*Лабораторная работа № 7*

**Работа с процессами в операционной системе Linux с использованием С-программ**

**Цель работы:** получение навыков создания С-программ и их использования для работы с процессами в операционной системе Linux

**Задание 1**

**Компиляция и запуск C-программ**

Для компиляции программ в Linux мы будем применять компилятор gcc. Для того чтобы он нормально работал, необходимо, чтобы исходные файлы, содержащие текст программы, имели имена, заканчивающиеся на .c.

В простейшем случае откомпилировать программу можно, запуская компилятор командой gcc имя\_исходного\_файла.

Если программа была написана без ошибок, то компилятор создаст исполняемый файл с именем имя\_исходного\_файла.out. Изменить имя создаваемого исполняемого файла можно, задав его с помощью опции -o:

gcc имя\_исходного\_файла -o имя\_исполняемого\_файла

Компилятор gcc имеет несколько сотен возможных опций. Получить информацию о них вы можете в UNIXManual.

Обычно во всех версиях Linux имеется компилятор с именем cc, поддерживающий опцию –о.

Запустить программу на исполнение можно, набрав имя исполняемого файла и нажав клавишу <Enter>.

**Самостоятельная работа**

1. Перейти во вторую текстовую консоль.

<alt>+F2

2. Запустить редактор Vim

3. Создать файл, содержащий следующие строки:

#include <stdio.h> int main()

{

printf(“Добро пожаловать!\n”); Return 0; }

4. Сохраняем файл под именем quine.c

:w quine.c

5. Переключаемся на другую консоль.

<alt>+F3

6. Компилируем программу.

gcc quine.c -o binfile

В консоль не должно быть выведено никаких ошибок. Если ошибки появились, то возвращаемся в редактор по <Alt>+F2, проверяем правильность набора программы и исправляем ошибки.

7. Запускаем скомпилированный файл (не забываем прописать путь целиком, даже несмотря на то, что файл в текущей папке).

./binfile

Должна появиться строка «Добро пожаловать!»

**Задание 2**

**Процессы в операционной системе Linux**

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. Атрибуты процессов**

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

**Идентификатор процесса**  (processID) это целое число однозначно идентифицирующее процесс. Процесс с идентификатором 1 это процесс init.

**Идентификатор родительского процесса** (parentprocessID) указывает на родительский процесс.

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

**Идентификатор сеанса** (sessionID). 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** (programmenvironment) 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 об ращения 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 (realuserID) и группы (realgroupID), совпадающие с соответствующими идентификаторами пользователя, запустившего процесс. Кроме того, с процессом связаны эффективные идентификаторы пользователя (effectiveuserID) 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. Команды для работы с процессами**

**ps [-axewjlu] [-o формат] [-U пользователь] [-ppid]**

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.

- ppid - 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

- top - выводит сведения о запущенных процессах в динамике. Прекратить работукоманды можно нажав клавишу q

**Самостоятельная работа**

1. Освойте работу с командой ps. Попробуйте запускать ее с различными аргументами. Если вывод команды не помещается на экране, используйте команду less.

2. Запишите в отчет следующую информацию о запущенных Вами процессах: pid, ppid, tty, ruid, command. Вывод должен быть отсортирован по номеру процесса.

3. Ключи -c, -v, -j, -u изменяют формат вывода команды. Попробуйте выполнить команду ps с каждым из этих ключей. Результат запишите в отчет.

4. Выведите сведения о процессах в динамике. Запишите в отчет дополнительные сведения, которые были представлены командой top.

**Задание 3**

**Создание процесса в Linux. Системный вызов fork()**

В операционной системе UNIX новый процесс может быть порожден единственным способом – с помощью системного вызова fork(). При этом вновь созданный процесс будет являться практически полной копией родительского процесса. У порожденного процесса по сравнению с родительским процессом (на уровне уже полученных знаний) изменяются значения следующих параметров:

· идентификатор процесса – PID;

· идентификатор родительского процесса – PPID.

В процессе выполнения системного вызова fork() порождается копия родительского процесса и возвращение из системного вызова будет происходить уже как в родительском, так и в порожденном процессах. Этот системный вызов является единственным, который вызывается один раз, а при успешной работе возвращается два раза (один раз в процессе-родителе и один раз в процессе-ребенке)! После выхода из системного вызова оба процесса продолжают выполнение регулярного пользовательского кода, следующего за системным вызовом.

**Прогон программы с fork() с одинаковой работой родителя и ребенка**

Для иллюстрации сказанного давайте рассмотрим следующую программу:

/\* Программа – пример создания нового процесса с одинаковой работой процессов ребенка и родителя \*/

#include <sys/types.h> #include <unistd.h> #include <stdio.h>

int main()

{

pid\_ t pid, ppid; int a = 0; (void)fork();

/\* При успешном создании нового процесса с этого места псевдопараллельно начинают работать два процесса: старый и новый \*/

/\* Перед выполнением следующего выражения значение переменной a в обоих процессах равно 0 \*/

a = a+1;

/\* Узнаем идентификаторы текущего и родительского процесса (в каждом из процессов !!!) \*/

pid = getpid(); ppid = getppid();

/\* Печатаем значения PID, PPID и вычисленное значение переменной a (в каждом из процессов !!!) \*/

printf("My pid = %d, my ppid = %d, result = %d\n", (int)pid, (int)ppid, a);

return 0;

}

Наберите эту программу, откомпилируйте ее и запустите на исполнение. Проанализируйте полученный результат.

Изменение пользовательского контекста процесса. Семейство функций для системного вызова exec(). Для изменения пользовательского контекста процесса применяется системный вызов exec(), который пользователь не может вызвать непосредственно. Вызов exec() заменяет пользовательский контекст текущего процесса на содержимое некоторого исполняемого файла и

устанавливает начальные значения регистров процессора (в том числе устанавливает программный счетчик на начало загружаемой программы). Этот вызов требует для своей работы задания имени исполняемого файла, аргументов командной строки и параметров окружающей среды. Для осуществления вызова программист может воспользоваться одной из шести функций:

execlp(), execvp(), execl() и execv(), execle(), execve(), отличающихся друг от друга представлением параметров, необходимых для работы системного вызова exec().

**Прогон программы с использованием системного вызова exec().**

Для иллюстрации использования системного вызова exec() давайте рассмотрим следующую программу:

/\* Программа 03-2.с, изменяющая пользовательский контекст процесса (запускающая другую программу) \*/

#include <sys/types.h>

#include <unistd.h>

#include <stdio.h>

int main(int argc, char \*argv[], char \*envp[])

{

/\* Мы будем запускать команду cat c аргументом командной строки 03-2.с без изменения параметров среды, т.е. фактически выполнять команду "cat 03-2.c", которая должна выдать содержимое данного файла на экран. Для функции execle в качестве имени программы мы указываем ее полное имя с путем от корневой директории —/bin/cat. Первое слово в командной строке у нас должно совпадать с именем запускаемой программы. Второе слово в командной строке – это имя файла, содержимое которого мы хотим распечатать. \*/

(void) execle("/bin/cat", "/bin/cat", "03-2.c", 0, envp);

/\* Сюда попадаем только при возникновении ошибки \*/

printf("Erroronprogramstart\n");

exit(-1);

return 0; /\* Никогда не выполняется, нужен для того, чтобы компилятор не выдавал warning \*/

}

Откомпилируйте ее и запустите на исполнение. Поскольку при нормальной работе будет распечатываться содержимое файла с именем 03-2.c, такой файл при запуске должен присутствовать в текущей директории (проще всего записать исходный текст программы под этим именем). Проанализируйте результат.

**Задание 4**

**Команды для управления процессами.**

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

**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.

**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.

**Команды для работы с процессами**

* 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 manps.

**kill [-ssignal| -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 johnusers — 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 [ami]** 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.

**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 указанным идентификатором. Если идентификатор опущен, то ожидает завершения всех процессов запущенных 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).

**Самостоятельная работа**

1. Запишите в отчет информацию о процессах запущенных пользователем root. Вывод должен быть отсортирован по номеру процесса.

2. Создайте текстовый файл dao.txt, написав в него свою фамилию и имя.

3. Создайте в домашнем каталоге именованный канал fifo. Выполните команду cat /home/dao.txt>fifo. Теперь с другого терминала выполните команду cat fifo.

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

5. Установите обработчик сигнала SIGINT. Для этого выполните команду: trap "echo получен сигнал SIGINT " INT. Пошлите несколько раз оболочке сигнал SIGINT. Для этого следует нажать клавиши CTRL-C.