемого (поле 1) и вызывающего (поле 2) процесса.

В цепочке ожидания, в поле 3 указывается адрес РСВ вызываемого процесса, если вызываемый процесс занят. В поле 4 занятого процесса находится число процессов, которые ожидают данный.

Если процесс А пытается вызвать процесс В, а у процесса В в РСВ занята цепочка связей, то есть он является вызываемым по отношению к другим процессам, тогда адрес процесса В записывается в цепочке ожидания РСВ процесса А, а в поле счетчика ожидания РСВ 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 РСВ других процессов ищется первый (по приоритету или другим признакам) процесс, в поле 3 РСВ которого стоит имя ожидаемого процесса, в данном случае В, тогда управление передается этому процессу.

Пример: Имеем связь процессов Х и В, и процесс А вызвал процесс В (рис.2.11).

Процесс X Процесс В Процесс А



Процесс В Процесс А

Рис. 2.11. Состояние после выполнения процесса Х

***2.7.4. Операции над процессами***

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

1. **Создание** процесса 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.

1. **Уничтожение** процесса означает его удаление из системы. Ресурсы, выделенные этому процессу, возвращаются системе, имя в системных списках стирается и блок управления процессом освобождается.
2. **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.
3. **Запуск** (выбор) процесса, осуществляемый планировщиком, который выделяет процессу время процессора.
4. **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 быть 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 ВС при однопрограммном режиме работы выполняющийся процесс может приостановить сам себя 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.

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

5) Процесс и система прерывания. Особенности функционирования.

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

### *2.8.2. Обработка прерываний*

В общем случае под **прерыванием** понимают событие, требующее от системы прекращения работы активного процесса и перехода к обработке другого задания. Вычислительная система, обладающая возможностью использования прерываний, должна содержать целый комплекс программно-аппаратных средств, обеспечивающих регистрацию сигнала прерывания, приостановку активного процесса, переход к обработке прерывания и восстановление прерванного процесса. Совокупность действий, выполняемых после регистрации сигнала прерывания, называется **обработкой прерывания**. **Система обработки прерываний** ⎯ комплекс программно-аппаратных средств, обеспечивающих приостановку активного процесса и переход к обработке прерывания.

Рассмотрим работу системы прерываний подробней. В реальной жизни нам очень часто приходится сталкиваться с аналогами прерываний, реализованных в вычислительных системах. К примеру, предположим, что вы читаете книгу, и в это время раздается телефонный звонок. Тогда вы откладываете книгу, подходите к телефону, отвечаете на звонок, а затем возвращаетесь к прерванному на самом интересном месте занятию. В этот момент вам необходимо определить, с какого места продолжить чтение. Вернемся к тому моменту, когда вы отложили книгу. Если вы ожидали важного звонка, то вы могли закрыть книгу, даже не запомнив места, на котором остановились. В другом случае вы могли бы оставить пометку карандашом в том месте, где вас *прервал* телефонный звонок, а на нужной странице положить закладку. Но если книга вас весьма заинтересовала, то вы могли бы не сразу реагировать на звонок, а дочитать до конца предложения, абзаца, а то и до конца "самого интересного места".

Аналогичным образом работают прерывания в вычислительных системах. Когда процессор получает сигнал прерывания, он должен прервать выполняемый в нем процесс. При этом возможны несколько вариантов реакции процессора на сигнал прерывания:

1. Прерывание возможно только после определенных команд, при этом необходимо сохранение минимального количества информации о состоянии системы.
2. Прерывание возможно после завершения любой команды процессора.
3. Прерывание возможно после завершения очередного такта процессора. При этом требуется сохранение большого объема информации.

Большинство процессоров построены таким образом, что сигнал прерывания проверяется ими только после завершения выполнения текущей команды выполняющегося процесса. Однако, в системе могут существовать прерывания, для которых невозможно ожидание завершения очередной команды, например, если возникает ошибка в самой команде. Поэтому в системе могут использоваться одновременно несколько из перечисленных способов для обработки различных прерываний.

Обнаружив сигнал прерывания, процессор должен запомнить состояние прерванного процесса для того, чтобы продолжить его выполнение после обработки прерывания. После этого в процессор загружается новый процесс, выполняющий обработку прерывания. Эта процедура получила название **переключения контекста** или **переключения состояния**. Она является одной из важнейших в организации работы операционной системы, причем ее реализация требует поддержки и выполнения некоторых функций на аппаратном уровне.

Каждый процессор оснащен аппаратурой для управления последовательностью выполнения команд. Это специальный регистр, постоянно хранящий адрес следующей команды. Он называется **словом состояния программы** *(Program Status Word, PSW),* иначе называемый также **словом состояния процесса** *(Process Status Word, PSW),* **счетчиком программы** *(Рrogram Counter, PC),* **счетчиком адреса программы** *(Program Address Counter, PAC),* **указателем команды** *(Instruction Pointer, IP).* Как минимум, такой регистр должен содержать адрес следующей команды (или последовательности команд), а в дополнение к этому ⎯ различную информацию, которую система связывает с процессом.

При переключении контекста процессор сохраняет содержимое PSW, затем помещает в него новое значение, соответствующее процессу, называемому **обработчиком прерывания** *(Interrupt Handler, IH),* который и выполняет обработку самого прерывания. После завершения обработки прерывания в PSW восстанавливается содержимое сохраненного ранее PSW.

В процессе обработки прерывания можно выделить следующие **фазы прерывания** (рис. 2.20):



Рис. 2.20. Фазы прерывания

1. Тз — **время задержки** между моментом возникновения сигнала прерывания и прерыванием активного процесса. Оно зависит от принятого в системе (процессоре) способа обработки сигнала прерывания (смотри выше).
2. Тс — **время сохранения** необходимой информации. Зависит от количества сохраняемой информации при принятом способе обработки сигнала прерывания.
3. Тд — **время дешифрации** сигнала прерывания. Зависит от аппаратуры, дешифрирующей сигнал прерывания.
4. Тв — **время восстановления** прерванного процесса. Зависит от количества восстанавливаемой информации.

Время между возникновением сигнала прерывания и началом выполнения обработчика прерывания называется **временем реакции системы на сигнал прерывания** (Тр).

Для реализации механизма прерываний необходима аппаратная схема фиксации сигнала запроса на прерывание. Такая схема обычно содержит регистр, на котором фиксируется наличие сигналов во входных линиях **запросов на прерывания**. Объединенные схемой "ИЛИ", сигналы с разрядов регистра формируют общий сигнал о наличии запроса на прерывание. Затем все разряды регистра опрашиваются в порядке приоритетов входных линий. При этом используются 2 варианта реализации опроса:

1. Полноупорядоченная схема**.** Регистры опрашиваются по порядку приоритетов, от высшего к низшему. Это простая схема, однако при возникновении прерываний с низким приоритетом необходимо проверить *все* регистры запросов на прерывания с более высоким приоритетом.
2. Частично упорядоченная схема**.** Все прерывания делятся на **классы** и вводится 2-уровневая система приоритетов: первый уровень — среди различных классов, второй — внутри каждого класса. При этом, сначала по полноупорядоченной схеме определяется класс прерывания, на которое поступил запрос, а затем, также по полноупорядоченной схеме внутри установленного класса, определяется само прерывание. Это ускоряет поиск прерывания с низким приоритетом, однако усложняет процедуру поиска.

### *2.8.3. Классы прерываний*

По своему назначению, причине возникновения прерывания делятся на различные **классы**. Традиционно выделяют следующие классы:

1. *Прерывания от схем контроля машины*. Возникают при обнаружении сбоев в работе аппаратуры, например, при несовпадении четности в микросхемах памяти.
2. *Внешние прерывания*. Возбуждаются сигналами запросов на прерывание от различных внешних устройств: таймера, клавиатуры, другого процессора и пр.
3. *Прерывания по вводу/выводу*. Инициируются аппаратурой ввода/вывода при изменении состояния каналов ввода/вывода, а также при завершении операций ввода/вывода.
4. *Прерывания по обращению к супервизору*. Вызываются при выполнении процессором **команды обращения к супервизору** (вызов функции операционной системы). Обычно такая команда инициируется выполняемым процессом при необходимости получения дополнительных ресурсов либо при взаимодействии с устройствами ввода/вывода.
5. *Программные прерывания*. Возникают при выполнении **команды вызова прерывания** либо при обнаружении ошибки в выполняемой команде, например, при арифметическом переполнении.

В последнее время принято прерывания 4 и 5 классов объединять в один класс программных прерываний, причем, в зависимости от источника, вызвавшего прерывание, среди них выделяют такие подтипы:

1. прерывание вызванное исполнением процессором *команды* *перехода к подпрограмме обработки прерывания*
2. прерывания, возникающие в результате *исключительной* ( аварийной) *ситуации* в процессоре (деление на "0", переполнение и т.д.).

В связи с многообразием различных ВС и их постоянным развитием меняется и организация системы прерываний. Так, с появлением виртуальной памяти, появился класс страничных прерываний, который можно отнести и к классу исключительных ситуаций в процессоре; в системах с кэш-памятью существуют прерывания подкачки страниц в кэш-память и т.д.

Во время выполнения обработчика одного из прерываний возможно поступление другого сигнала прерывания, поэтому система должна использовать определенную *дисциплину обслуживания заявок* на прерывания. Обычно различным классам либо отдельным прерываниям присваиваются различные *абсолютные приоритеты*, т.о. при поступлении запроса с высшим приоритетом текущий обработчик снимается, а его место занимает обработчик вновь поступившего прерывания. Количество уровней вложенности прерываний называют **глубиной системы прерываний**. Обработчики прерываний либо дисциплина обслуживания заявок на прерывание должны иметь также специальные средства для случая, когда при обработке прерывания возникает запрос на обработку прерывания от того же источника.