Каждый процесс Linux выполняют одну программу и изначально получает один поток управления. У процесса есть счетчик команд, который отслеживает следующую исполняемую команду. Linux позволяет процессу создавать дополнительные потоки. Linux представляет собой многозадачную систему, и несколько независимых процессов могут работать одновременно.

На рабочих станциях также работают десятки фоновых **процессов-демонов** (daemons).Они запускаются при загрузке системы из сценария оболочки.

Например, демон cron позволяет планировать активность на минуты, часы, месяцы вперед. В назначенное время, например, запускает требуемую программу (в виде нового процесса).

Процессы Linux создаются с пом. системного вызова **fork**. Новый созданный процесс- точная копия родительского, называется дочерним. У родительского и дочернего процессов есть собственные приватные образы памяти. Если родительский процесс изменяет какие-либо данные впоследствии, то эти изменения невидимы для дочернего (и наоборот).

В Linux процессы могут общаться между собой посредством некой формы передачи сообщений. Можно создать канал между двумя процессами, в который один процесс пишет поток байтов, а другой - его читает. Эти каналы называются **трубами (pipes)**. Синхронизация процессов достигается путем блокирования процесса при попытке прочитать данные из пустого канала. Когда данные появляются в канале, процесс разблокируется.

При помощи каналов организуются конвейеры оболочки. Когда оболочка видит строку вроде

Sort <f| head

Она создает 2 процесса sort и head , а также устанавливает между ними канал, так что выход программы sort направляется на вход программы head. Если канал переполняется, то система приостанавливает работу sort до тех пор, пока head не удалит из него какие-то данные.

Процессы могут общаться при помощи программных прерываний. Один процесс может послать другому **сигнал** (signal). Процессы могут сообщить системе, какие действия следует предпринимать, когда придет тот или иной сигнал. Варианты такие: проигнорировать сигнал, перехватить его, позволить сигналу убить процесс. Если процесс выбрал перехват посылаемых ему сигналов, то он должен указать процедуру обработки сигналов. В этом случае при поступлении сигнала, управление передается сразу же обработчику. Когда процедура обработки сигнала завершает свою работу, управление передается снова в то место, где оно находилось, когда пришел сигнал (это аналогично обработке аппаратных прерываний ввода-вывода). Процесс может посылать сигнал только членам своей **группы процессов**, состоящей из его родителя, дочерей, внуков и пр. Например, процесс выполняет вычисления с плавающей точкой и делит на ноль, то он получает сигнал SIGFPE (сигнал исключения при выполнении операции с плавающей точкой).

Рассмотрим некоторые системные вызовы для управления процессами.

Fork- создание процесса.

Waitpid(pid, &status, opts) – ожидает завершения дочернего процесса.

Kill(pid, sig) – послать сигнал процессу.

Pause () – приостановить процесс.

Exit(stasus) – завершить выполнение процесса и вернуть статус.

Execve(name, argv, envp) – заменить образ памяти процесса.

Итак, для ожидания завершения дочернего процесса родительский процесс делает системный вызов waitpid, который ждет завершения потомка (потомков). У waitpid 3 параметра. Первый – позволяет вызывающей стороне ждать конкретного потомка. Если он равен -1 , то waitpid ожидает завершения любого дочернего процесса. Второй параметр содержит статус завершения дочернего процесса (нормальное(0)/ненормальное(код ошибки)). Третий определяет, будет ли вызывающая сторона блокирована или сразу получит управление обратно (если ни один потомок не завершен).

Выполнение команды выполняется системным вызовом exec, который заменяет весь образ памяти содержимым файла, указанного в первом параметре.

Если дочерний процесс уже завершился, а родительский не ожидает этого события, то дочерний процесс переходит в состояние зомби - живого мертвеца, т.е. приостанавливается. Когда родительский процесс, наконец, обращается к процедуре waitpid, дочерний процесс завершается.

Системный вызов alarm (будильик) позволяет установить интервал времени, по истечении которого посылается сигнал SIGALARM.

Только один процесс в системе рождается особенным способом — init — он порождается непосредственно ядром при старте системы. Все остальные процессы появляются путём дублирования текущего процесса с помощью системного вызова fork. Сразу после выполнения fork процесс  переходит в состояние «готов».

Фактически, процесс стоит в очереди и ждёт, когда планировщик (scheduler) в ядре даст процессу выполняться на процессоре. Как только планировщик поставил процесс на выполнение, началось состояние «выполняется». Процесс может выполняться весь предложенный промежуток (квант) времени, а может уступить место другим процессам, воспользовавшись системным вывозом sched\_yield. Некоторые системные вызовы могут выполняться долго, например, ввод-вывод. В таких случаях процесс переходит в состояние «ожидает». Как только системный вызов будет выполнен, ядро переведёт процесс в состояние «готов».

В Linux также существует состояние «ожидает», в котором процесс не реагирует на сигналы прерывания. В этом состоянии процесс становится «неубиваемым», а все пришедшие сигналы встают в очередь до тех пор, пока процесс не выйдет из этого состояния.

Ядро само выбирает, в какое из состояний перевести процесс. Чаще всего в состояние «ожидает (без прерываний)» попадают процессы, которые запрашивают ввод-вывод.

Состояние «остановлен»

В любой момент можно приостановить выполнение процесса, отправив ему сигнал SIGSTOP. Процесс перейдёт в состояние «остановлен» и будет находиться там до тех пор, пока ему не придёт сигнал продолжать работу (SIGCONT) или умереть (SIGKILL). Остальные сигналы будут поставлены в очередь.   
  
Завершение процесса

Ни одна программа не умеет завершаться сама. Они могут лишь попросить систему об этом с помощью системного вызова exit или быть завершенными системой из-за ошибки.

Хотя аргумент системного вызова принимает значение типа int, в качестве кода возврата берется лишь младший байт числа.