**Лабораторная работа № 4**

**ПРОЦЕССЫ В LINUX**

**Часть 1**

**УПРАВЛЕНИЕ ПРОЦЕССАМИ**

**1. Цель работы**

Целью работы является изучение методов управления и создания пользовательских процессов в ОС Linux.

**2. Теоретическая часть**

***Изучаются команды:***

* ps – запрос информации о процессах текущего терминала;
* & – запуск фонового процесса;
* fg, bg – переводит процесс в активный или фоновый режим;
* jobs – запрос листинга списка заданий;
* nohup – защита фоновых процессов от прерывания выполнения при выходе из сеанса работы с системой;
* nice – понижение приоритета процесса;
* kill – прекращение выполнения процесса.

Процессом называется программа в стадии выполнения. Управление процессами – задача ядра операционной системы. Для поддержания информации о процессах ядро использует специальную структуру – таблицу процессов. Каждому процессу назначается уникальное число-идентификатор процесса, которое используется ядром для однозначного определения того, какому процессу соответствует запись в таблице процессов.

В ОС UBUNTU между процессами поддерживаются отношения «родитель» – «потомок». Каждый процесс может породить произвольное число процессов-потомков, но каждый процесс имеет только одного родителя, и нет ни одного процесса, который не имел бы родителя. Самый первый процесс, запускаемый ядром в ходе загрузки системы, называется init. Init служит для управления процессом загружки, в том числе запуска процессов-демонов. Демоны, также известные как системные сервисы, – это процессы, производящие неинтерактивную обработку данных пользователя, т.е. процесс-демон не подключен к терминальному устройству. Примером процессов-демонов могут служить серверы протоколов НТТР и FТР httpd и ftpd или сервер системного журнала syslogd.

Часто имена исполняемых файлов программ-демонов заканчиваются буквой d. Процессу init назначается идентификатор 1, первому порожденному init процессу – 2 и так далее. В ОС UBUNTU для идентификатора процесса общеупотребительным является сокращение PID.

После того как процесс порожден, т.е. ему отведена запись в системной таблице процессов, он находится в состоянии готовности и может быть выбран планировщиком процессов операционной системы для выполнения. При выполнении операций ввода – вывода и в ряде других случаев процесс может перейти в состояние ожидания некоторого события; тогда до наступления этого события или поступления в процесс некоторого прерывания он не будет выбираться для выполнения. Кроме процессов, находящихся в состояниях готовности и ожидания, в UBUNTU могут присутствовать остановленные процессы и процессы-зомби. Останов и возобновление выполнения процессов мы обсудим позднее, а процессов-зомби коснемся в ходе рассмотрения системных вызовов ОС UBUNTU.

Для вывода списка запущенных процессов: ps

Команда ps принимает следующие ключи:

-а – вывод информации о процессах, запущенных от имени всех пользователей, а не только текущего;

-х – вывод информации о процессах, неподключенных к терминальным устройствам, например, о процессах-демонах;

-u – вывод информации о ресурсах, используемых процессами;

-m – сортировка списка по объему используемой процессам оперативной памяти;

-r - сортировка списка по используемому процессом процессорному времени;

-U <имя пользователя> – вывод процессов, запущенных от имени указанного пользователя;

-p <список идентификаторов процессов> – вывод информации только об указанных процессах; идентификаторы в списке разделяются запятой.

Выводимый список включает несколько столбцов; эти столбцы содержат следующую информацию:

USER – имя пользователя, запустившего процесс;

PID – идентификатор процесса;

%CPU – процент процессорного времени, потребляемый процессом;

%МЕМ – процент объема памяти, занятой процессом;

СОММАND – команда, использованная для запуска процесса;

ТТ – терминальное устройство, к которому подключен процесс;

VSZ – объем виртуальной памяти, затребованный процессом, в блоках по 1024 байта;

RSZ – реальный объем памяти, используемый процессом, в блоках по 1024 байта;

START – время запуска процесса.

STAT – информация о состоянии процесса; первая буква означает состояние процесса:

S – процесс находится в состоянии ожидания менее 20 секунд;

I – процесс находится в состоянии ожидания 20 секунд или более; К – процесс находится в состоянии готовности; Т – процесс остановлен; Z – процесс-зомби.

За первой буквой может также быть указана дополнительная информация. Например, команда:

рs -u -r -х

может быть использована для вывода подробной информации обо всех процессах, запущенных от имени текущего пользователя, причем список процессов будет упорядочен по степени использования процессорного времени.

*Для наблюдения за процессами системы:* top

программа top выводит таблицу, в которой каждому процессу соответствует одна строка. Сортировкой строк можно управлять, *а по умолчанию критерием для сортировки является идентификатор процесса.* В отличие от программ, с которыми мы имели дело до этого, top не возвращает управление пользователю сразу после печати результата. Вместо этого через некоторые заданные промежутки времени программа собирает информацию о процессах системы и снова выводит ее на экран. Для выхода из программы следует нажать на клавишу [Q].

Программа принимает следующие параметры:

-о<параметр> – указывает показатель для сортировки строк с информацией о процессах; по умолчанию сортировка производится по убыванию; для того, чтобы сортировка проводилась по возрастанию, перед названием показателя следует поместить знак ’+ ‘ не отделяя его пробелами ни от ключа, ни от названия показателя; среди показателей наиболее интересны следующие: command – команда, которой был запущен процесс; cpu – степень использования процессорного времени; pid – идентификатор; time – время выполнения процесса;

rsize – реальный используемый процессом объем памяти; vsize – объем виртуальной памяти, используемый процессом;

-s <число> – указывает задержку в секундах, между обновлением информации в таблице; по умолчанию 1 секунда;

-n <число> – указывает число процессов, о которых будет выведена информация.

Например, следующая команда: top -n 10 -s 3 -о cpu

выведет таблицу из не более чем 10 строк, в которой процессы будут отсортированы по убыванию используемого ими процессорного времени, причем обновляться таблица будет каждые 3 секунды.

Заголовок таблицы содержит следующие обозначения столбцов для показателей:

PID – идентификатор процесса;

ТIМЕ - процессорное время, потребленное процессом с момента запуска;

%СРU – процент процессорного времени, потребляемого процессом;

USERNAME – пользователь, от имени которого выполняется процесс.

Одним из способов взаимодействия между процессами в UBUNTU являются сигналы; сигналы – это сообщения, которые могут быть посланы процессу при помощи системного вызова. Каждому виду сигналов в системе соответствует название и номер.

Многие сигналы имеют специальный смысл. В частности, сигнал SIGTERM уведомляет процесс о необходимости завершения, сигнал SIGTERM используется для безусловного завершения процесса, сигналы SIGSTOP и SIGCONT соответственно останавливают и возобновляют выполнение процесса и т.д.

*Для того чтобы послать сигнал процессу из программы оболочки:*

kill [-s <название сигнала>] <список идентификаторов процессов>

Например, чтобы остановить выполнение некоторого процесса, с идентификатором 123, можно воспользоваться следующей командой:

kill -s SIGSTOP 123

Для его возобновления, можно применить такую команду:

kill -s SIGCONT 123

Если при вызове kill не указать название сигнала, то будет послан сигнал SIGTERM. Так, для завершения процесса с идентификатором 123 можно использовать команду:

kill 123

Для вывода всех сигналов, поддерживаемых системой, служит ключ -l:

kill -l

Программа-оболочка позволяет запускать программы в фоновом режиме. Сразу после запуска программы, указанной в команде запуска в фоновом режиме, пользователь снова получает возможность вводить новые команды в программе-оболочке, в то время как запущенная программа продолжает выполняться.

Для запуска программы в фоновом режиме команда завершается символом амперсанда ‘&'. Например, команда:

. /loop.sh &

запустит на выполнение файл сценария loop.sh в фоновом режиме.

Протяженность времени, отводимого планировщиком операционной системы процессу, зависит от приоритета процесса. Чем выше приоритет процесса, тем большую часть процессорного времени он будет получать во время выполнения. Приоритет процесса выражается числом. Чем меньше число, тем выше приоритет. По умолчанию процессы запускаются с приоритетом породившего их процесса.

*Для запуска процесса со значением приоритета, отличного от приоритета процесса-предка:*

nice [-n <число> ] <команда>

При запуске процесса значение приоритета будет откорректировано, исходя из переданного числа. Если ключ -n не был введен в команде, то используется число 10. Чем выше число, тем меньший приоритет получит процесс. Число, передаваемое в составе ключа –n, может лежать в пределах от –20 до 20, но пользователям системы, не обладающим администраторскими полномочиями, как правило, не позволяется задавать отрицательные значения, т.е. рядовой пользователь не может повысить приоритет процесса.

*Для изменения приоритета после запуска процесса:* renice -n <число> <список идентификаторов процессов>

Программа renice работает аналогично nice с той разницей, что изменяется не приоритет создаваемого процесса, а приоритеты всех запущенных процессов, идентификаторы которых входят в разделенный пробелами список, передаваемый команде в качестве параметра. Число в составе ключа -n прибавляется к текущему приоритету процесса.

При вводе команды в программе-оболочке вне зависимости о того, запускается ли программа в фоновом режиме или нет, программа-оболочка порождает дочерний процесс для исполнения программы самим процессом программы-оболочки:

ехес <команда>

Рассмотрим следующий сценарий:

#!/bin/sh ps ехес *./lоор.sp*

Пусть, он сохранен под именем ехес.sh.

./ехеc.sh &

ps

Первая команда запускает на выполнение сценарий ехес.sh, что приводит к порождению нового процесса. В этом можно убедиться, изучив результат вызова ps, который производится этим сценарием. После этого сценарий запускает уже рассмотренный нами сценарий 1оор.sh. Вторая команда из приведенной выше пары вызывает рs для выдачи списка запущенных процессов. Обратите внимание на то, что нового процесса для выполнения сценария lоор.sh создано не было – его выполняет тот же процесс, который был порожден для выполнения сценария ехес.sh.

В качестве еще одного примера, рассмотрим следующую команду: ехес ls

Результат ее выполнения – завершение работы программы-оболочки – так как процесс, выполнявший код программы-оболочки получил инструкцию выполнять код программы ls, то завершение кода программы ls, наступающее после вывода листинга текущего каталога, приведет к завершению выполняющего ее процесса. Таким образом, процесс, изначально выполнявший код программы-оболочки, будет завершен.

### Методика выполнения работы

1. Вывести на экран листинг характеристик (в длинном и коротком форматах) процессов, инициализированных с вашего терминала. Проанализировать и объяснить содержание каждого поля сообщения.
2. Разработать и запустить простейшую процедуру в фоновом режиме с бесконечным циклом выполнения, предусматривающую, например, перенаправление вывода каких-то сообщений в файл или в фиктивный файл, и использующую команду sleep для сокращения частоты циклов процедур.
3. Выполнить п. 1. Объяснить измерения в листинге характеристик процессов. Объясните содержание PID и PPID.
4. Понизите значение приоритета процедуры. На что и как повлияет эта операция при управлении вычислительным процессом системы? Как отразятся ее результаты в описателях процессов?
5. Проанализируйте листинг процессов. Какой процесс является родительским для процедуры?
6. Выйдите из системы и войдите заново. Проанализируйте листинг процессов. Объясните изменения в системе.