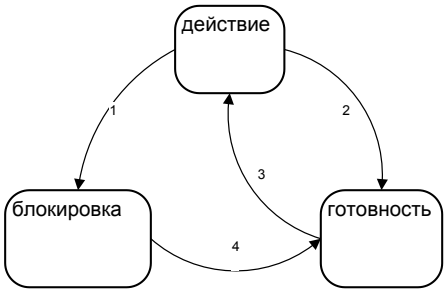
Процессы. Основные понятия.

в период своего выполнения процесс проходит через ряд дискретных состояний.



Порядок выбора процессов на исполнение из находящихся в состоянии готовности определяется специальным алгоритмом – планировщиков процессов.
Процесс может выполняться в одном из двух состояний – пользовательском и системном.

Алгоритмы планирования процессов

Все множество алгоритмов планирования можно разделить на 3 класса:

- Невытесняющая многозадачность – ОС загружает в память несколько задач, но активно только одно из них.
- совместная или кооперативная многозадачность - основана на том что активному процессу позволяется выполняться пока он сам по собственной инициативе, не отдаст управление операционной системе для того чтобы та выбрала из очереди другой готовый к выполнение процесс
- вытесняющая многозадачность – это такие способы планирования потоков в которых решение о переключении процессора с выполнения одного потока на выполнение другого потока принимается операционной системой а не активной задачей.

При вытесняющем мультипрограммировании функции планирования процессов/потоков целиком сосредоточены в операционной системе и программист пишет свое приложение, не заботясь о том, что оно будет выполняться одновременно с другими задачами. При этом операционная система выполняет следующие функции – определяет момент снятия с выполнения активного потока, запоминает его контекст, выбирает из очереди готовых потоков следующий, запускает новый поток, на выполнение, загружая его контекст.

При совместной или кооперативной многозадачности механизм планирования распределен между операционной системой и прикладными программами. Прикладная программа, получив управление от операционной системы, сама определяет момент завершения очередного цикла своего выполнения и только затем передает управление ОС с помощью какого-либо системного вызова. Такой механизм создает проблемы, как для пользователей, так и для разработчиков приложений.

Однако такой подход иногда является преимуществом, так как дает возможность разработчику приложений самому проектировать алгоритм планирования прерываний (приостановления вычислений) наиболее подходящий для данного фиксированного набора задач.

Некоторые известные алгоритмы планирования процессов:

1. В пакетной обработке
   - «Первым пришел первым обслужен». – алгоритм без переключений.
   - «кратчайшая задача – первая» - алгоритм без переключений. Выбирается та задача, которая будет выполняться наименьшее количество времени.
   - «наименьшее оставшееся время выполнения» - алгоритм с переключениями. Когда поступает новая задача то происходит сверка нового задания с выполняющимся и если новое работать будет меньше, то оно и будет выполняться.
2. в системах разделения времени:
   a. «кольцевой режим».
   b. Система с использованием приоритетов. Существует две разновидности:
      i. Со статическими приоритетами.

ii. С динамически меняющимися приоритетами - итоговое значение приоритета складывается из двух составляющих – статической и динамической. Статическая задается при запуске. Динамическая зависит от того как процесс использует процессорное время.

параллельные процессы

Процессы называются параллельными если они существуют одновременно. Параллельные процессы могут работать совершенно независимо друг от друга либо быть асинхронными. Это означает что данным процессам необходимо периодически синхронизироваться и взаимодействовать при они могут быть достаточно сложны.

1. «состояние гонки» и критическая секция

Во время выполнения параллельных процессов может возникнуть ситуация одновременного доступа из нескольких процессов к массивам данных, хранимых в глобальных переменных. В этом случае результат такого обращения к будет непредсказуемым и в большинстве случаев данные будут испорчены.
Для того, чтобы избежать этой ситуация придуман и используется в различных языках программирования механизм блокировок «критическая секция», который гарантирует что только один поток/процесс находится в этом состоянии, остальные же потоки находятся в заблокированном состоянии и ожидают выхода процесса из этой части программы.

2. Сигналы.

Сигнал дает возможность задаче реагировать на событие, источником которого может быть операционная система или другая задача. Сигналы вызывают прерывание задачи и выполнение заранее предусмотренных действий. Сигналы могут вырабатываться синхронно, то есть как результат работы самого процесса, а могут быть направлены процессу другим процессом, то есть вырабатываться асинхронно. Синхронные сигналы чаще всего приходят от системы прерываний процессора и свидетельствуют о действиях процесса, блокируемых аппаратурой, например деление на нуль, ошибка адресации.
Примером асинхронного сигнала является сигнал с терминала – например нажатие CTRL-C для снятия процесса с выполнения. В результате нажатия этой комбинации клавиш ОС вырабатывает сигнал и направляет его активному процессу. В данном случае реакцией на сигнал является безусловное завершение процесса.
В системе может быть определен набор сигналов. Программный код процесса, которому поступил сигнал, может либо:

- проигнорировать его,
- прореагировать на него стандартным действием (например, завершиться),
- выполнить специфические действия, определенные прикладным программистом. В данном случае в программном коде необходимо предусмотреть специальные системные вызовы, с помощью которых операционная система информируется, какую процедуру надо выполнить в ответ на поступление того или иного сигнала.

Наиболее часто используемые сигналы:
9 (KILL) – принудительное уничтожение процесса
15 (TERM) - программное завершение процесса
1 (HUP) - сигнал отбоя. Многие системные процессы при получении этого сигнала перечитывают свои конфигурационные файлы.
Иногда, правда, редко, процессы впадают в такие состояния, что их нельзя "убить" даже выдав команду kill -9 <pid>.
За сигналами SIGUSR1, SIGUSR2 не закреплено системных функций, они доступны для использования под любые цели.

Создание процессов и потоков.

Создать процесс это, прежде всего, означает создать описатель процесса, в качестве которого выступает одна или несколько информационных структур, содержащих все сведения о процессе, необходимые операционной системе для управления им. В число таких сведений могут входить например идентификатор процесса, данные о расположении в памяти исполняемого модуля, степень привилегированности. В Unix-е это называется дескриптором процесса, в NT – объект-процесс (object-process).

В Unix процесс имеет следующие атрибуты:

1. программный код
2. внутренние данные(переменные , массивы, объекты и т.д.)
3. идентификаторы процесса - идентификатор процесса предка - идентификатор группы процесса
4. реальный и эффективный идентификаторы пользователя и гр. Пользователя.
5. список групп доступа
6. информация об открытых файлах
7. текущий каталог

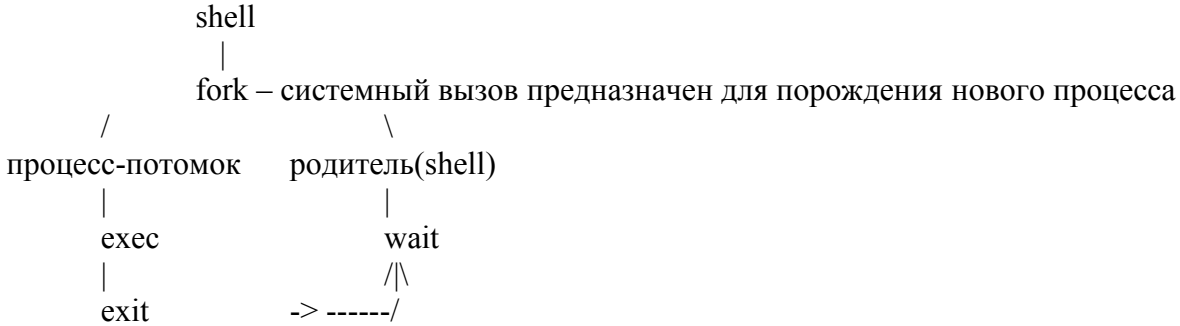
При управлении процессами операционная система использует два основных типа информационных структур: дескриптор процесса и контекст процесса.

Дескрипторы отдельных процессов объединены в список образующий таблицу процессов. Контекст, так же как и дескриптор процесса, доступен только программам ядра, то есть находится в виртуальном адресном пространстве операционной системы, однако он хранится не в области ядра, а непосредственно примыкает к образу процесса и перемещается вместе с ним, если это необходимо из оперативной памяти на диск.

1. Порождение процессов UNIX. Запуск программ на выполнение.

Новые процессы UNIX создаются только путем создания дубликата текущего процесса с помощью системного вызова FORK.

Схема создания нового процесса:



после того как к wait поступает сигнал от exit, shell корректно удаляет процесс-потомок.

В случае сбоя о потомке будет заботиться init

Новый процесс получает от исходного всю информацию, в том числе информацию об открытых файлах. Однако три файла вновь открываются автоматически: Stdin – стандартный ввод, Stdout – стандартный вывод, Stderr – файл диагностических сообщений.

2. Процессы Windows

У каждого процесса есть есть 4-х гигабайтное адресное пространство(32 разряда) в котором пользователь занимает 2 гигабайта (однако это может быть до 3-х ГБ увеличено настройкой) а операционная система занимает оставшуюся часть. Т.о. операционная система присутствует в адресном пространстве каждого процесса, хотя она защищена от изменений с помощью аппаратного блока управления памятью MMU. У процесса есть идентификатор процесса, один или несколько потоков, список дескрипторов и маркер доступа, хранящий информацию защиты. Процессы создаются с помощью вызова Win32, который принимает на входе имя исполняемого файла, определяющего начальное содержимое адресного пространства, и создает первый поток. Потоки могут создаваться

динамически в процессе работы. Когда поток завершает свою работу, он может прекратить свое существование, процесс же завершается с завершением последнего активного потока.
Т.к. переключение потоков в Win2k занимает много времени (переключение в режим ядра а затем возврат в режим пользователя), то есть более простой механизм но обеспечивающий псевдопараллелизм – волокна, подобны потокам но планируемые в пространстве пользователя создавшей их программой. У каждого потока может быть несколько волокон, так же как у процесса может быть несколько потоков, с той разницей, что когда волокно логически блокируется, оно помещается в очередь блокированных волокон, после чего для работы выбирается другое волокно в контексте того же потока. ОС не знает о смене волокон, так как все тот же поток продолжает работу.
Необходимые параметры для системного вызова CreateProcess

- указатель на имя исполняемого файла
- сама командная строка (неразобранная на составные части)
- указатель на описатель защиты процесса
- указатель на описатель защиты для начального потока
- бит, управляющий наследованием дескрипторов
- разнообразные флаги (режим отладки, консоли и т.д.)
- указатель на строки окружения
- указатель на имя текущего каталога нового процесса
- указатель на структуру, описывающую начальное окно на экране
- указатель на структуру, возвращающему процесу 18 значений

Идентификаторы (процессов и пользователей)

1. Идентификатор процесса PID.

Каждый процесс в системе идентифицируется целым положительным числом, называемым идентификатором процесса.

2. Реальные и эффективные (действующие) идентификаторы пользователя и группы пользователей. (UID/EUID, GID/EGID).

Каждый пользователь системы идентифицируется целым положительным числом, называемым индентификатором пользователя UID. Пользователь может быть членом одной или нескольких групп пользователей. Каждое число, поставленное в соответствие группе, называется идентификатором группы – GID.
Процесс имеет:

- Реальные идентификаторы пользователя и группы, значения которой равны соответствующим идентификатором пользователя, запустившим на выполнение этот процесс.
- Эффективные (действующие) идентификаторы пользователя (группы), на основе которых определяются привилегии процесса. обычно EUID и EGID совпадает с реальными, кроме случая, когда в ходе защиты файла установлены признаки setuid и/или setgid. В этом случае эффективные идентификаторы (EUID) пользователя и/или группы (EGID) становятся равными идентификаторам пользователя (группы) владельца файла.

Системные вызовы

Pid = fork(); - Системный вызов fork служит для создания нового процесса.
Pid = wait (status); – ожидание завершения процесса потомка.
Exit(arg); - завершить процесс
Execl – выполнить (запустить) файл. Новая программа загружается из файла с именем “name”, который либо является исполняемым либо содержит интерпретируемые команды. При успешном завершении работы значение **execl()** не возвращается, при ошибке возвращаемое значение равно –1.