**Лекция 7 Взаимодействие и планирование процессов**

**Цель занятия:** Сформировать знания студентов о взаимодействии и планировании процессов в операционных системах.

## Контекст и дескриптор процесса

На протяжении существования процесса его выполнение может быть многократно прервано и продолжено. Для того, чтобы возобновить выполнение процесса, необходимо восстановить состояние его операционной среды. Состояние операционной среды отображается состоянием регистров и программного счетчика, режимом работы процессора, указателями на открытые файлы, информацией о незавершенных операциях ввода-вывода, кодами ошибок выполняемых данным процессом системных вызовов и т.д. Эта информация называется *контекстом процесса*.

Кроме этого, операционной системе для реализации планирования процессов требуется дополнительная информация: идентификатор процесса, состояние процесса, данные о степени привилегированности процесса, место нахождения кодового сегмента и другая информация. В некоторых ОС (например, в ОС UNIX) информацию такого рода, используемую ОС для планирования процессов, называют *дескриптором процесса*.

Дескриптор процесса по сравнению с контекстом содержит более оперативную информацию, которая должна быть легко доступна подсистеме планирования процессов. Контекст процесса содержит менее актуальную информацию и используется операционной системой только после того, как принято решение о возобновлении прерванного процесса.

Очереди процессов представляют собой дескрипторы отдельных процессов, объединенные в списки. Таким образом, каждый дескриптор, кроме всего прочего, содержит по крайней мере один указатель на другой дескриптор, соседствующий с ним в очереди. Такая организация очередей позволяет легко их переупорядочивать, включать и исключать процессы, переводить процессы из одного состояния в другое.

Программный код только тогда начнет выполняться, когда для него операционной системой будет создан процесс. Создать процесс - это значит:

1.      создать информационные структуры, описывающие данный процесс, то есть его дескриптор и контекст;

2.      включить дескриптор нового процесса в очередь готовых процессов;

3.      загрузить кодовый сегмент процесса в оперативную память или в область свопинга.

## Алгоритмы планирования процессов

В однозадачной ОС используется группа алгоритмов «без переключения». К ним относятся алгоритмы: FIFO  (первый пришел, первый обслужен), SJF (кратчайшее задание - первым), HRN (модификация алгоритма SJF, учитывающая длительность ожидания задания в очереди).

Планирование процессов в мультипрограммных ОС включает в себя решение следующих задач:

1.      определение момента времени для смены выполняемого процесса;

2.      выбор процесса на выполнение из очереди готовых процессов;

3.      переключение контекстов "старого" и "нового" процессов.

Первые две задачи решаются программными средствами, а последняя в значительной степени аппаратно.

Существует множество различных алгоритмов планирования процессов, по разному решающих вышеперечисленные задачи, преследующих различные цели и обеспечивающих различное качество мультипрограммирования. Среди этого множества алгоритмов рассмотрим подробнее две группы наиболее часто встречающихся алгоритмов: алгоритмы, основанные на квантовании, и алгоритмы, основанные на приоритетах.

В соответствии с алгоритмами, основанными на квантовании, смена активного процесса происходит, если:

·         процесс завершился и покинул систему,

·         произошла ошибка,

·         процесс перешел в состояние ОЖИДАНИЕ,

·         исчерпан квант процессорного времени, отведенный данному процессу.

Процесс, который исчерпал свой квант, переводится в состояние ГОТОВНОСТЬ и ожидает, когда ему будет предоставлен новый квант процессорного времени, а на выполнение в соответствии с определенным правилом выбирается новый процесс из очереди готовых. Таким образом, ни один процесс не занимает процессор надолго, поэтому квантование широко используется в системах разделения времени. Граф состояний процесса, изображенный на рисунке 2.1, соответствует алгоритму планирования, основанному на квантовании.

Кванты, выделяемые процессам, могут быть одинаковыми для всех процессов или различными. Кванты, выделяемые одному процессу, могут быть фиксированной величины или изменяться в разные периоды жизни процесса. Процессы, которые не полностью использовали выделенный им квант (например, из-за ухода на выполнение операций ввода-вывода), могут получить или не получить компенсацию в виде привилегий при последующем обслуживании. По разному может быть организована очередь готовых процессов: циклически, по правилу "первый пришел - первый обслужился" (FIFO) или по правилу "последний пришел - первый обслужился" (LIFO).

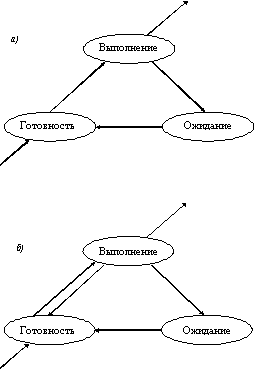
Для обеспечения более «справедливого» распределения времени процессора используются многоуровневые очереди с обратными связями. Новый процесс входит в сеть очередей с конца верхней очереди. Он перемещается по этой очереди, реализующей принцип ­FIFO, пока не получит в свое распоряжение ЦП. Если задание завершается или освобождает ЦП, чтобы подождать завершения операции ввода-вывода или наступления некоторого события, то оно, это задание, выходит из сети очередей. Если выделенные квант времени истекает до того, как процесс добровольно освободит ЦП, этот процесс помещается в конец следующей очереди более  низкого уровня. Этот  процесс в следующий раз получит в свое распоряжение ЦП, когда он достигнет начала данной очереди, если­ при этом не будет ожидающих в первой очереди. Если данный процесс будет продолжать использовать полный квант времени, предоставляемый на каждом уровне, он продолжит переход в конец очереди следующего нижележащего уровня. Обычно в системе предусматривается некоторая  очередь самого нижнего уровня, которая реализует принцип  циклического обслуживания и в которой данный процесс циркулирует до тех пор, пока не завершится.Во многих структурах многоуровневых очередей с обратными связями квант времени, предоставляемый данному процессу при переходе в каждую очередь более низкого уровня, увеличивается.

Другая группа алгоритмов использует понятие "приоритет" процесса. Приоритет - это число, характеризующее степень привилегированности процесса при использовании ресурсов вычислительной машины, в частности, процессорного времени: чем выше приоритет, тем выше привилегии.

Приоритет может выражаться целыми или дробными, положительным или отрицательным значением. Чем выше привилегии процесса, тем меньше времени он будет проводить в очередях. Приоритет может назначаться директивно администратором системы в зависимости от важности работы или внесенной платы, либо вычисляться самой ОС по определенным правилам, он может оставаться фиксированным на протяжении всей жизни процесса либо изменяться во времени в соответствии с некоторым законом. В последнем случае приоритеты называются динамическими.

Существует две разновидности приоритетных алгоритмов: алгоритмы, использующие относительные приоритеты, и алгоритмы, использующие абсолютные приоритеты.

В обоих случаях выбор процесса на выполнение из очереди готовых осуществляется одинаково: выбирается процесс, имеющий наивысший приоритет. По разному решается проблема определения момента смены активного процесса. В системах с относительными приоритетами активный процесс выполняется до тех пор, пока он сам не покинет процессор, перейдя в состояние ОЖИДАНИЕ (или же произойдет ошибка, или процесс завершится). В системах с абсолютными приоритетами выполнение активного процесса прерывается еще при одном условии: если в очереди готовых процессов появился процесс, приоритет которого выше приоритета активного процесса. В этом случае прерванный процесс переходит в состояние готовности. На рисунке 4.2 показаны графы состояний процесса для алгоритмов с относительными (а) и абсолютными (б) приоритетами.



*Рисунок 4.2 - Графы состояний процессов в системах  
(а) с относительными приоритетами; (б)с абсолютными приоритетами*

Во многих операционных системах алгоритмы планирования построены с использованием как квантования, так и приоритетов. Например, в основе планирования лежит квантование, но величина кванта и/или порядок выбора процесса из очереди готовых определяется приоритетами процессов.

### Вытесняющие и невытесняющие алгоритмы планирования

Существует два основных типа процедур планирования процессов - вытесняющие (preemptive) и невытесняющие (non-preemptive).

Non-preemptive multitasking - невытесняющая многозадачность - это способ планирования процессов, при котором активный процесс выполняется до тех пор, пока он сам, по собственной инициативе, не отдаст управление планировщику операционной системы для того, чтобы тот выбрал из очереди другой, готовый к выполнению процесс.

Preemptive multitasking - вытесняющая многозадачность - это такой способ, при котором решение о переключении процессора с выполнения одного процесса на выполнение другого процесса принимается планировщиком операционной системы, а не самой активной задачей.

Понятия preemptive и non-preemptive иногда отождествляются с понятиями приоритетных и бесприоритетных дисциплин, что совершенно неверно, а также с понятиями абсолютных и относительных приоритетов, что неверно отчасти. Вытесняющая и невытесняющая многозадачность - это более широкие понятия, чем типы приоритетности. Приоритеты задач могут как использоваться, так и не использоваться и при вытесняющих, и при невытесняющих способах планирования. Так в случае использования приоритетов дисциплина относительных приоритетов может быть отнесена к классу систем с невытесняющей многозадачностью, а дисциплина абсолютных приоритетов - к классу систем с вытесняющей многозадачностью. А бесприоритетная дисциплина планирования, основанная на выделении равных квантов времени для всех задач, относится к вытесняющим алгоритмам.

Основным различием между preemptive и non-preemptive вариантами многозадачности является степень централизации механизма планирования задач. При вытесняющей многозадачности механизм планирования задач целиком сосредоточен в операционной системе, и программист пишет свое приложение, не заботясь о том, что оно будет выполняться параллельно с другими задачами. При этом операционная система выполняет следующие функции: определяет момент снятия с выполнения активной задачи, запоминает ее контекст, выбирает из очереди готовых задач следующую и запускает ее на выполнение, загружая ее контекст.

При невытесняющей многозадачности механизм планирования распределен между системой и прикладными программами. Прикладная программа, получив управление от операционной системы, сама определяет момент завершения своей очередной итерации и передает управление ОС с помощью какого-либо системного вызова, а ОС формирует очереди задач и выбирает в соответствии с некоторым алгоритмом (например, с учетом приоритетов) следующую задачу на выполнение. Такой механизм создает проблемы как для пользователей, так и для разработчиков.

Для пользователей это означает, что управление системой теряется на произвольный период времени, который определяется приложением (а не пользователем). Если приложение тратит слишком много времени на выполнение какой-либо работы, например, на форматирование диска, пользователь не может переключиться с этой задачи на другую задачу, например, на текстовый редактор, в то время как форматирование продолжалось бы в фоновом режиме. Эта ситуация нежелательна, так как пользователи обычно не хотят долго ждать, когда машина завершит свою задачу.

Поэтому разработчики приложений для non-preemptive операционной среды, возлагая на себя функции планировщика, должны создавать приложения так, чтобы они выполняли свои задачи небольшими частями. Например, программа форматирования может отформатировать одну дорожку дискеты и вернуть управление системе. После выполнения других задач система возвратит управление программе форматирования, чтобы та отформатировала следующую дорожку. Подобный метод разделения времени между задачами работает, но он существенно затрудняет разработку программ и предъявляет повышенные требования к квалификации программиста. Программист должен обеспечить "дружественное" отношение своей программы к другим выполняемым одновременно с ней программам, достаточно часто отдавая им управление. Крайним проявлением "недружественности" приложения является его зависание, которое приводит к общему краху системы. В системах с вытесняющей многозадачностью такие ситуации, как правило, исключены, так как центральный планирующий механизм снимет зависшую задачу с выполнения.

Однако распределение функций планировщика между системой и приложениями не всегда является недостатком, а при определенных условиях может быть и преимуществом, потому что дает возможность разработчику приложений самому проектировать алгоритм планирования, наиболее подходящий для данного фиксированного набора задач. Так как разработчик сам определяет в программе момент времени отдачи управления, то при этом исключаются нерациональные прерывания программ в "неудобные" для них моменты времени. Кроме того, легко разрешаются проблемы совместного использования данных: задача во время каждой итерации использует их монопольно и уверена, что на протяжении этого периода никто другой не изменит эти данные. Существенным преимуществом non-preemptive систем является более высокая скорость переключения с задачи на задачу.

Примером эффективного использования невытесняющей многозадачности является файл-сервер NetWare, в котором, в значительной степени благодаря этому, достигнута высокая скорость выполнения файловых операций. Менее удачным оказалось использование невытесняющей многозадачности в операционной среде Windows 3.х.

Однако почти во всех современных операционных системах, ориентированных на высокопроизводительное выполнение приложений (UNIX, Windows NT, OS/2, VAX/VMS), реализована вытесняющая многозадачность. В последнее время дошла очередь и до ОС класса настольных систем, например, OS/2 Warp и Windows. Возможно в связи с этим вытесняющую многозадачность часто называют истинной многозадачностью.

### Взаимодействие и синхронизация процессов и потоков

В мультипрограммных однопроцессорных системах процессы чередуются, обеспечивая эффективное выполнение программ. В многопроцессорных системах возможно не только чередование, но и перекрытие процессов. Обе эти технологии, которые можно рассматривать как примеры параллельных вычислений, порождают одинаковые проблемы. Выполнение процессов и потоков в мультипрограммной среде всегда имеет асинхронный характер – невозможно предсказать относительную скорость выполнения процессов. Момент прерывания потоков, время нахождения их в очередях к разделяемым ресурсам, порядок выбора потоков для выполнения – все эти события являются результатом стечения многих обстоятельств и являются случайными, это справедливо как по отношению к потокам одного процесса, выполняющим общий программный код, так и по отношению к потокам разных процессов, каждый из которых выполняет собственную программу.

Способы взаимодействия процессов (потоков) можно классифицировать по степени осведомленности одного процесса о существовании другого

1. Процессы *не осведомлены* о наличии друг друга (например, процессы разных заданий одного или различных пользователей). Это независимые процессы, не предназначенные для совместной работы. Хотя эти процессы и не работают совместно, ОС должна решать вопросы конкурентного использования ресурсов. Например, два независимых приложения могут затребовать доступ к одному и тому же диску или принтеру. ОС должна регулировать такие обращения.
2. Процессы *косвенно осведомлены* о наличии друг друга (например, процессы одного задания). Эти процессы не обязательно должны быть осведомлены о наличии друг друга с точностью до идентификатора процесса, однако они разделяют доступ к некоторому объекту, например, буферу ввода-вывода, файлу или БД. Такие процессы демонстрируют сотрудничество при разделении общего объекта.
3. Процессы *непосредственно осведомлены* о наличии друг друга (например, процессы, работающие последовательно или поочередно в рамках одного задания). Такие процессы способны общаться один с другим с использованием идентификаторов процессов и изначально созданы для совместной работы. Эти процессы также демонстрируют сотрудничество при работе.

Таким образом, потенциальные проблемы, связанные с взаимодействием и синхронизацией процессов и потоков, могут быть представлены следующей таблицей.

|  |  |  |  |
| --- | --- | --- | --- |
| **Степень осведомленности** | **Взаимосвязь** | **Влияние одного процесса на другой** | **Потенциальные проблемы** |
| Процессы не осведомлены друг о друге | Конкуренция | * Результат работы одного процесса не зависит от действий других. * Возможно влияние одного процесса на время работы другого | * Взаимоисключения * *Взаимоблокировки* * Голодание |
| Процессы косвенно осведомлены о наличии друг друга | Сотрудничество с использованием разделения | * Результат работы одного процесса может зависеть от информации, полученной от других. * Возможно влияние одного процесса на время работы другого | * Взаимоисключения * *Взаимоблокировки* * Голодание * Синхронизация |
| Процессы непосредственно осведомлены о наличии друг друга | Сотрудничество с использованием связи | * Результат работы одного процесса зависит от информации, полученной от других процессов. * Возможно влияние одного процесса на время работы другого | * *Взаимоблокировки*(расходуемые ресурсы) * Голодание |

**Контрольные вопросы:**

* 1. *Что такое прерывания в ОС?*
  2. *Что такое параллельные процессы в ОС?*
  3. *Что такое семафоры в ОС?*
  4. *Что представляют собой тупики в ОС?*
  5. *Что такое взаимодействие и синхронизация процессов и потоков? Как они выполняются?*