**Тема 3. Общие сведения о процессах и потоках**

**Лекция 8 «Процессы в операционной системе. Модель процесса».**

**Лекция 9 «Создание процесса. Завершение процесса».**

**Лекция 10 «Иерархия процесса. Состояние процесса. Реализация процесса».**

**Процесс** - это выполняемая в данный момент программа. Компьютерная программа сама по себе лишь пассивная последовательность инструкций, а процесс - непосредственное выполнение этих инструкций. Выполнение процесса должно осуществляться последовательно. Процесс определяется как сущность, представляющая основную единицу работы, которая должна быть реализована в системе.



Рис.1 Статусы процессов в современных ОС.

На любой ЭВМ всегда имеется процесс, соответствующий операционной системе этой ЭВМ, а также один или несколько процессов, отвечающих пользовательским программам. На однопроцессорных ЭВМ в любой момент времени может выполнятся только один процесс. Любая ОС должна уметь производить запуск процессов, приостановку, их выполнение, завершение их выполнения и синхронизацию процессов между собой. Для каждого процесса ОС предоставляет собственное адресное пространство. С целью обеспечения переносимости адресное пространство всегда начинается с нуля и продолжается непрерывно до предела. Ни один процесс кроме ОС не знает в какой именно части физической памяти и каким образом располагается его адресное пространство.

Операционная система контролирует следующую деятельность, связанную с процессами:

1. создание и удаление процессов;

2. планирование процессов;

3. синхронизация процессов;

4. коммуникация процессов;

5. разрешение тупиковых ситуаций.

Каждому процессу ОС системой выделяются ресурсы: дисковое пространство, устройство ввода- вывода, канал передачи информации и прочее. Каждый процесс имеет возможность создавать другие процессы и контролировать их выполнения. Каждому процессу в ОС отводятся определенные права.

ОС для каждого из процесса хранит всю информацию об этих процессах в специальных таблицах. В этих таблицах обязательно описываются права процесса, полное состояние регистров процессора для данного процесса, объем ОП отводимый процессу и отображение этой памяти на реальную физическую память, а также список всех ресурсов, отводимых данному процессу. При переключении с одного процесса на другой ОС пользуется этими учетными записями.

**Модель процесса.** Все функционирующее на компьютере ПО организовано в виде набора последовательных процессов, или процессов. Процессом является выполняемая программа, включая текущие значения счетчика команд, регистров и переменных. У каждого процесса есть собственный виртуальный центральный процессор. На самом деле, реальный процессор переключается с процесса на процесс, но для лучшего понимания системы значительно проще рассматривать набор процессов, идущих параллельно (псевдопараллельно), чем пытаться представить себе процессор, переключающийся от программы к программе. Это переключение и называется многозадачностью или мультипрограммированием. Поскольку процессор переключается между программами, скорость, с которой процессор производит свои вычисления, будет непостоянной и, возможно, даже будет отличной при каждом новом запуске процесса. Поэтому не следует программировать процессы, исходя из каких-либо жестко заданных временных предположений.

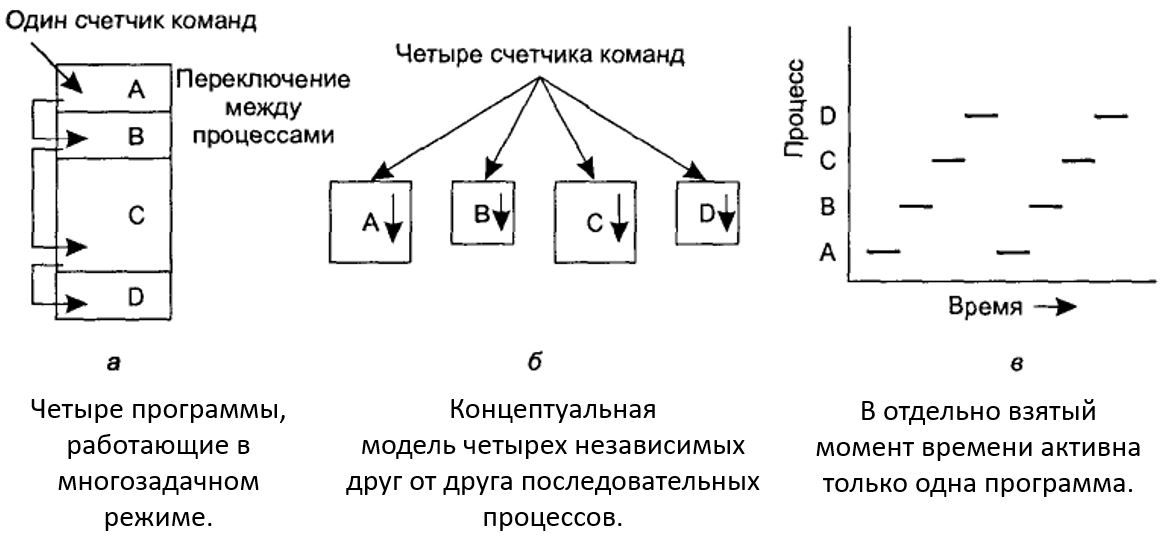


Рис.2 Модели процесса

**Воспользуемся следующей аналогией:** представьте себе программиста, разбирающегося в кулинарии и пекущего торт на день рождения своей дочери. В его распоряжении есть рецепт торта, кухня, оборудованная всем необходимым, и ингредиенты для торта: мука, яйца, сахар, ванилин и т. п. Согласно этой аналогии, рецепт - это программа (то есть алгоритм, записанный в заданном виде), программист исполняет роль процессора, а ингредиенты торта являются входными данными. Процессом является следующая последовательность действий: программист читает рецепт, смешивает продукты и печет торт. Теперь представьте, что на кухню прибегает плачущий сын программиста и кричит, что его ужалила пчела. Программист отмечает, на чем он остановился (сохраняет текущее состояние процесса), находит справочник по оказанию первой помощи и действует в соответствии с инструкцией. Таким образом, наш процессор переключился с одного процесса (выпечка торта) на другой, с большим приоритетом (оказание первой помощи), и у каждого процесса есть своя программа (рецепт торта и справочник по оказанию первой помощи). После проведения всех необходимых процедур по борьбе с укусом пчелы программист возвращается к торту, продолжая с той операции, на которой он прервался. У него есть программа, входные и выходные данные, а также состояние. Один процессор может переключаться между различными процессами, используя некий алгоритм планирования для определения момента переключения от одного процесса к другому.

**Создание процесса.** Простейшей операционной системе не требуется создание новых процессов, поскольку внутри них работает одна-единственная программа, запускаемая во время включения устройства. В более сложных системах надо создавать новые процессы. Обычно они создаются:

* При запуске [ОС](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) (например, когда происходит [инициализация](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B8%D1%86%D0%B8%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) [драйверов устройств](https://ru.wikipedia.org/wiki/%D0%94%D1%80%D0%B0%D0%B9%D0%B2%D0%B5%D1%80)),
* При появлении запроса на создание процесса - происходит в случае, если работающий процесс выполняет [системный вызов](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%BD%D1%8B%D0%B9_%D0%B2%D1%8B%D0%B7%D0%BE%D0%B2).

Операционной системе необходим способ, позволяющий удостовериться в наличии всех необходимых процессов. В простейших системах, а также системах, разработанных для выполнения одного-единственного приложения, можно реализовать такую ситуацию, в которой все процессы, которые когда-либо могут понадобиться, присутствуют в системе при ее загрузке. В универсальных системах необходим способ создания и прерывания процессов по мере необходимости. События, приводящие к созданию процессов:

* Инициализация системы.
* Выполнение работающим процессом системного запроса на создание процесса
* Запрос пользователя на создание процесса.
* Инициирование пакетного задания (способ запуска задач на исполнение для избегания простоев вычислительных систем, список запускаемых программ с указанием параметров запуска и входных данных. Задачи запускаются последовательно, пример, мейнфреймы производства IBM.).

Обычно при загрузке операционной системы создаются несколько процессов. Некоторые из них являются высокоприоритетными процессами, то есть обеспечивающими взаимодействие с пользователем и выполняющими заданную работу. Остальные процессы являются фоновыми, они не связаны с конкретными пользователями, но выполняют особые функции. Например, один фоновый процесс может быть предназначен для обработки приходящей на компьютер почты, активизируясь только по мере появления писем. Другой фоновый процесс может обрабатывать запросы к web-страницам, расположенным на компьютере, и активизироваться для обслуживания полученного запроса.

В интерактивных системах пользователь может запустить программу, набрав на клавиатуре команду или дважды щелкнув на значке программы. В обоих случаях результатом будет создание нового процесса и запуск в нем программы.

С технической точки зрения во всех перечисленных случаях новый процесс формируется одинаково: текущий процесс выполняет системный запрос на создание нового процесса. В роли текущего процесса может выступать процесс, запущенный пользователем, системный процесс, инициированный клавиатурой или мышью, а также процесс, управляющий пакетами. В любом случае этот процесс всего лишь выполняет системный запрос на создание нового процесса. Системный запрос заставляет операционную систему создать новый процесс, а также прямо или косвенно содержит информацию о программе, которую нужно запустить в этом процессе.

В UNIX существует только один системный запрос, направленный на создание нового процесса: fork (ветвление). Этот запрос создает дубликат вызываемого процесса. После выполнения запроса fork двум процессам — родительскому и дочернему — соответствуют одинаковые образы памяти, строки окружения и одни и те же открытые файлы. Обычно дочерний процесс выполняет системный вызов execve для изменения своего образа памяти и запуска новой программы. Так, когда пользователь набирает на клавиатуре команду sort, оболочка создает путем ветвления дочерний процесс, который и выполняет программу sort. Смысл этого двухступенчатого процесса заключается в том, что дочерний процесс успевает обработать описания файлов после fork, но до execve, чтобы выполнить перенаправление стандартных устройств ввода и вывода и потока сообщений об ошибках. В Windows же вызов всего одной функции CreateProcess интерфейса Win32 управляет и созданием процесса, и запуском в нем нужной программы. И в UNIX, и в Windows после создания нового процесса родительский и дочерний процессы имеют собственные различные адресные пространства. При изменении любым процессом слова в адресном пространстве — это изменение незаметно для других процессов. В UNIX начальное адресное пространство дочернего процесса является копией родительского, но сами адресные пространства различны, и перезаписываемая память совместно не используется (некоторые приложения UNIX совместно используют текст программы, поскольку его нельзя модифицировать). В то же время созданный процесс может использовать совместно с родительским процессом некоторые другие ресурсы, например, открытые файлы.

**Завершение процесса.** После того как процесс создан, он начинает выполнять свою работу. Но ничто не длится вечно, даже процесс — рано или поздно он завершится, чаще всего благо­даря одному из следующих событий:

* Обычный выход (преднамеренно);
* Выход по ошибке (преднамеренно);
* Выход по неисправимой ошибке (непреднамеренно);
* Уничтожение другим процессом (непреднамеренно).

В основном процессы завершаются по мере выполнения своей работы. Первой причиной завершения процесса - после окончания компиляции программы компилятор выполняет системный запрос, чтобы сообщить операционной системе об окончании работы. В Unix системный вызов ***exit***, в Windows – ***ExitProcess***. Второй причиной завершения процесса может стать неустранимая ошибка. Например, если пользователь набрал на клавиатуре команду ее foo.c для компиляции программы foo.c, а соответствующего файла не существует, компилятор просто закончит работу. Интерактивные процессы, рассчитанные на работу с экраном, обычно не завершают работу при получении неверных параметров, вместо этого выводя на экран диалоговое окно и прося пользователя ввести правильные параметры.

Третьей причиной завершения процесса является ошибка, вызванная самим процессом, чаще всего связанная с ошибкой в программе. В качестве примера можно привести выполнение недопустимой команды, обращение к несуществующей области памяти и деление на ноль. Четвертой причиной завершения процесса может служить выполнение другим процессом системного запроса на уничтожение процесса. В Unix – ***kill***, в Win32 – ***TerminateProcess***.

**Иерархия процессов.** В некоторых системах родительский и дочерний процессы остаются связанными между собой определенным образом. Дочерний процесс также может, в свою очередь, создавать процессы, формируя иерархию процессов.

В UNIX процессы образуют иерархию процессов. У процесса может быть только один родитель и сколько угодно потомков. Сигнал, посылаемый пользователем с клавиатуры, доставляется всем членам группы, взаимодействующим с клавиатурой в данный момент (обычно это все активные процессы, созданные в текущем окне). Каждый из процессов может перехватить сигнал, игнорировать его или выполнить другое действие, предусмотренное по умолчанию.

В Windows не существует понятия иерархии процессов, и все процессы равноправны. Единственное, в чем проявляется иерархия – создание процесса, в котором родительский процесс получает специальный маркер, позволяющий контролировать дочерний процесс. Но маркер можно передать другому процессу, нарушая иерархию. В UNIX это невозможно.

В многозадачных операционных системах появилась возможность работать одновременно с несколькими процессами. Операционные системы с вытесняющей многозадачностью позволяли добиться ощущения работы нескольких процессов одновременно. При этом потребовались средства управления несколькими процессами.

Unix — одна из первых многозадачных ОС. 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.

Каждый процесс имеет уникальный числовой идентификатор PID. Процессы в ней имеют древовидную иерархию, где корнем является процесс init c PID 1. Новый процесс можно создать системным вызовом fork, он будет являться точной копией процесса родителя. Любой процесс кроме init всегда имеет процесс родитель (атрибут PPID (англ. Parent PID)); процессы, родитель которых завершил свою работу становятся дочерними процессами init.

**Состояние процесса.** Несмотря на то, что процесс является независимым объектом, со своим счетчиком команд и внутренним состоянием, существует необходимость взаимодействия с другими процессами.

Три возможных состояния процесса:

- выполнение (в этот конкретный момент использующий процессор);

- готовность (процесс временно приостановлен, чтобы позволить выполняться другому процессу);

- ожидание (процесс не может быть запущен прежде, чем произойдет некое внешнее событие).

|  |  |
| --- | --- |
| http://konstantinos777.narod.ru/1_10.files/image006.jpg | 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 об этих переходах. С точки зрения самих процессов есть два состояния выполнения и ожидания. |

Состояния процесса: 1) Выполнение; 2) Готовность; 3) Ожидание; 4) Создание; 5) Завершение.

Процесс, помимо главного рабочего состояния, может находиться в других состояниях, например, ожидания.

[](https://commons.wikimedia.org/wiki/File:Unix_processes_states.PNG?uselang=ru)

Рис.3 Граф переходов

Процесс в ОС [Linux](https://ru.wikipedia.org/wiki/Linux) может находиться в одном из следующих состояний:

R (running/runnable) — процесс исполняется либо ожидает своей очереди исполняться;

D — непрерываемый сон — процесс ожидает определённого события;

S — прерываемый сон — процесс ожидает определённого события либо сигнала;

T — остановка — процесс приостановлен, например, отладчиком;

Z ([zombie](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81-%D0%B7%D0%BE%D0%BC%D0%B1%D0%B8" \o "Процесс-зомби)) — процесс уже завершился, но ещё не передал родительскому процессу свой код возврата.

**Реализация процессов.**

Для реализации модели процессов ОС содержит массив структур, называемый таблицей процессов, с одним элементом для каждого процесса (блоком управления процессом). Элементы таблицы содержат информацию о состоянии процесса, счетчике команд, указателе стека, распределении памяти, а также всю остальную информацию, которую необходимо сохранять при переключении в состояние готовности или блокировки для последующего запуска.