4.1 Процессы

4.1.1 Понятие процесса

Процесс (задача) - программа, находящаяся в режиме выполнения.

С каждым процессом связывается его **адресное пространство,** из которого он может читать и в которое он может писать данные.

Адресное пространство содержит:

- саму программу
- данные к программе
- стек программы

С каждым процессом связывается набор **регистров,** например:

- счетчика команд (в процессоре) - регистр в котором содержится адрес следующей, стоящей в очереди на выполнение команды. После того как команда выбрана из памяти, счетчик команд корректируется и указатель переходит к следующей команде.
- указатель стека
- и д.р.

Во многих операционных системах вся информация о каждом процессе, дополнительная к содержимому его собственного адресного пространства, хранится в **таблице процессов** операционной системы.

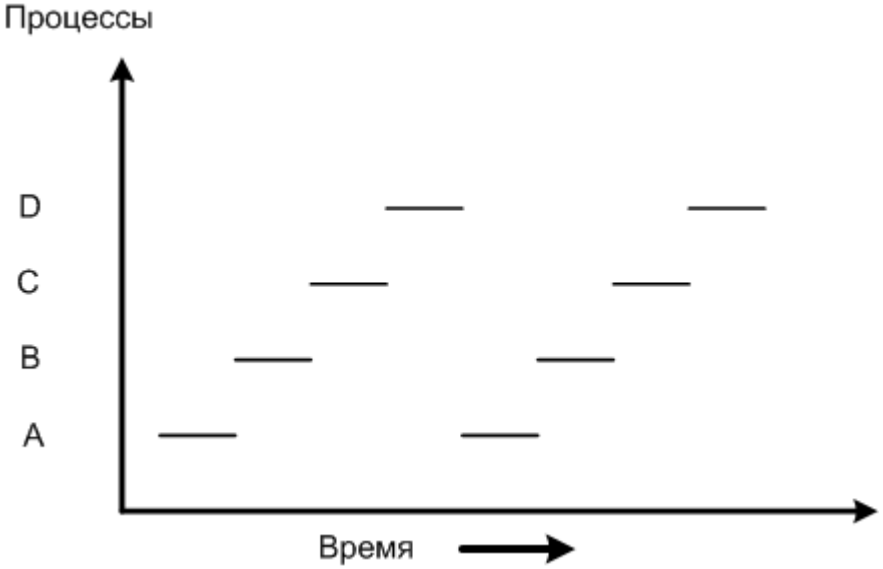
Некоторые поля таблицы:

Управление процессом	**Управление памятью**	**Управление файлами**
Регистры	Указатель на текстовый сегмент	Корневой каталог
Счетчик команд	Указатель на сегмент данных	Рабочий каталог
Указатель стека	Указатель на сегмент стека	Дескрипторы файла
Состояние процесса		Идентификатор пользователя
Приоритет		Идентификатор группы
Параметры планирования		
Идентификатор процесса		
Родительский процесс		
Группа процесса		
Время начала процесса		
Использованное процессорное время		

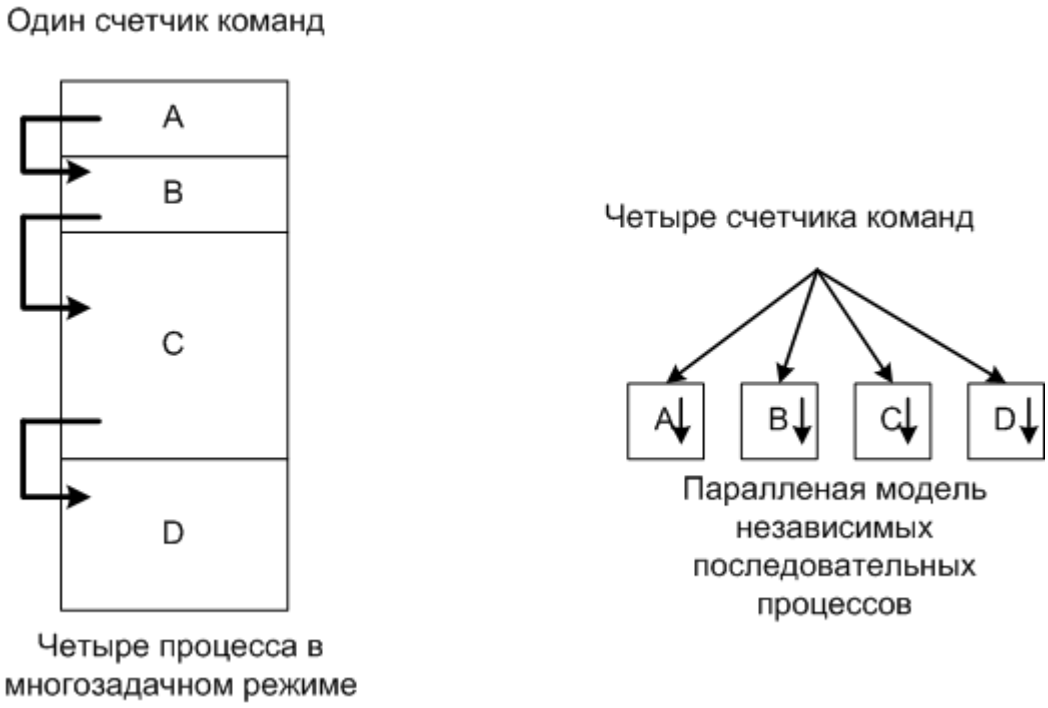
4.1.2 Модель процесса

В многозадачной системе реальный процессор переключается с процесса на процесс, но для упрощения модели рассматривается набор процессов, идущих параллельно (псевдопараллельно).

Рассмотрим схему с четырьмя работающими программами.



В каждый момент времени активен только один процесс



С права представлены параллельно работающие процессы, каждый со своим счетчиком команд. Разумеется, на самом деле существует только один физический счетчик команд, в который загружается логический счетчик команд текущего процесса. Когда время, отведенное текущему процессу, заканчивается, физический счетчик команд сохраняется в памяти, в логическом счетчике команд процесса.

4.1.3 Создание процесса

Три основных события, приводящие к созданию процессов (вызов **fork** или **CreateProcess**):

- Загрузка системы
- Работающий процесс подает системный вызов на создание процесса
- Запрос пользователя на создание процесса

Во всех случаях, активный текущий процесс посылает системный вызов на создание нового процесса.

В UNIX каждому процессу присваивается идентификатор процесса (PID - Process IDentifier)

4.1.4 Завершение процесса

Четыре события, приводящие к остановке процесса (вызов **exit** или **ExitProcess**):

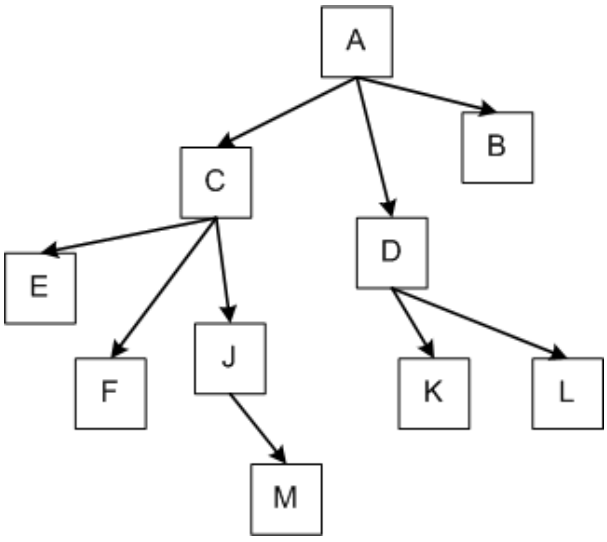
- Плановое завершение (окончание выполнения)
- Плановый выход по известной ошибке (например, отсутствие файла)
- Выход по неисправимой ошибке (ошибка в программе)
- Уничтожение другим процессом

Таким образом, приостановленный процесс состоит из собственного адресного пространства, обычно называемого **образом памяти** (**core image**), и компонентов таблицы процессов (в числе компонентов и его регистры).

4.1.5 Иерархия процессов

В UNIX системах заложена жесткая иерархия процессов. Каждый новый процесс созданный системным вызовом fork, является дочерним к предыдущему процессу. Дочернему процессу достаются от родительского переменные, регистры и т.п. После вызова fork, как только родительские данные скопированы, последующие изменения в одном из процессов не влияют на другой, но процессы помнят о том, кто является родительским.

В таком случае в UNIX существует и прародитель всех процессов - процесс **init**.

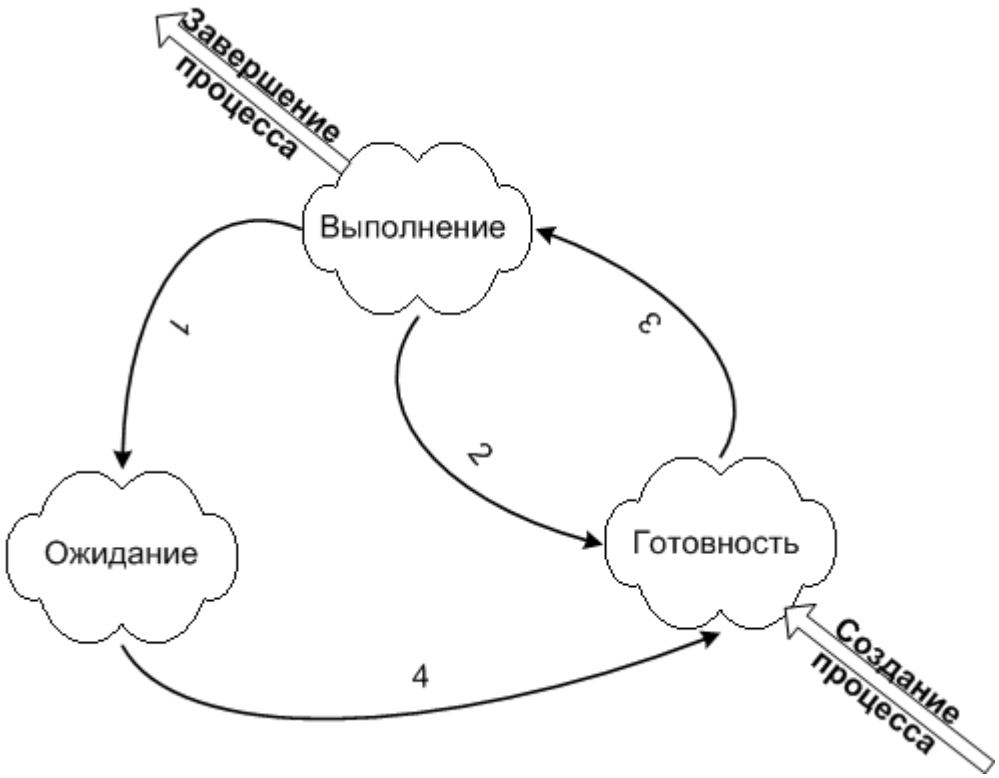


Дерево процессов для систем UNIX

4.1.6 Состояние процессов

Три состояния процесса:

- Выполнение (занимает процессор)
- Готовность (процесс временно приостановлен, чтобы позволить выполняться другому процессу)
- Ожидание (процесс не может быть запущен по своим внутренним причинам, например, ожидая операции ввода/вывода)



Возможные переходы между состояниями.

1. Процесс блокируется, ожидая входных данных

2. Планировщик выбирает другой процесс

3. Планировщик выбирает этот процесс

4. Поступили входные данные

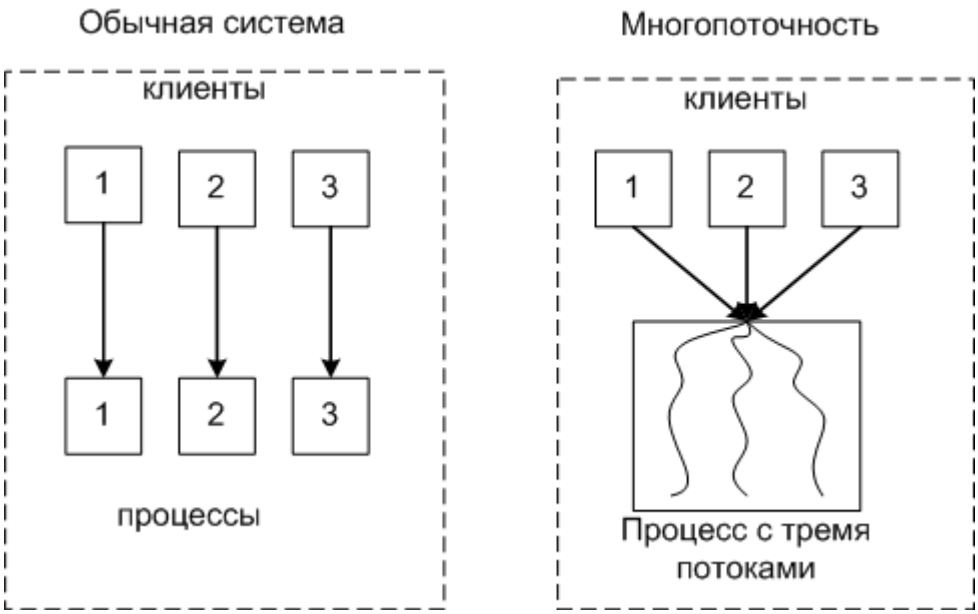
Переходы 2 и 3 вызываются планировщиком процессов операционной системы, так что сами процессы даже не знают о этих переходах. С точки зрения самих процессов есть два состояния выполнения и ожидания.

На серверах для ускорения ответа на запрос клиента, часто загружают несколько процессов в режим ожидания, и как только сервер получит запрос, процесс переходит из "ожидания" в "выполнение". Этот переход выполняется намного быстрее, чем запуск нового процесса.

4.2 Потоки (нити, облегченный процесс)

4.2.1 Понятие потока

Каждому процессу соответствует адресное пространство и одиночный **поток** исполняемых команд. В многопользовательских системах, при каждом обращении к одному и тому же сервису, приходится создавать новый процесс для обслуживания клиента. Это менее выгодно, чем создать квазипараллельный поток внутри этого процесса с одним адресным пространством.



Сравнение многопоточной системы с однопоточной

4.2.2 Модель потока

С каждым потоком связывается:

- Счетчик выполнения команд
- Регистры для текущих переменных
- Стек
- Состояние

Потоки делят между собой элементы своего процесса:

- Адресное пространство
- Глобальные переменные
- Открытые файлы
- Таймеры
- Семафоры
- Статистическую информацию.

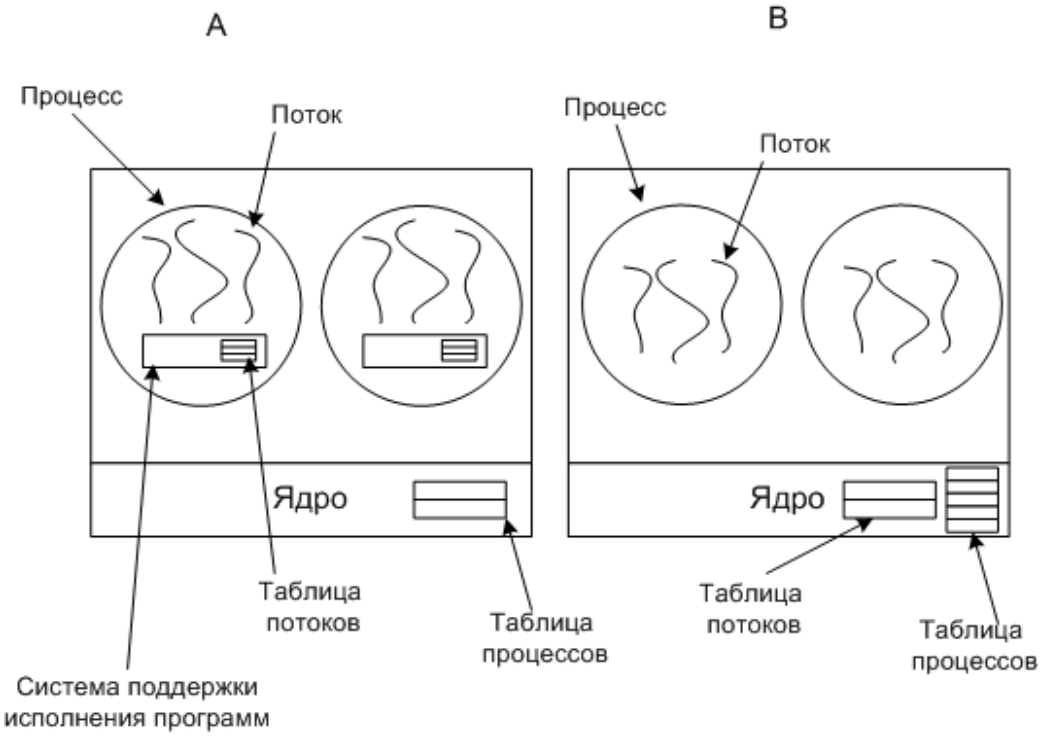
В остальном модель идентична модели процессов.

В POSIX и Windows есть поддержка потоков на уровне ядра.

4.2.3 Преимущества использования потоков

1. Упрощение программы в некоторых случаях, за счет использования общего адресного пространства.
2. Быстрота создания потока, по сравнению с процессом, примерно в 100 раз.
3. Повышение производительности самой программы, т.к. есть возможность одновременно выполнять вычисления на процессоре и операцию ввода/вывода. Пример: текстовый редактор с тремя потоками может одновременно взаимодействовать с пользователем, форматировать текст и записывать на диск резервную копию.

4.2.4 Реализация потоков в пространстве пользователя, ядра и смешанное



A - потоки в пространстве пользователя
B - потоки в пространстве ядра
В случае **A** ядро о потоках ничего не знает. Каждому процессу необходима **таблица потоков**, аналогичная таблице процессов.

Преимущества случая **A**:

- Такую многопоточность можно реализовать на ядре не поддерживающим многопоточность
- Более быстрое переключение, создание и завершение потоков
- Процесс может иметь собственный алгоритм планирования.

Недостатки случая **A**:

- Отсутствие прерывания по таймеру внутри одного процесса
- При использовании блокирующего (процесс переводится в режим ожидания, например: чтение с клавиатуры, а данные не поступают) системного запроса все остальные потоки блокируются.
- Сложность реализации