Лабораторная работа
Команды для управления процессами.

1 Процессы

Процесс (process) — блок адресного пространства в котором выполняются одна или более нитей, экземпляр выполняемой программы. Любой процесс может запускать другие процессы. Таким образом, процессы в среде UNIX образуют иерархическую структуру. На вершине этой структуры находится процесс init, являющийся предком всех остальных процессов.

1.1 Атрибуты процессов

С каждым процессом связан набор атрибутов, которые помогают системе контролировать выполнение процессов и распределять между ними ресурсы системы.

Идентификатор процесса (process ID)
: это целое число однозначно идентифицирующее процесс. Процесс с идентификатором 1 это процесс init.

Идентификатор родительского процесса (parent process ID)
: указывает на родительский процесс.

Идентификатор группы процессов (process group ID).
: Процессы могут объединяться в группы. Каждая группа обозначается идентификатором группы. Процесс, идентификатор которого совпадает с идентификатором группы, называется *лидером* группы.

Идентификатор сеанса (session ID).
: Каждая группа процессов принадлежит к сеансу. Сеанс связывает процессы с *управляющим терминалом*. Когда пользователь входит в систему, все создаваемые им процессы будут принадлежать сеансу, связанному с его текущим терминалом.

Программное окружение (programm environment)
: это просто набор строк, заканчивающихся нулевым символом. Строки называются переменными окружения и имеют следующий формат:

```
имя переменной = значение переменной
```

Дескрипторы открытых файлов.
: Дескриптор файла — некоторое число, которое используется для обращения к файлу. При запуске процесс наследует дескрипторы от родительского процесса.

Текущий рабочий каталог
: это каталог от которого система производит разрешение относительных имен.

Текущий корневой каталог
: это каталог от которого производится разрешение абсолютных имен. Процесс не имеет доступа к файлам находящимся выше корневого каталога.

Идентификаторы пользователя и группы.
: С каждым процессом связаны действительные идентификаторы пользователя (real user ID) и группы (real group ID), совпадающие с соответствующими идентификаторами пользователя, запустившего процесс. Кроме того, с процессом связаны эффективные идентификаторы пользователя (effective user ID) и группы, определяющие права процесса в системе. Обычно, действительные и эффективные идентификаторы совпадают.

Приоритет (nice).

Значение nice ("дружелюбность") показывает готовность процесса уступить свое процессорное время другим процессам. Чем больше значение nice, тем ниже приоритет процесса.

2 Основные сведения о работе с процессами

Основным средством для создания процессов является системный вызов *fork*. При выполнении данного вызова ядро создает новый процесс, который является копией процесса вызвавшего fork. Созданный процесс называется *дочерним*, а процесс осуществивший вызов fork — *родительским*. В дочернем процессе вызов возвращает значение ноль, а в родительском он возвращает идентификатор дочернего процесса. Дочерний процесс наследует дескрипторы открытых файлов и значения переменных окружения родительского процесса.

Другой системный вызов для работы с процессами — *exec*. Он позволяет сменить выполняемую программу. Вызову exec передаются в качестве аргументов имя программы которую надо выполнить и список ее аргументов. При выполнении вызова в пространство памяти вызывающего процесса загружается новая программа, которая запускается с начала. При выполнении вызова exec дескрипторы открытых файлов сохраняют свое значение.

Для завершения процесса используется системный вызов exit. Вызов имеет целочисленный аргумент называемый *кодом завершения* процесса. Как правило при успешном завершении процесса код завершения равен нулю, а в случае возникновения ошибки отличен от нуля. Родительский процесс может получить статус завершения дочернего процесса выполнив системный вызов *wait* или *waitpid*.

3 Механизмы межпроцессного взаимодействия

UNIX имеет большое число механизмов межпроцессного взаимодействия. Наиболее популярными средствами являются *сигналы*, *программные каналы* (pipes) и *именованные каналы* (FIFO).

3.1 Сигналы

Сигналы обеспечивают простой метод прерывания работы процессов. Сигналы используются в основном для обработки исключительных ситуаций. Процесс может определять действия выполняемые при поступлении сигнала, блокировать сигналы, посылать сигналы другим процессам. Существует более двадцати различных сигналов. Основные:

`SIGCHLD`
: сигнал о завершении дочернего процесса.

`SIGHUP`
: сигнал освобождения линии. Посылается всем процессам, подключенным к управляющему терминалу при отключении терминала. Многие демоны при получении данного сигнала заново просматривают файлы конфигурации и перезапускаются.

`SIGINT`
: сигнал посылается всем процессам сеанса, связанного с терминалом, при нажатии пользователем клавиши прерывания (CTRL-C).

`SIGTERM`
: сигнал приводит к немедленному прекращению работы получившего сигнал процесса.

`SIGKILL`
: сигнал приводит к немедленному прекращению работы получившего сигнал процесса. В отличие от `SIGTERM` процесс не может блокировать и перехватывать данный сигнал.

`SIGSEGV`
: сигнал посылается процессу, если тот пытается обратиться к неверному адресу памяти.

`SIGSTOP`
: сигнал приводящий к остановке процесса. Для отправки сигнала SIGSTOP активному процессу текущего терминала можно воспользоваться комбинацией клавиш (CTRL-Z).

`SIGCONT`
: сигнал возобновляющий работу остановленного процесса.

`SIGUSR1,SIGUSR2`
: сигналы определяемые пользователем.

Для того, чтобы отправить процессу сигнал можно использовать команду `kill`. Для того, чтобы процесс мог отправить сигнал другому процессу необходимо чтобы эффективные идентификаторы пользователя у посылающего процесса и у процесса получателя совпадали. Процессы с эффективным идентификатором пользователя равным нулю могут посылать сигналы любым процессам.

3.2 Каналы

Часто возникает ситуация когда два процесса последовательно обрабатывают одни и те же данные. Для обеспечения передачи данных от одного процесса к другому в подобных ситуациях используются программные каналы. Программный канал (pipe) служит для установления связи, соединяющей один процесс с другим. Запись данных в канал и чтение из него осуществляются при помощи системных вызовов write и read, т.е. работа с каналами аналогична работе с файлами. Для создания программного канала используется системный вызов *pipe*. Вызов возвращает два дескриптора файлов, первый из которых открыт для чтения из канала, а второй для записи в канал.

Каналы используются, например, при организации конвейера. При выполнении команды:

```
find /usr/bin -name a* | sort
```

создается канал, команда find выводит в него результаты своей работы, а команда sort считывает из этого канала данные для сортировки.

Главным недостатком программных каналов является то, что они могут использоваться только для связи процессов имеющих общее происхождение (напр., родительский процесс и его потомок). Другой недостаток ограниченное время существования канала (программные каналы уничтожаются после завершения обращающегося к ним процесса).

Именованные каналы идентичны программным в отношении записи и чтения данных, но они являются объектами файловой системы. Именованный канал имеет имя, владельца и права доступа. Открытие и закрытие именованного канала осуществляется как открытие и закрытие любого файла, но при чтении и записи он ведет себя аналогично каналу.

Для создания именованного канала используется команда `mkfifo`. Если некоторый процесс открывает именованный канал для записи, то этот процесс блокируется до тех пор, пока другой процесс не откроет этот канал для чтения, и наоборот.

4 Команды для работы с процессами

`ps [-axewjlu] [-o` *формат*`] [-U` *пользователь*`] [-p` *pid*`]`
Выводит список и статус процессов работающих в системе. Без аргументов выводит список процессов текущего пользователя, подключенных к терминалу. Значения параметров следующие:

`-a` вывести информацию о процессах всех пользователей.

`-x` вывести информацию о процессах не подключенных к терминалу.

`-e` вывести значения переменных окружения процесса.

`-w` использовать строки длиной 132 символа. Если указан несколько раз, то строки не обрезаются совсем.

`-j, -l, -u`
: меняют формат вывода информации.

`-o` *формат*
: вывести информацию в указанном формате.

`-U` *пользователь*
: вывести информацию о процессах указанного пользователя.

`-p` *pid* вывести информацию о процессе с указанным идентификатором.

Значение формата для параметра `-o` является списком из следующих ключевых слов разделенных запятыми (без пробелов):

`command`
: командная строка и аргументы.

`nice` уровень nice (приоритет).

`pgid` идентификатор группы процессов.

`pid` идентификатор процесса.

`ppid` идентификатор родительского процесса.

`rgid, ruid`
: реальные идентификаторы группы и пользователя.

`uid` реальный идентификатор пользователя.

`tty` управляющий терминал

Для различных систем параметры и ключевые слова могут сильно различаться. Подробности об использовании `ps` на конкретной системе можно получить при помощи команды `man ps`.

`kill [-s` *signal*`| -`*signal*`]` *pid*
Посылает сигнал указанному процессу. Если значение сигнала опущено, предполагается SIGTERM. `signal` — символическое имя сигнала без префикса SIG, либо номер сигнала. Пример:
`kill -HUP 172` — послать сигнал SIGHUP процессу с идентификатором 172.

`nice [-`*nice*`]` *команда* `[`*аргументы*`]`
Выполняет команду с меньшим приоритетом. Если *nice* не задан, то предполагается 10. Значение *nice* может быть от -20 (наивысший приоритет) до 20 (наименьший приоритет). Отрицательные числа задаются как –nice. Увеличение приоритета может выполнить только суперпользователь. Пример:
`nice -10 john users` — запустить программу john с пониженным приоритетом.

`mkfifo [-m` *режим_доступа*`]` *имя*
Создает именованный канал с указанным именем и режимом доступа.

`tty` Выводит имя текущего терминала.

`who [am i]` Выводит список пользователей работающих в системе.

`uname [-amnrsv]` Выводит информацию о системе.

`uptime` Выводит время работы системы и ее среднюю загрузку за последние 5, 10 и 15 минут.

5 Средства оболочки предназначенные для работы с процессами

Список — последовательность из одного или более конвейеров разделенных операторами `;`, `&`, `&&` или `||`. Более высокий приоритет у операторов `&&` и `||`. Если команда завершается оператором &, то оболочка выполняет ее в фоновом режиме. Если между двумя командами стоит оператор &&, то вторая команда будет выполнена только в том случае, если первая завершится успешно. Если между двумя командами стоит `||`, то вторая команда будет выполнена только в том случае, если код завершения первой команды отличен от нуля. Если команды разделены точкой с запятой, то вторая команда будет выполнена после завершения первой, независимо от результата выполнения первой команды.

Оболочка содержит несколько встроенных команд для работы с процессами:

`wait [pid]`
: Ожидает завершения процесса с указаным идентификатором. Если идентификатор опущен, то ожидает завершения всех процессов запущеных оболочкой.

`exec` *команда* `[`*аргумент*`]...`
: Указанная команда заменяет оболочку и получает в качестве параметров заданные аргументы.

`exit [n]`
: Приводит к завершению оболочки с кодом завершения `n`. Если арумент опущен, то код завершения ноль.

`trap [`*действие условие*`...]`
: Устанавливает обработчик события. *Условие* либо EXIT, либо имя сигнала без префикса SIG. EXIT соответствует завершению работы оболочки. Если *действие* равно “-”, то обработчик сбрасывается в значение по умолчанию. Например, после выполнения команды:

  ```
  trap "echo PRESSED" INT
  ```

  оболочка будет выводить слово PRESSED после каждого нажатия клавиш CTRL-C. (Нажатие клавиш CTRL-C приводит к посылке сигнала SIGINT процессам подключенным к терминалу).

6 Практическое задание

1. Освойте работу с командой `ps`. Попробуйте запускать ее с различными аргументами. Если вывод команды не помещается на экране, используйте команду less.
2. Выведите в файл отчета[1] (lab4.txt) следующую информацию о запущенных Вами процессах: pid, ppid, tty, ruid, command. Вывод должен быть отсортирован по номеру процесса.
3. Выведите в файл отчета информацию о процессах запущенных пользователем root. Вывод должен быть отсортирован по номеру процесса.
4. Ключи -c, -v, -j, -u изменяют формат вывода команды. Попробуйте выполнить команду `ps` с каждым из этих ключей. Результат сохраните в файле отчета.
5. Запустите команду `/home/labs/back`. Выполните команду `ps` и найдите в выведенном списке процесс `/home/labs/back`. Отправьте процессу сигнал SIGTERM. Снова выполните ps.

[1] Если дано задание вывести что-либо в файл отчета, то это означает, что сначала надо вывести результат на экран, убедиться в отсутствии ошибок и только после этого повторно выполнить команду, перенаправив её вывод в файл отчета. Лишняя информация в файле отчета наказуема.

6. Запустите команду:

   ```
   /home/labs/looper first & nice /home/labs/looper secnd&
   ```

   Программа looper выполняет 100 000 000 пустых циклов, выводит первый аргумент и номер цикла, после чего все повторяется.

7. При помощи команды `ps`, запущеной на другом терминале, определите номера процессов `looper` и уничтожьте их.

8. Создайте в домашнем каталоге именованный канал fifo. Выполните команду

   ```
   cat /home/labs/dao.txt >fifo
   ```

   Теперь с другого терминала выполните команду

   ```
   cat fifo
   ```

9. При помощи команд tty, w, uname, uptime выведите в файл отчета (lab3.txt) имя текущего терминала, информацию о пользователях, работающих в системе, название и версию операционной системы, время работы системы.

10. Установите обработчик сигнала SIGINT. Для этого выполните команду:

    ```
    trap "echo иполучен сигнал SIGINT " INT
    ```

    Пошлите несколько раз оболочке сигнал SIGINT. Для этого следует нажать клавиши CTRL-C.

11. Добавьте в начало файла отчета строку с номером лабораторной работы и Вашим именем.