Процессы

Обычно *программой* называют совокупность файлов.
Это может быть набор исходных текстов, объектных файлов или собственно выполняемый файл.

Для того чтобы программа могла быть запущена на выполнение, система сначала должна создать
окружение (*environment*) или среду выполнения задачи,
куда относятся ресурсы памяти, возможность доступа к устройствам ввода/вывода и различным системным ресурсам, включая услуги ядра.

Это окружение (среда выполнения задачи) получило название *процесса*.
Можно представить процесс как совокупность данных ядра системы, необходимых для описания образа программы в памяти и управления ее выполнением.
Можно также представить процесс, как программу в стадии ее выполнения.

Процесс состоит из инструкций, выполняемых процессором,
данных и информации о выполняемой задаче, такой как размещенние в памяти, открытые файлы и статус процесса.

Linux многозадачная система, поэтому одновременно исполняется множество процессов,
каждый из которых следует собственному набору инструкций,
не передавая управление набору инструкций другого процесса.
Процесс считывает и записывает информацию
в *свой сегмент данных и в стек*,
но ему недоступны данные и стеки других процессов
(процессы *изолированы* друг от друга).

Для обмена данными и синхронизации в Linux существует набор средств, образующих *систему межпроцессного взаимодействия*.
Это такие средства как :
каналы (pipes),
разделяемая память (shared memory),
семафоры (semaphores),
сигналы (signals),
очереди сообщений сообщения (message queue),
сокеты (sockets) и др.

Типы процессов

Системные процессы являются частью ядра и
всегда расположены в оперативной памяти.
Системные процессы запускаются особым образом при
инициализации ядра системы.
Выполняемые инструкции и данные этих процессов
находятся в ядре системы.
Они могут вызывать функции и обращаться к данным,
недоступным для остальных процессов.
Системными процессами являются, например:
shed (диспетчер свопинга),
vhand (диспетчер страничного замещения),
kmadaemon (диспетчер памяти ядра)
bdflush (диспетчер буферного кэша) и др.

К системным процессам следует отнести процесс *init*,
являющийся прародителем всех остальных процессов в UNIX.
Хотя *init* не является частью ядра,
и его запуск происходит из исполняемого файла (/etc/init),
его работа жизненно важна для функционирования всей системы в целом.

Демоны - это неинтерактивные процессы, которые запускаются
путем загрузки в память соответствующих им программ
(исполняемых файлов), и выполняются в *фоновом (background)* режиме.
Демоны обычно запускаются при инициализации системы
(но после инициализации ядра) и обеспечивают работу различных
подсистем (например, системы печати, системы сетевого доступа и сетевых
услуг, системы терминального доступа, и т. п.)
Демоны не связаны ни с одним пользовательским сеансом работы и не
могут непосредственно управляться пользователем.
Большую часть времени демоны ожидают
пока тот или иной процесс запросит определенную услугу, например,
доступ к файловому архиву, печать документа и т.д.

К *прикладным процессам* относятся
все остальные процессы, выполняющиеся в системе.
Как правило, это процессы, порожденные
в рамках *пользовательского сеанса* работы.
Например, запуск команды ***ls*** породит соответствующий процесс.
Важнейшим пользовательским процессом является основной командный
интерпретатор (*login bash)*, который обеспечивает
работу пользователя.
Он запускается сразу же после регистрации в системе,
а завершение работы *login shell* приводит к отключению от системы.
Пользовательские процессы могут выполняться
как в *интерактивном* (foreground),
так и в *фоновом (background)*
режимах, но в любом случае время их жизни (и выполнения)
ограничено сеансом работы пользователя.

При выходе из системы все
пользовательские процессы будут уничтожены.
Интерактивные процессы монопольно владеют терминалом, и пока такой процесс не завершит свое выполнение, пользователь не сможет работать с другими приложениями

Атрибуты процесса

Процесс имеет несколько атрибутов, позволяющих управлять его работой,

Каждый процесс имеет уникальный идентификатор (*Process ID)* ***PID***, позволяющий ядру системы различать процессы.
Когда создается новый процесс,
ядро присваивает ему следующий свободный
(т. е. не ассоциированный ни с каким процессом) идентификатор.
Присвоение идентификаторов происходит по возрастающий,
т. е. идентификатор нового процесса больше,
чем идентификатор процесса, созданного перед ним.
Если идентификатор достиг максимального значения,
следующий процесс получит минимальный свободный *PID*
и цикл повторится.
Когда процесс завершает свою работу,
ядро освобождает занятый им идентификатор.

Атрибутом процесса является также идентификатор
родительского процесса (*Parent Process ID)* ***PPID*** *,*
*то есть и*дентификатор процесса, породившего данный процесс.

Реальным идентификатором пользователя ***UID*** данного процесса
является идентификатор пользователя, запустившего этот процесс.
Эффективный идентификатор ***EUID*** служит для определения
прав доступа процесса к системным ресурсам
(в первую очередь к ресурсам файловой системы).

Обычно реальный и эффективный идентификаторы эквивалентны,
т. е. процесс имеет в системе те же права, что и пользователь,
запустивший его.

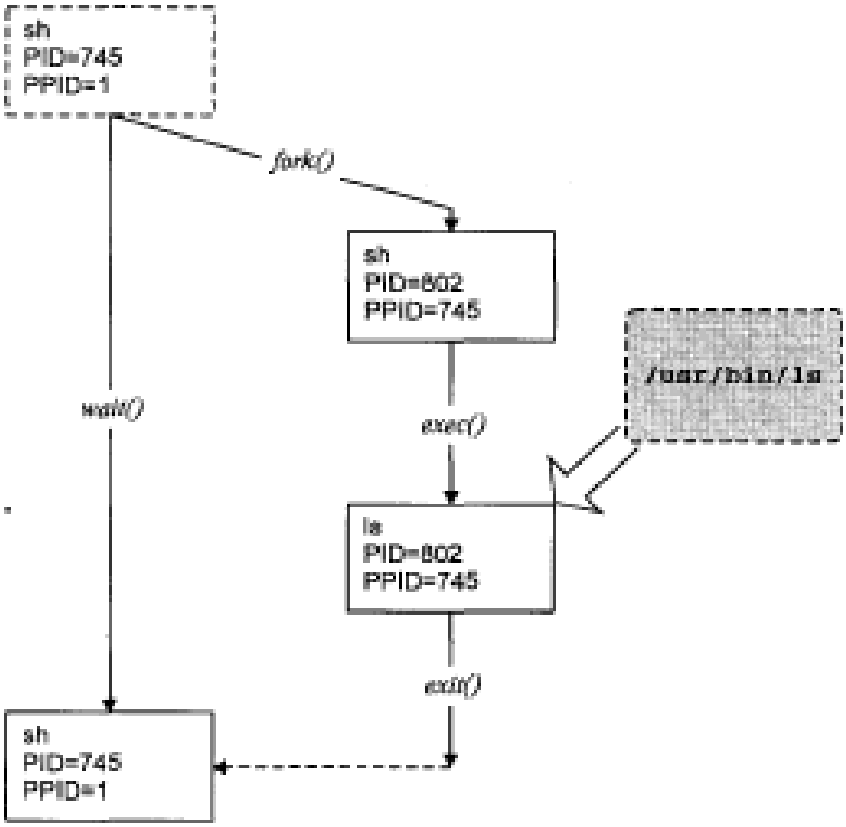
Однако существует возможность задать процессу более широкие права,
чем права пользователя путем установки флага *SUID*
*(*когда эффективному идентификатору присваивается значение
идентификатора владельца исполняемого файла, например,
администратора).

Аналогично идентификаторам пользователя с процессом ассоциируются
реальный и эффективный идентификаторы группы ***GID, EGID*** *.*

Статический приоритет или *nice-приоритет* лежит в диапазоне от -20 до 19, по умолчанию используется значение 0.
Значение –20 соответствует наиболее высокому приоритету,
nice-приоритет не изменяется планировщиком,
он наследуется от родителя или его указывает пользователь.

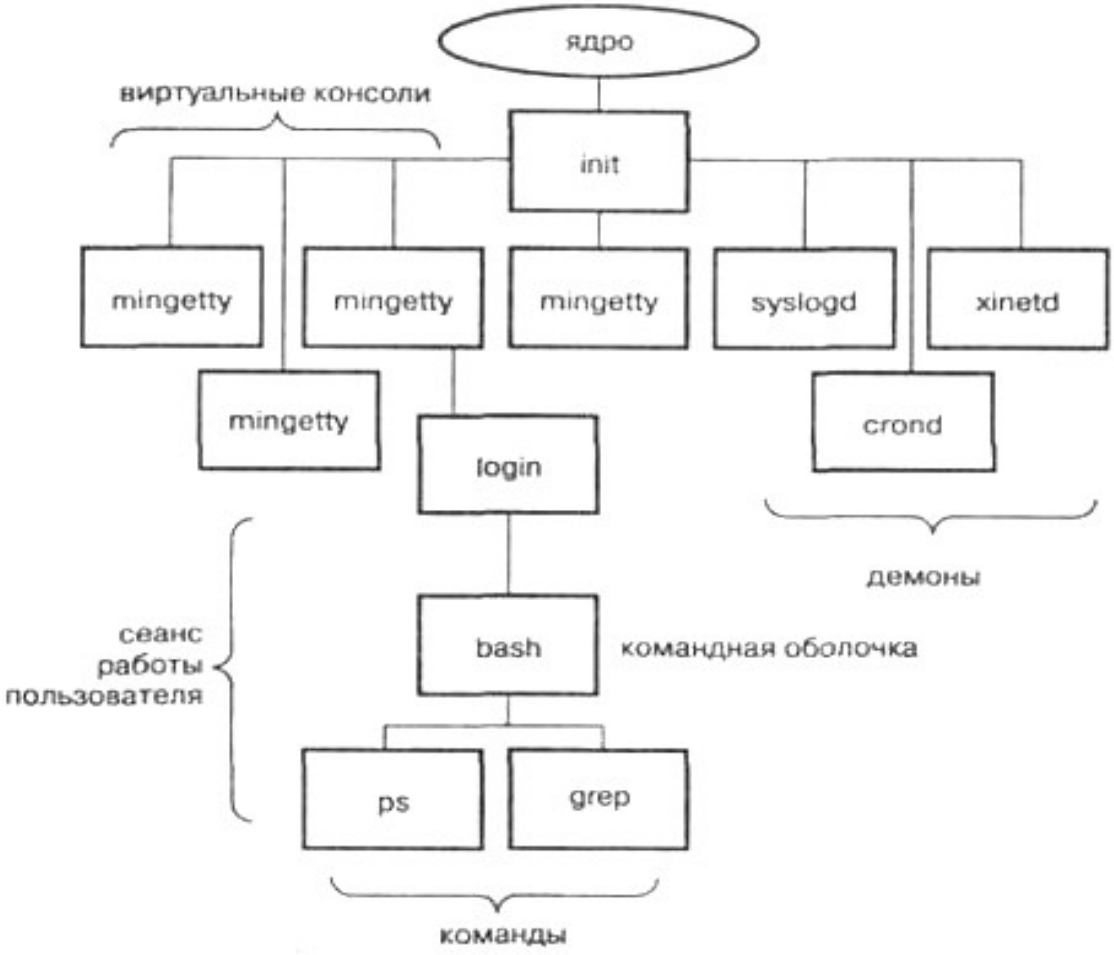
Динамический приоритет используется планировщиком для планирования выполнения процессов. Этот приоритет хранится в поле *prio* структуры t*ask_struct* процесса.
Динамический приоритет вычисляется исходя из значения параметра *nice* для данной задачи путем вычисления надбавки или штрафа, в зависимости от интерактивности задачи.
Пользователь имеет возможность изменять только статический приоритет процесса. При этом повышать приоритет может только *root*.
Существуют две команды управления приоритетом процессов:
nice и ***renice***.

Запуск процесса



Иерархия процессов

В Linux реализована четкая иерархия процессов в системе.
На этапе запуска системы происходит предопределенная цепочка запусков определенных процессов.
Фрагмент иерархии процессов:



Задачей i*nit*-а является запуск всего остального нужным образом.
Init читает файл */etc/inittab*, в котором содержатся инструкции для дальнейшей работы.
Первой инструкцией, обычно, является запуск *скрипта инициализации /etc/init.d* .
Здесь происходит проверка и монтирование файловых систем, установка часов системного времени, включение своп-раздела, присвоение имени хоста и т.д.
Далее будет вызван следующий скрипт, который переведет нас на "уровень запуска" по умолчанию.

Это подразумевает просто некоторый набор демонов, которые должны быть запущены:

/etc/init.d/syslogd – скрипт, отвечающий за запуск и остановку *системного логгера* (система журнальной регистрации событий SYSLOG, записывает системные сообщения в файлы журналов */var/log*).

Xined – демон Интернет-служб, управляет сервисами для интернета. Демон прослушивает сокеты и если в каком-то из них есть сообщение определяет какому сервису принадлежит данный сокет и вызывает соответствующую программу для обработки запроса.

crond – демон читает пользовательские файлы *расписаний* из каталога */var/spool/cron/* и общесистемное расписание, хранящееся в файле */etc/crontab* и файлах, расположенных в каталоге */etc/cron.d/*.
После того, как расписания загружены в оперативную память, *crond* ежеминутно проверяет наличие записи в расписании, соответствующей текущему времени, и,
если такая найдена, запускает указанную команду
от имени указанного пользователя.
отвечает за просмотр файла crontab и выполнение, внесенных в него команд в указанное время для опредленного пользователя.

Последним важным действием *init* является
запуск некоторого количества *getty (get teletype),*
управляющих доступом к физическим и виртуальным терминалам (tty).
mingetty – виртуальные терминалы, назначением которых
является слежение за консолями пользователей.
mingetty запускает программу *login*
– начало сеанса роботы пользователя в системе.
Задача *login*-а – регистрация пользователя в системе.
После успешной регистрации пользователя чаще всего грузиться командный интерпретатор пользователя (*shell*), например, *bash*.

Состояния процессов

Для практических занятий

ps - Вывести список процессов
top - Интерактивно наблюдать за процессами
uptime - Посмотреть загрузку системы
w - Вывести список активных процессов для всех пользователей
free - Вывести объем свободной памяти
pstree - Отображает все запущенные процессы в виде иерархии