Параллельное программирование

Процессы и потоки

9.03.2016

Процессы и потоки

Под процессом понимается некоторая часть (единица) работы, создаваемая операционной системой.

Надо сказать, что программа и процесс – не обязательно эквивалентные понятия. Программа может состоять из нескольких процессов, а в некоторых ситуациях процесс может быть не связан с конкретной программой. Как говорят, процессы – артефакты ОС, программы – артефакты разработчика.

Потоки позволяют одной программе состоять из параллельно выполняемых частей, причём все части имеют доступ к одним и тем же переменным, постоянным и адресному пространству в целом.

Для потоков требуется меньший объём затрат системных ресурсов. Потоки иногда можно рассматривать как облегчённые процессы в том смысле, что они позволяют воспользоваться многими преимуществами процессов без больших затрат на организацию взаимодействия между ними.

Под потоком подразумевается часть выполняемого кода в операционной системе (вообще говоря в процессе), которая может быть регламентирована определённым образом.

Чтобы некоторую часть работы можно было назвать *процессом*, она должна иметь *адресное пространство*, назначаемое операционной системой, и идентификатор или *идентификационный номер* процесса (ID).

Процесс должен обладать определённым статусом, иметь свой элемент в таблице процессов. Процесс может содержать один или несколько потоков, выполняющихся в рамках его адресного пространства, использовать системные ресурсы, требуемые для этих потоков.

Каждый процесс имеет *основной (или первичный) поток*, под которым понимают программный поток управления. Процесс состоит из множества выполняющихся инструкций, размещённых в адресном пространстве этого процесса. Каждый поток, имея собственную последовательность инструкций, выполняется независимо от других, а все они параллельны друг другу.

Процесс с несколькими потоками называется многопоточным.

Все потоки одного процесса существуют в одном и том же адресном пространстве, и все ресурсы, принадлежащие процессу, разделяются между потоками. Сами потоки не владеют никакими ресурсами.

Адресное пространство процесса распределяется между инструкциями, данными, принадлежащими процессу, и стеками, обеспечивающими вызовы функций и хранение локальных переменных.

Процессы делят на две большие группы: пользовательские и системные.

Процессы, которые выполняют системный код, называют системными. Они применяются в системе в целом.

Коротко перечислим задачи системных процессов:

- Распределение оперативной памяти
- Обмен страницами между внутренними и вспомогательными запоминающими устройствами
- Контроль устройств

и многие другие...

Системные процессы могут выполнять некоторые задачи по поручению пользовательских процессов. Например, выделять оперативную память, делать запросы на ввод-вывод и т. п.

Пользовательские процессы при выполнение своего собственного кода иногда обращаются к системным функциям. При этом, выполняя свой собственный код, пользовательский процесс пребывает в пользовательском режиме, в котором он не может выполнять определённые привилегированные машинные команды.

При вызове системных функций пользовательский процесс выполняет инструкции операционной системы и удерживает процессор, пока не будет выполнен системный вызов, для выполнения которого процесс обращается к ядру операционной системы. В это время о пользовательском процессе говорят, что он находится *в привилегированном* режиме или *режиме ядра* и не может быть выгружен никаким другим процессом.

Блок управления процессами (PCB – process control block)

Процессы имеют характеристики, используемые для их идентификации и определения их поведения. Ядро операционной системы поддерживает необходимые структуры данных и предоставляет системные функции, при помощи которых пользователь может получить доступ к этой информации.

Данные PCB описывают процесс с точки зрения потребностей операционной системы. С

помощью этой информации система может управлять любым процессом, и, при переключении с одного процесса на другой, она сохраняет текущее состояние выполняющего процесса и его контекст в области сохранения PCB, чтобы можно было возобновить выполнение этого процесса в следующий раз, когда ему будет выделен центральный процессор.

Блок PCB содержит следующую информацию:

1. Текущее состояние и приоритет процесса
2. ID процесса, а также ID родительского и сыновнего процессов
3. Указатели на выделенные ресурсы
4. Указатели на область памяти процесса
5. Указатели на родительский и сыновний процессы
6. Процессор, занятый процессом
7. Регистры управления и состояния
8. Стековые указатели

Также PCB содержит информацию о *межпроцессорном взаимодействии* (IPC – inter-process communication), то есть о сообщениях и сигналах, которыми обмениваются процессы.

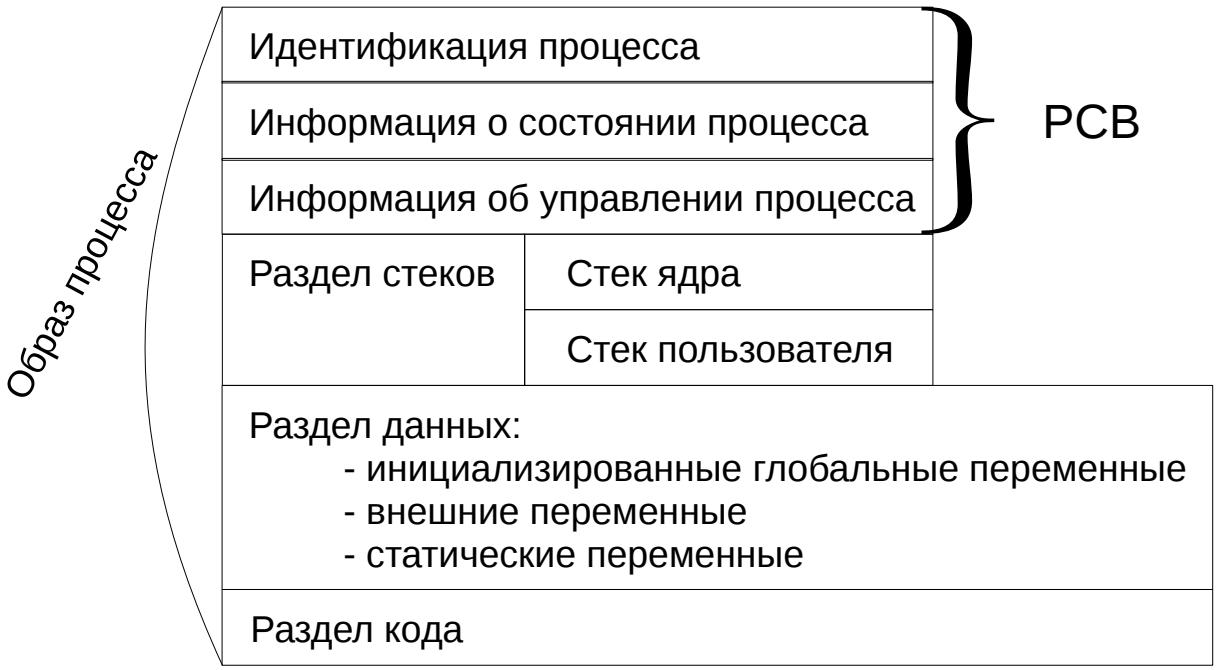
Таким образом операционная система может координировать параллельно выполняющиеся процессы.

При выполнении процесса стековые указатели, содержимое регистров пользователя, информация о состоянии процессора размещена в регистрах центрального процессора. При переключении операционной системы с одного процесса на другой вся информация из этих регистров сохраняется и может быть восстановлена, когда процесс снова получает центральный процессор во временное пользование.

Адресное пространство процесса имеет три логических раздела:

1. Текстовый – предназначен для кода программы
2. Информационный – в нём хранятся данные программы
3. Стековый – для стеков

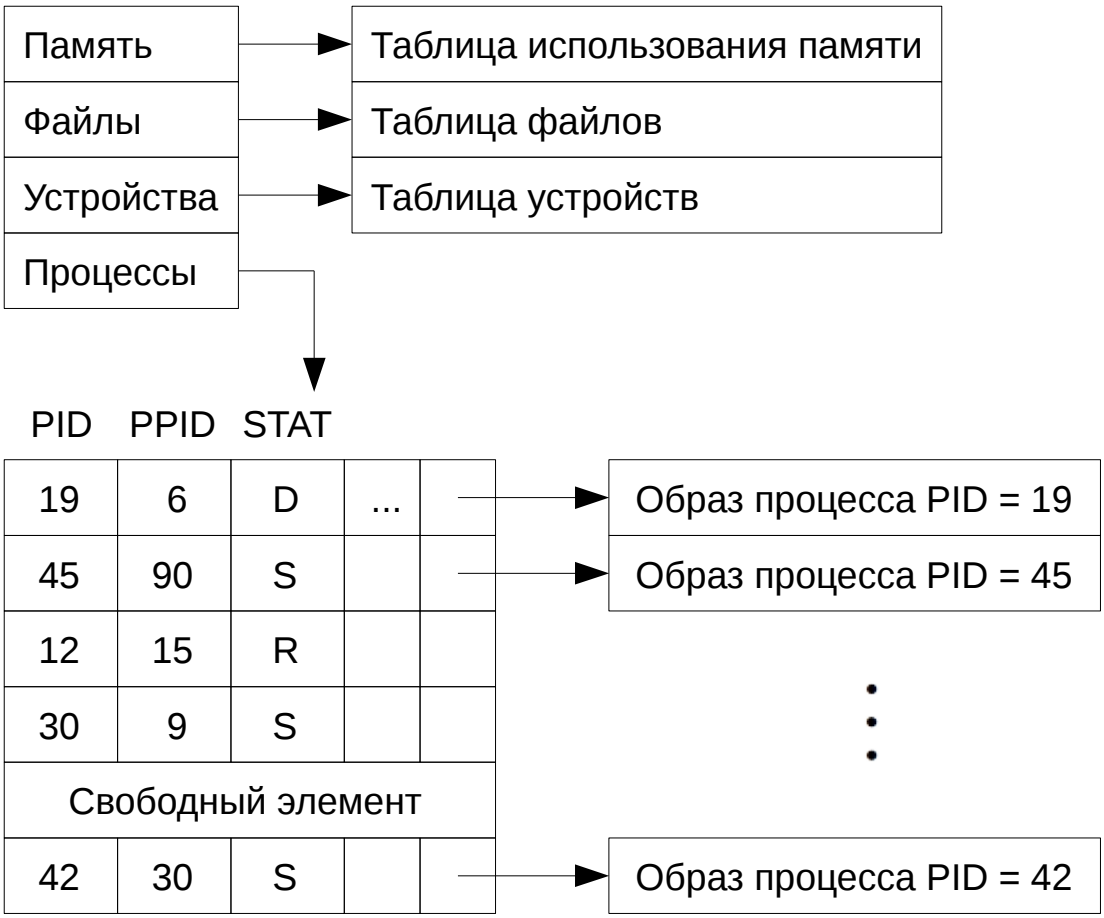
Схема:



Отметим, что адресное пространство процесса виртуально, разделы виртуального адресного пространства процесса представляют собой смежные блоки памяти, и каждый такой раздел и физическое адресное пространство разделены на участки памяти, именуемые страницами. У каждой страницы есть уникальный *номер страничного блока*.

Несмотря на то, что виртуальное адресное пространство каждого процесса защищено, то есть приняты меры по предотвращению доступа к нему других процессов, текстовый раздел процесса может использоваться совместно несколькими процессами. Два процесса могут также разделять один и тот же программный код.

Чтобы управлять всеми процессами, хранимыми во внутренней памяти, операционная система создаёт и поддерживает *таблицы процессов*. Вообще говоря, операционная система содержит отдельные таблицы для всех объектов, которыми она управляет (для ресурсов, для файлов, для оперативной памяти имеются свои отдельные таблицы).

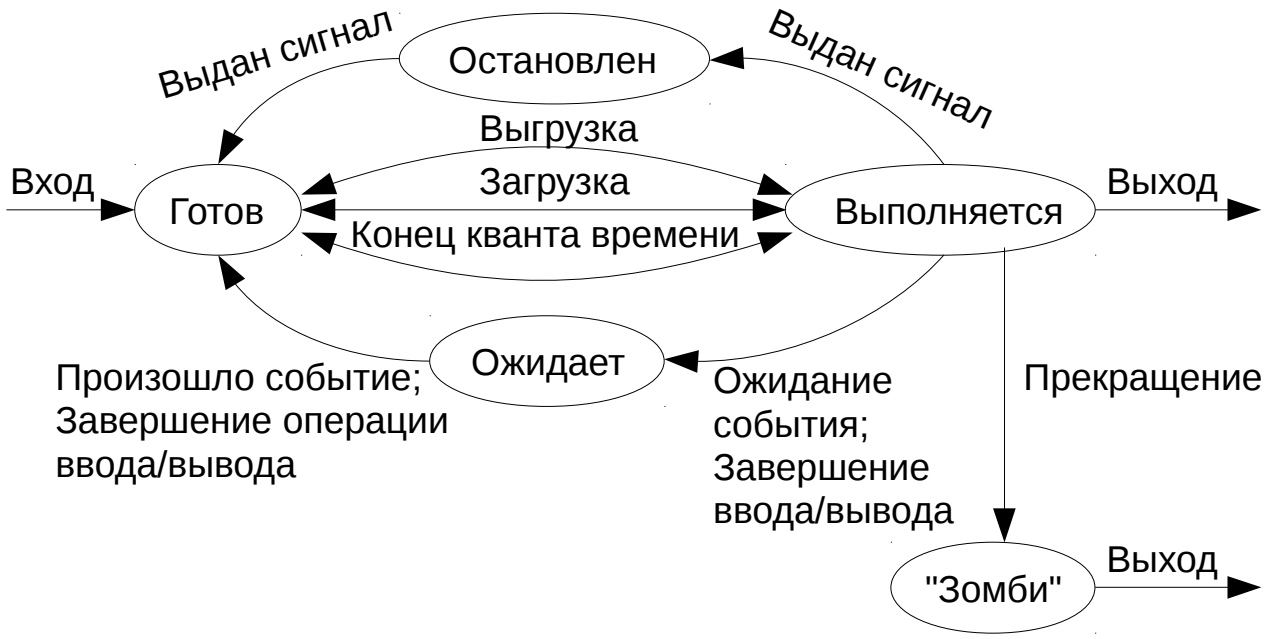


PID = ID (process identifier), PPID – идентификатор родительского процесса (parent process identifier), STAT – статус.

Состояния процессов

Процесс во время выполнения изменяет свои состояния и может находится в одном и следующих:

1. Состояние выполнения
2. Состояние готовности (работоспособности
3. Состояние зомби
4. Состояние ожидания (блокировки)
5. Состояние останова



16.03.2016

Переходы процессов из одного состояния в другое:

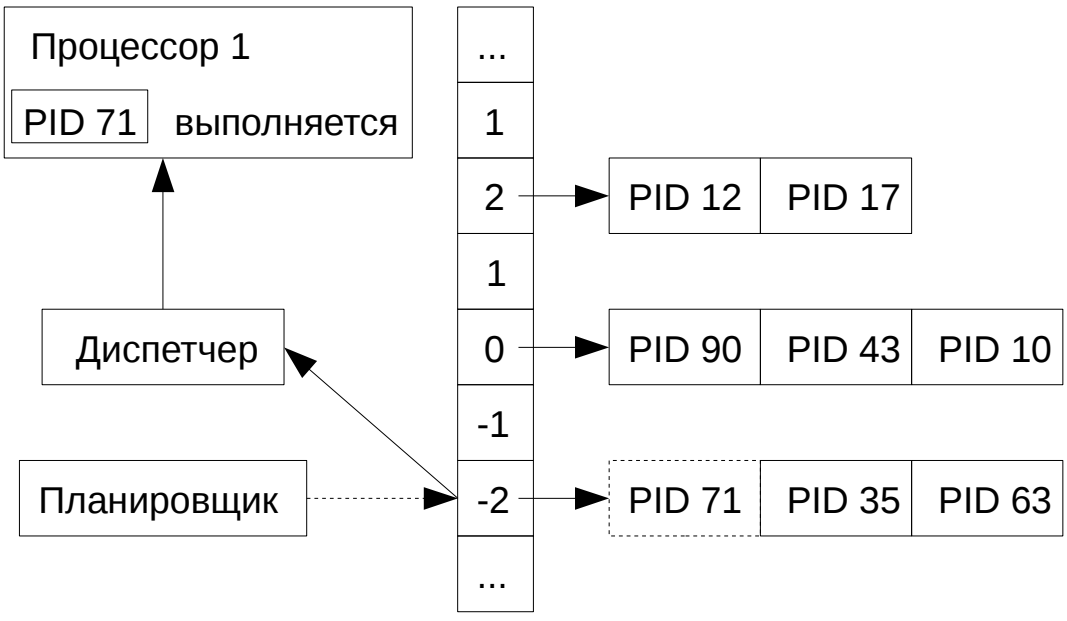
Переходы между состояниями	Описание переходов
Готовый → выполняется (загрузка)	Процесс назначается процессору.
Выполнение → готовый (конец кванта времени)	Квант времени истёк, процесс возвращается в очередь готовых процессов.
Выполнение → готовый (досрочная выгрузка)	Процесс выгружается до истечения его кванта времени (это возможно, если стал готов процесс с более высоким приоритетом). Выгруженный процесс помещается в очередь готовых процессов.
Выполнение → ожидание (блокировка)	Процесс отказался от процессора до истечения кванта времени. (Возможно, ему надо подождать наступления некоторого события, или он вызывает системную функцию, например ввода/вывода). Процесс помещается в очередь ждущих процессов.
Ожидание → готовый (разблокировка)	Событие, которое ожидал процесс, произошло.
Выполнение → остановка	Процесс отказался от процессора из-за получения им сигнала останова.
Остановлен → готов	Процесс получил сигнал продолжить работу и возвращается в очередь готовых процессов.
Выполнение → зомби	Процесс прекращён и ожидает, пока родительский процесс не извлечёт из таблицы процессов статус его завершения.

Зомби → выход	Родительский процесс извлекает из таблицы процессов статус завершения, и процесс "зомби" покидает систему.
Выполнение → выход	Процесс завершён, но он покидает систему после того, как родительский процесс извлечёт из таблицы процессов его статус завершения.

Планирование процессов

Если количество готовых к выполнению процессов более одного, то планировщик системы должен определить, какой из процессов первым назначить процессору. Для этого планировщик поддерживает структуры данных, которые позволяют наиболее эффективно распределять процессорное время между процессами. Каждый процесс получает класс или тип приоритета и размещается в соответствующей очереди вместе с другими работоспособными процессами того же приоритетного класса. Поэтому существует несколько приоритетных очередей. Эти приоритетные очереди упорядочиваются и помещаются в массив распределения, называемый многоуровневой приоритетной очередью.

Для выполнения процессором планировщик назначает тот процесс, который находится в головной части непустой очереди, имеющей самый высокий приоритет.



Приоритеты в системе делятся на *динамические* и *статические*. Если установленный приоритет является статическим, то изменить его нельзя. А динамический приоритет изменить можно.

Процессы с самым высоким приоритетом могут монополизировать процессор.

Если приоритет процесса динамический, то начальное значение приоритета может быть

заменено более высоким, и в результате процесс будет переведён в очередь с более высоким приоритетом.

В то же время процесс, который монополизирует процессор, может получить более низкий приоритет, или в системе могут появиться другие процессы, у которых приоритет выше, чем у процесса-монополиста.

В операционной системе Unix/Linux диапазон приоритетов изменяется от -20 до 19, и чем выше значение уровня, тем ниже приоритет процесса.

При назначении приоритета пользовательскому процессу необходимо учитывать, на какие действия процесс тратит времени больше всего. Одни интенсивно используют процессорное время, а другие большую часть времени ожидают выполнения операций ввода/вывода или наступления каких-либо событий. Если такой процесс готов к исполнению, то ему надо немедленно предоставить процессор, чтобы он мог сделать следующий запрос устройствам ввода/вывода.

Процессы, которые взаимодействуют между собой, могут требовать довольно высокий приоритет, чтобы рассчитывать на приличное время реакции.

Системные процессы имеют более высокий приоритет, чем пользовательские.

Стратегии планирования

Процессы размещаются в приоритетных очередях в соответствии со стратегией планирования. В Unix/Linux системах в основном используются две стратегии:

1. FIFO (rirst in first out)
2. RR (round-robin – каруселька)

Каждый процесс также имеет так называемый фактор уступчивости, который используют для вычисления уровня приоритета вызывающего процесса. Процесс наследует приоритет процесса, который его создал, и чтобы понизить приоритет процесса, надо увеличить его фактор уступчивости. Только процессы привилегированных пользователей и ядра системы могут увеличивать уровни своих приоритетов. Чем ниже фактор уступчивости, тем выше уровень приоритета процесса.

Переключение контекста

Переключение контекста происходит в тот момент, когда процессор переключается с одного процесса на другой. При этом операционная система сохраняет контекст текущего процесса и

восстанавливает контекст следующего процесса, выбранного для выполнения.

PCB блок прерванного процесса при этом обновляется, и изменяется значение поля состояния, то есть признак состояния выполнения заменяется признаком другого состояния. Сохраняется и обновляется содержимое регистров процессора, состояние стека, данные об идентификации, о стратегии планирования и учётная информация.

Операционная система должна отслеживать статус устройств ввода/вывода, процесса и других ресурсов, а также состояние всех структур данных, связанных с управлением памятью.

Выгруженный (прерванный) процесс помещается в соответствующую очередь.

Переключение контекста происходит в следующих случаях:

1. Процесс выгружается
2. Процесс добровольно отказывается от процессора
3. Процесс делает запрос устройствам ввода/вывода или должен ждать наступления события
4. Процесс переходит из пользовательского режима в режим ядра

Когда выгруженный процесс снова выбирается для выполнения процессором, его контекст восстанавливается и выполнение продолжается с точки, на которой он был прерван в предыдущем сеансе.

23.03.2016

Создание процесса

Для того, чтобы выполнять программы, операционная система создаёт процессы, при этом для каждого процесса создаётся и инициализируется новый PCB-блок, в раздел его идентификации записывается уникальный ID процесса и ID родительского процесса.

Программный счётчик (счётчик команд) устанавливается на входную точку программы, а указатели системных стеков устанавливаются таким образом, чтобы определять стековые границы процесса.

Если процессу не присвоено значение приоритета, то оно задаётся самым низким. Изначально процесс не обладает никакими ресурсами, если нет явного запроса на ресурсы или если они не были унаследованы от родительского процесса.

Процесс входит в состояние выполнения и помещается в очередь готовых к выполнению процессов. Для него выделяется адресное пространство, размер которого определяется по умолчанию на основе типа процесса. Кроме того, размер может быть установлен

родительским процессом, который может передать системе размер адресного пространства в момент создания процесса.

Родительские и сыновние процессы

В системе Linux есть процесс, который называется init и является родителем всех пользовательских процессов. Это первый процесс, который виден сразу после загрузки системы Linux.

Он организует систему, при необходимости выполняет другие программы, а также запускает сетевые программы, работающие в фоновом режиме (deamon).

Процесс init имеет идентификационный номер 1. Сыновние процессы всегда имеют собственный уникальный ID, PCB-блок и отдельную структуру в таблице процессов.

Сыновний процесс в свою очередь может породить новый процесс, приложения могут создавать дерево процессов.

При определённых условиях процесс может порождать два новых потомка, тогда образуется два новых процесса.

Сыновний процесс может быть создан с собственным исполняемым образом, или с образом, являющимся дубликатом родительского процесса.

Если создаётся дубликат образа родительского процесса, то сыновний процесс наследует множество его атрибутов, включая среду, приоритет, стратегию планирования, ограничения по ресурсам, открытые файлы и разделы общей памяти. Если сыновний процесс перемещает указатель текущей позиции в файле или закрывает файл, то результаты этих действий видны родительскому процессу.

Если родителям выделяются дополнительные ресурсы после создание процесса потомка, то они не будут доступны потомку. С другой стороны, если сыновний процесс используется какие-либо ресурсы, они будут недоступны для процесса родителя.

Некоторые атрибуты родителя, которые не наследуются потомком:

1. ID процесса и PCB блок
2. Потомок не наследует никаких файловых блокировок, созданных родителем
3. Для сыновнего процесса используются собственные значения таких временных характеристик, как коэффициент загрузки процессора и время создания.

Несмотря на то, что сыновние процессы связаны некоторым отношением с родительскими, они действуют отдельно. То есть их программные и стековые счётчики действуют отдельно.

Так как разделы данных копируются, то процесс-потомок (сыновний процесс) может

изменять значения своих переменных, не оказывая влияния на родительскую копию данных.

Родительский и сыновний процессы совместно используют раздел программного кода и выполняют инструкции, расположенные непосредственно после вызова системной функции, создавшей сыновний процесс.

Они не выполняют эти инструкции на этапе блокировки из-за соперничества за процессор.

После создания образ сыновнего процесса может быть заменён другим исполняемым образом. При этом разделы программного кода, данных и стеков переписываются с новым образом процесса. Новый процесс сохраняет свой идентификационный номер и идентификационный номер родителя. Переменные среды также сохраняются, если во время замены исполняемого образа не были заданы новые переменные среды. Файлы, которые были открыты до момента замены исполняемого образа, остаются открытыми, и новый процесс будет создавать файлы с теми же файловыми расширениями. Время процессора при этом не сбрасывается.

Завершение процесса

Когда процесс завершается, его PCB разрушается, а используемое им адресное пространство и ресурсы освобождаются. Код завершения помещается в главную таблицу процессов, и как только родительский процесс примет этот код, соответствующая структура таблицы процессов будет удалена.

Условия, при которых процесс завершается:

1. Все инструкции выполнены
2. Процесс явным образом передаёт управление родительскому процессу или вызывает системную функцию, которая завершает процесс
3. Сыновние процессы могут завершится автоматически при завершении родительского процесса
4. Родительский процесс посылает сигнал о завершении сыновний процессов

Имеются ситуации аварийного завершения:

1. Процесс требует больше памяти, чем ему может предоставить система
2. Процесс пытается получить доступ к неразрешённым ресурсам
3. Процесс пытается выполнить некорректную инструкцию или запрещённые вычисления

Завершение процесса может быть инициировано пользователем, если этот процесс является интерактивным.

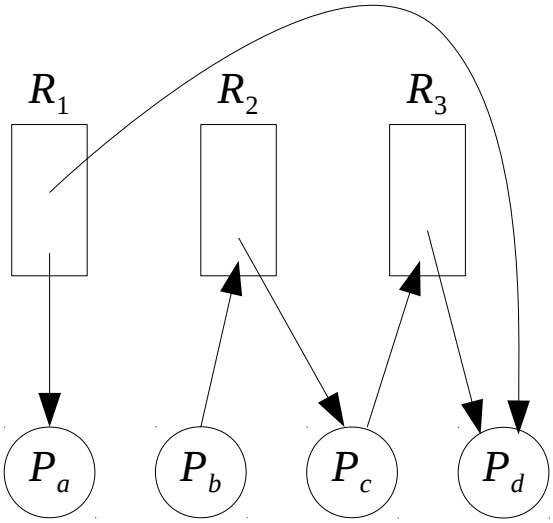
Родительский процесс должен ожидать до тех пор, пока не завершаться все его сыновние процессы. При этом, если процесс-родитель считает код завершения сыновнего процесса, то процесс-потомок покидает систему нормально, в противном случае процесс остаётся в состоянии зомби. Если родительский процесс никогда не примет сигнал, так как он уже успел завершиться и выйти из системы, то процесс-потомок остаётся в состоянии зомби до тех пор, пока процесс init не примет его код завершения. Большое количество зомбированных процессов негативно отражается на производительности системы.

Ресурсы процессора

Под ресурсом будем понимать всё то, что использует процесс в любой момент времени в качестве источника данных, средств обработки, вычислений или отображения информации.

Процесс должен сделать запрос к операционной системе на доступ к ресурсу. Если ресурс свободен, ОС позволяет процессу его использовать, если занят – процессу приходится ждать.

Граф распределения ресурсов:



Ресурсы могут быть разделяемыми (например R_1 с картинки).

Разделяемые ресурсы могут допускать одновременный или параллельный доступ нескольких процессов или разрешать доступ только одному процессу, но в течении ограниченного промежутка времени.

Если ресурс является таким, что к нему нельзя организовать разделяемый доступ (общий доступ), то о процессе, который использует этот ресурс, говорят, что он имеет монопольный доступ к ресурсу.

Существуют ресурсы, которые могут изменяться процессами.

Граф распределения ресурсов (см. рисунок выше) представляет собой ориентированный граф, где множество вершин представляет собой объединение множества процессов и множества ресурсов. Если ребро направлено от процесса к ресурсу, то оно называется ребром запроса, а если ребро направлено от ресурса к процессу, то его называют ребром назначения.

Ресурсы можно разделить по типам на *аппаратные*, *информационные* и *программные*.

Аппаратные ресурсы – это физические устройства (процессоры, оперативная память, устройства ввода/вывода и т. д.). Как правило они могут совместно использоваться несколькими процессами.

Для доступа к некоторым аппаратным ресурсам различными процессами используется система прерываний (например процессор, оперативная память).

Пример разделяемого, но не прерываемого ресурса: принтер.

Информационные ресурсы

К ним относятся данные объектов, системные данные, к которым в частности можно отнести переменные среды, файлы, дескрипторы, глобально определённые переменные, разделяемые ресурсы, которые могут быть модифицированы процессами.

Сыновний процесс наследует ресурсы родительского процесса, права доступа к ним, существующие на момент создания процесса-потомка.

Доступ к совместно используемым файлам и памяти с разрешением записи должен быть синхронизирован, для чего можно использовать семафоры и мьютексы (mutex, от mutual exclusion – взаимное исключение).

Программные ресурсы

К ним относятся разделяемые библиотеки. А также процессы могут совместно использовать приложения, программы, утилиты. При этом в оперативной памяти находится только одна копия программного кода, и существуют отдельные копии данных, по одной для каждого процесса.

Существуют системные ресурсы, для которых в операционной системе установлены жёсткие ограничения:

- Размер стека процесса
- Размер создаваемого файла
- Квант времени процессора, выделяемый процессу

- Объём оперативной памяти, выделенной процессу
- Количество дескрипторов открытых файлов

30.03.2016

Асинхронные и синхронные процессы

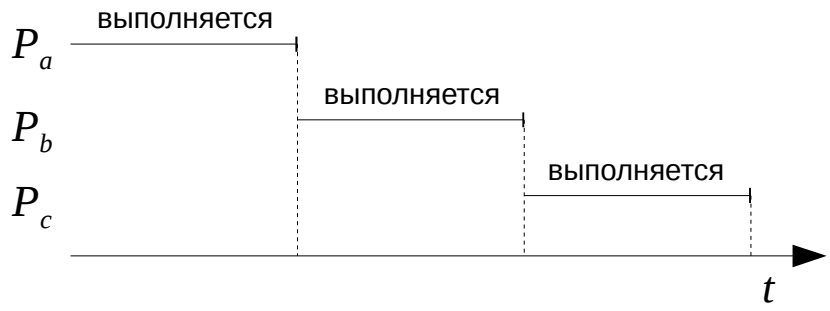
Асинхронные процессы выполняются независимо друг от друга, и между ними могут быть прямые родственные отношения родитель-сын, а могут и не быть.

Если процесс A создаёт процесс B, то они могут выполняться независимо, но в некоторый момент времени родительский процесс должен получить статус завершения сыновнего процесса.

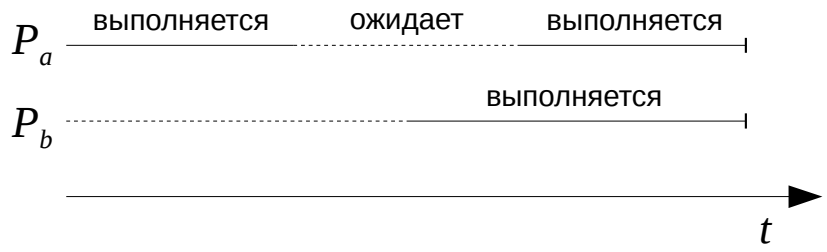
Если между процессами нет прямых родственных отношений, у них может быть общий родитель.

Асинхронные процессы могут выполняться последовательно, параллельно или с перекрытием.

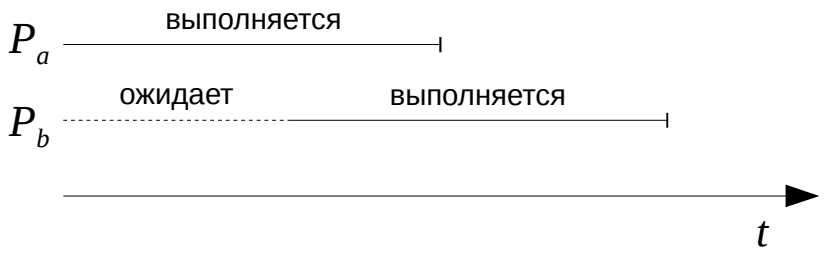
1. Асинхронные процессы. Последовательное выполнение



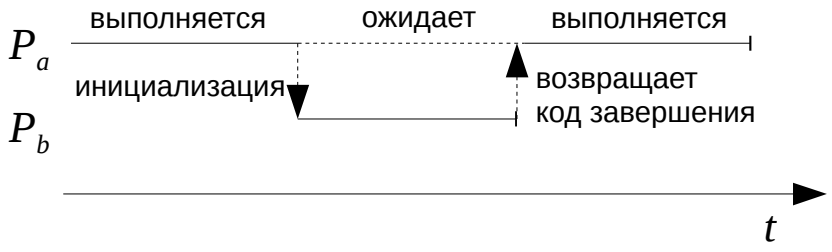
2. Асинхронные процессы. Параллельное выполнение



3. Асинхронные процессы. Выполнение с перекрытием



4. Синхронные процессы



В первом случае (при последовательном выполнении) при совместном использовании таких ресурсов, как файлы или оперативная память, синхронизация доступа к ресурсам не требуется.

Во втором и третьем случаях (параллельное выполнение или выполнение с перекрытием) процессы могут попытаться модифицировать данные, и, следовательно, здесь потребуется синхронизация по доступу к данным и к ресурсам.

В четвёртом случае (синхронные процессы) один процесс приостанавливает своё выполнение до тех пор, пока другой не завершит свою работу.

Потоки

Все потоки одного процесса существуют в одном и том же адресном пространстве. Все ресурсы, принадлежащие процессу, разделяются между потоками, при этом потоки не владеют никакими ресурсами.

Потоки разделяют дескрипторы файлов и файловые указатели, но каждый поток имеет собственные программные указатели, набор регистров, состояния и стек.

Все стеки потоков находятся в стековом разделе своего процесса. Раздел данных процесса совместно используется потоками процесса.

Поток может считывать и записывать информацию из области памяти своего процесса. Когда основной поток записывает данные в память, то любые сыновние потоки могут получить доступ к этим данным. Потоки могут создавать другие потоки в пределах того же процесса. Все потоки в одном процессе считаются равноправными. Потоки также могут приостановить, возобновить или завершить другие потоки в своём процессе. Потоки могут выполняться параллельно и в среде с одним процессором. Это достигается за счёт

переключения контекста, при условии, что операционная система поддерживает многозадачность при наличии единственного процессора.

Переключение контекста между потоками одного процесса происходит значительно быстрее, чем переключение контекста между процессами, так как потоки в рамках одного процесса не имеют собственного адресного пространства, и операционной системе требуется отслеживать меньший объём информации.

Контекст потока содержит только идентификационный номер потока ID, стек, набор регистров и приоритет. В регистрах содержится программный указатель и указатель стека. Программный код потока содержатся в текстовом разделе соответствующего процесса.

Сравнение потоков и процессов

Сходства:

1. Оба (и поток, и процесс) имеют ID, состояние, набор регистров, приоритет и привязку к определённой стратегии планирования.
2. Оба имеют атрибуты, которые описывают их особенности для операционной системы.
3. Имеют информационные блоки.
4. Оба разделяют ресурсы с родительским процессом.
5. Оба функционируют независимо от родительского процесса.
6. Создатель процесса и потока может управлять процессом и потоком.
7. Оба могут изменять свои атрибуты
8. Оба могут создавать новые ресурсы
9. Оба не имеют доступа к ресурсам другого процесса

Отличия:

Процессы	Потоки
Процессы имеют собственное адресное пространство	Потоки разделяют адресное пространство процесса, который их создал
Процессы имеют собственную копию раздела данных родительского процесса	Потоки имеют прямой доступ к разделу данных своего процесса
Процессы должны использовать специальный механизм межпроцессного взаимодействия для связи с братским процессом	Потоки могут напрямую взаимодействовать с потоками своего процесса
Для процессов требуются значительные затраты системных ресурсов	Для потоков почти не требуется системных затрат

Новые процессы требуют дублирования родительского процесса	Потоки не требуют
Процессы управляют только сыновними процессами	Потоки могут управлять в значительной степени потоками того же процесса
Изменения, вносимые в родительский процесс, на сыновних процессах не отражаются	Изменения, вносимые в основной поток (отмена, изменение приоритета) могут влиять на поведение других потоков процесса

Потоки, управляющие другими потоками

Потоки одного процесса считаются равноправными и находятся на одном уровне независимо от того, кто кого создал. Любой поток, имеющий доступ к ID другого потока, может отменить, приостановить, возобновить действие этого потока, либо изменить его приоритет. Отмена основного потока приведёт к завершению всех потоков процесса, то есть к ликвидации процесса. Любые изменения, внесённые в основной поток, могут повлиять на все потоки процесса. При изменении приоритета процесса все потоки также должны изменить свои приоритеты.

Преимущества использования потоков

1. При переключении контекста требуется меньше системных затрат или ресурсов, при условии, что в системе достаточное количество процессоров.
2. Повышается производительность приложения. Например, при использовании одного потока запрос к устройствам ввода/вывода может остановить весь процесс, а если имеется несколько потоков, то, пока один ждёт реализации запроса ввода/вывода, другие, независящие от заблокированного, могут продолжать выполнение, и, таким образом, приложение не заблокировано ожиданием.
3. Потоки имеют простую схему взаимодействия между параллельно выполняющимися потоками. То есть потоки не требуют специального механизма взаимодействия между подзадачами, могут напрямую передавать и получать данные других потоков, используют общую память, выделяемую в адресном пространстве процессом.
4. Использование потоков позволяет упростить структуру приложения. Каждому потоку назначается подзадача или подпрограмма, за выполнение которой он отвечает, и поток должен независимо управлять выполнением своей подзадачи. Каждому потоку можно присвоить приоритет, отражающий важность выполнения задачи, что позволяет упростить поддержку программного кода.

Недостатки использования потоков

1. Потоки могут легко разрушить адресное пространство процесса.

Потоки совместно используют одно и то же адресное пространство, и это делает данные незащищёнными от искажений. Потоки могут разрушить информацию процесса во время гонки данных, и, чтобы этого не происходило, необходимо организовывать синхронизацию доступа к данным.

2. Один поток может ликвидировать всю программу.

   В силу того, что потоки не имеют собственного адресного пространства, они не изолированы, и если поток стал причиной фатального нарушения доступа, это может привести к завершению всего процесса.

3. Потоки не могут многократно использоваться другими программами.

   Потоки зависят от процесса, в котором существуют, и их невозможно от этого процесса отделить, то есть потоки не могут существовать вне процессов, в которых они созданы.

Взаимоотношения между синхронизируемыми задачами

Существует четыре основных типа отношений синхронизации между любыми двумя потоками в одном процессе или между любыми двумя процессами в одном приложении.

1. Старт-старт (СС)
2. Финиш-старт (ФС)
3. Старт-финиш (СФ)
4. Финиш-финиш (ФФ)

Старт-старт

В отношениях такого типа синхронизация выполняется из тех условий, что одна задача не может начаться до тех пор, пока не начнётся другая. Одна задача может начаться раньше другой, но не позже.

Финиш-старт

В отношениях такого типа задача А не может завершиться до тех пор, пока не начнётся задача В. Такой тип отношений является типичным для родительско-сыновних процессов: родительский процесс не может завершить выполнение некоторых операций до тех пор, пока не будет сгенерирован сыновний процесс, или пока не будет получена обратная связь от сыновнего процесса, который начал выполнение. Сыновний процесс, предоставивший родительскому необходимую информацию, продолжает выполняться, а родительский после

этого может завершиться.

Старт-финиш

Это отношение можно считать обратным вариантом к отношению финиш-страт, то есть одна задача не может начаться до тех пор, пока не завершится другая. Эти отношения обычно предполагают наличие информационной зависимости между задачами.

Финиш-финиш

Это отношение означает, что одна задача не может завершиться до тех пор, пока не завершится другая. Этот тип отношений применим к описанию отношений между сыновними и родительскими процессами: обычно родительский процесс обязан ожидать, пока не завершатся все сыновние процессы, и только после этого может завершиться сам.