Лабораторная работа №2.
Управление процессами в ОС GNU/Linux.

1. Цель работы.

Изучить возможности, предоставляемые ОС GNU/Linux для создания и управления процессами.

2. Задание на лабораторную работу.

Ознакомиться с теоретическими сведениями. Исследовать примеры программ, приведенных в теоретической части. Написать, скомпилировать и отладить программу по заданию преподавателя. Составить отчет о выполненной работе.

3. Краткие теоретические сведения.

3.1. Дерево процессов. Идентификаторы процессов.

В загрузочном образе ОС GNU/Linux присутствует специальный процесс – init. В начале своей работы он считывает файл, сообщающий о количестве терминалов. Затем он разветвляется, порождая по процессу на терминал. Эти процессы ждут, пока кто-нибудь не зарегистрируется в системе. Процесс регистрации порождает оболочку для приема команд, которые могут породить другие процессы и т.д. Поэтому все процессы принадлежат единому дереву, в корне которого находится init. Получить дерево процессов можно при помощи команды pstree, рис. 1.

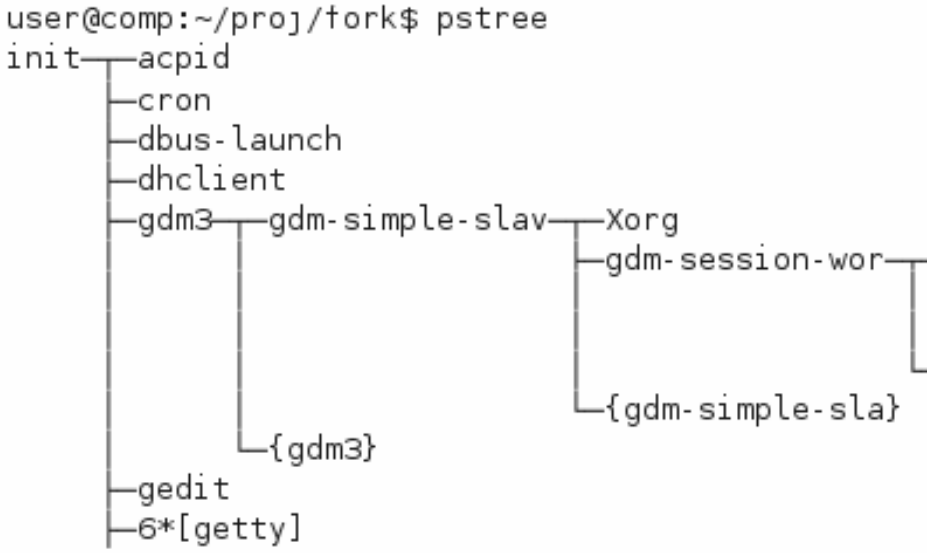

Рис. 1. Дерево процессов в ОС Debian GNU/Linux.

На рисунке видно, что главный процесс init запустил дочерние процессы: acpid, cron и другие. Один из дочерних процессов, gdm3 запустил, в свою очередь, два других дочерних процесса: gdm-simple-slav и {gdm3}. Процесс gdm-simple-slav также запустил несколько дочерних процессов и т.д. Таким образом при запуске команды pstree можно наблюдать дерево процессов, образованное от первичного процесса init.

Каждый процесс имеет уникальный идентификатор (PID). Функции

int getpid(); int getppid()

возвращают идентификаторы текущего и родительского процессов соответственно.

Пример. Программа выводит на дисплей PID собственный и родительского процесса.

```
/* Получение идентификатора процесса */
#include<stdio.h>
#include<unistd.h>
int main()
{
printf ("Process ID: %d\n", (int) getpid() );
printf ("Parent process ID: %d\n", (int) getppid());
getchar();
return 0;
}
```

Листинг 5. Результат работы программы.

```
user@comp:~/proj/fork$ ./getpid
Process ID: 2302
Parent process ID: 2261
```

После вывода идентификаторов процессов программа ожидает нажатия клавиши. В это время следует запустить отдельный терминал bash и ввести команду ps -axf, выводящую список процессов.

Листинг 6. Список процессов

```
user@comp:~$ ps -axf
  PID TTY      STAT   TIME COMMAND
    2 ?        S      0:00 [kthreadd]
    3 ?        S      0:00  \_ [migration/0]
……..
 2259 ?        Sl     0:02 gnome-terminal
 2260 ?        S      0:00  \_ gnome-pty-helper
 2261 pts/0    Ss     0:00  \_ bash
 2302 pts/0    S+     0:00  |  \_ ./getpid
 2305 pts/1    Ss     0:00  \_ bash
 2318 pts/1    R+     0:00      \_ ps -axf
```

В списке по полученным идентификаторам можно отыскать:

- родительский процесс – терминал bash (PID=2261);
- являющийся дочерним по отношению к bash процесс getpid (PID=2302);
- отдельно запущенный для вызова команды ps -axf терминал bash (PID=2305);
- процесс, отображающий список процессов ps (PID=2318).

Следует обратить внимание, что идентификаторы процессов увеличиваются, т.е. последний запущенный процесс будет иметь наибольший идентификатор.

3.2. Создание процессов.

Создание новых процессов в ОС GNU/Linux производится при помощи системного вызова, описанного в заголовочном файле unistd.h

pid_t fork().

В результате этого вызова создается новый процесс, являющийся почти точной копией исходного. У дочернего процесса свое адресное пространство, поэтому если в нем изменить значение переменной, то это никак не повлияет на значение переменной в родительскому процессе. Следующая после вызова fork команда будет выполнена уже обоими процессами.

Для того, чтобы процессы знали, кто из них родительский, а кто – дочерний существует два способа.

Первый способ основан на получении идентификатора при помощи вызова getpid. У родительского процесса он будет равен идентификатору до выполнения вызова fork. У дочернего он будет иметь большее значение, т.к. процесс запущен позже родительского.

Второй способ основан на анализе результата вызова fork. Возвращаемое значение типа pid_t (аналог типа int):

- = PID дочернего для родительского процесса;
- = 0 для дочернего процесса.

Функция wait приостанавливает родительский процесс до тех пор, пока не завершится один из его дочерних процессов:

pid_t wait(int *stat_loc);

Для ожидания завершения конкретного дочернего процесса используется функция

pid_t waitpid(pid_t pid, int *stat_loc, int options),

где pid – идентификатор дочернего процесса

Пример. Программа создает дочерний процесс. Для определения «кто есть кто» используется значение, возвращаемое вызовом fork. Перед вызовом fork переменной A присваивается значение 0. В дочернем процессе переменная увеличивается на 1. Родительский процесс ожидает окончания дочернего, после чего выводит значение переменной A.

```
/* вызов fork */
#include <stdio.h>
#include <unistd.h>
int main(int argc, char* argv[])
{
    int A = 0,child_pid;
    printf ("Before fork():\nA = %d\n",A);
    child_pid = fork(); // создание дочернего процесса
    if (child_pid == 0) // это дочерний процесс
    {
```

```
        printf("It's the child process. Child_pid = %d; getpid = %d; A =
%d\n",child_pid,getpid(),A);
        A++;
        printf("It's the child process. Change A = %d\nPress enter..\n",A);
        getchar();
        }
        else if (child_pid > 0) // это родительский процесс
        {
        wait(child_pid);
        printf ("It's the main process. Child_pid = %d; getpid = %d; getppid = %d; A
= %d\nPress enter..\n",child_pid,getpid(),getppid(),A);
        getchar();
        }
        else printf("fork error");
        return 0;
    }
```

Листинг 7. Результат работы программы.

```
user@comp:~/proj/fork$ ./fork
Before fork():
A = 0
It's the child process. Child_pid = 0; getpid = 1594; A
= 0
It's the child process. Change A = 1
Press enter..
It's the main process. Child_pid = 1594; getpid =
1593; getppid = 1515; A = 0
Press enter..
```

В отдельном терминале до завершения дочернего процесса следует выполнить команду ps -axf.

Листинг 8. Список процессов.

```
 1514 ?        S      0:00  \_ gnome-pty-helper
 1515 pts/0    Ss     0:00  \_ bash
 1593 pts/0    S+     0:00  |   \_ ./fork
 1594 pts/0    S+     0:00  |      \_ ./fork
 1538 pts/1    Ss     0:00  \_ bash
 1595 pts/1    R+     0:00      \_ ps -axf
```

Для замещения образа памяти процесса используется одна из разновидностей вызова exec

int execvp(const char *file, char *const argv[]).

Если применить его к обычному процессу, то выполнится программа, указанная в переменной file.

Пример. Программа запускает команду ls с аргументом /home/user. После успешного завершения ls команды, указанные после execvp не выполняются.

/* Замещение выполняемого процесса другой программой */

```
#include <stdio.h>
#include <unistd.h>
int main(int argc, char* argv[], char* envp[])
{
     char program [] = "ls";
     char* arg_list [10] = {"ls","/home/user"};
     execvp (program,arg_list);
     printf("Error on program start");
     return 123;
}
```

Листинг 9. Результат работы программы.

```
user@comp:~/proj/fork$ ./exec
1.txt  Desktop  distr  proj  spisok  tmp  tmp1  tmp2
```

3.3. Совместная работа вызовов fork и exec.

При последовательном использовании вызовов fork и exec выполняемый процесс вначале разветвляется на два, а затем вместо второго запускается другая программа.

Пример. Программа создает дочерний процесс, в котором выполняется вызов exec.

```
/* Запуск программы при помощи fork и exec */
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char* argv[])
{
   int child_pid = fork();
   if (child_pid == 0)
   {
     printf("It's the children process:\n");
     char* arg_list [] = {"ls","/home/user"};
     execvp ("ls", arg_list);
     exit(1);
   }
   else
   {
      wait(child_pid);
     printf("It's the main process. Press Enter..\n");
     getchar();
   }
   return 0;
}
```

Листинг 10. Результат работы программы.

```
user@comp:~/proj/fork$ ./fork_exec
It's the children process:
```

```
1.txt  Desktop      distr  er  proj  spisok  tmp  tmp1
tmp2
It's the main process. Press Enter..
```

Для завершения процессов используется функция
int kill(pid_t pid, int sig),
где pid – идентификатор процесса;
sig –сигнал, посылаемый процессу.

Пример. Программа создает дочерний процесс и после паузы отправляет ему сигнал завершения.

```
/* Завершение дочернего процесса */
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
int main(int argc, char* argv[])
{
   int child_pid = fork();
   if (child_pid == 0)
   {
     printf("It's the children process:\n");
     sleep(1);
     char* arg_list[] = {""};
     execvp ("top", arg_list);
     exit(1);
   }
   else
   {
     int k;
     sleep(7);
     k = kill (child_pid,SIGTERM);
     wait(child_pid);
     printf("Children process terminated..\n");
     printf("It's the main process. Press Enter..\n");
     getchar();
   }
   return 0;
}
```

Листинг 11. Результат работы программы.

```
user@comp:~/proj/fork$ ./kill
It's the children process:
 - 19:38:15 up 21 min,  2 users,  load average: 0.00,
0.05, 0.04
Tasks:  92 total,   1 running,  91 sleeping,   0 stopped,
0 zombie
Cpu(s):  0.7%us,  1.0%sy,  0.0%ni, 98.0%id,
```

```
0.3%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   124784k total,   120140k used,     4644k free,
2228k buffers
Swap:   223224k total,     4724k used,   218500k free,
50796k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU
%MEM    TIME+  COMMAND
 1585 user      20   0  2440 1144  900 R  1.0  0.9
0:00.09 top
  143 root      20   0     0    0    0 S  0.3  0.0   0:00.32
ata/0
  965 root      20   0 30396  16m 5884 S  0.3 13.8
0:12.24 Xorg
……..
    1 root      20   0  2036  644  612 S  0.0  0.5   0:03.42
init
Children process terminated..
It's the main process. Press Enter..
```

3.4. Контрольные вопросы.

3.4.1. Почему в листинге 7 переменная A сохранила значение 0, несмотря на то, что в дочернем процессе она была увеличена на 1?

3.4.2. В списке процессов, листинг 8, второй терминал bash имеет PID=1538, меньший, чем запущенная программа fork. Ошибка ли это?

3.4.3. Почему в листинге 9 не была выведена на дисплей строка «Error on program start»?

3.4.4. Что делает вызов wait(child_pid) в программе с совместной работой fork и exec? Что будет, если эту строку закомментировать?

4. Ход выполнения работы.

4.1. Написать, скомпилировать и отладить примеры программы из теоретической части. Ответить на вопросы преподавателя.

4.2. Выполнить задание по варианту (см. п. 5).

4.3. Составить отчет о выполненной работе (см. п. 6).

5. Варианты заданий.

Студент	**Задание**
1. Аскарова Лиана Ильдаровна	«Списки процессов». В дочернем процессе получить список процессов, записать его в файл и завершить процесс. В родительском процессе дождаться окончания дочернего, получить список процессов и дописать его в созданный дочерним процессом файл. Сравнить списки процессов.
2. Афанасьев Равиль Сергеевич	
3. Виноградов	«Несколько дочерних». Главный процесс в

Алексей Олегович	течение 20 секунд через каждые 5 секунд порождает по одному дочернему процессу. Дочерние процессы дописывают в файл свои идентификаторы (PID). Главный процесс ждет окончания дочерних и дописывает в файл идентификаторы дочерних процессов.
4. Гарифьянов Радик Ралусович	
5. Гильмутдинов Ильгиз Вахитович	«Эстафета». Определить в главном процессе целочисленную переменную. Дочерний процесс увеличивает эту переменную в 2 раза и проверяет, не превысило ли значение переменной числа, заданного в качестве аргумента командной строки. Если превысило, то процесс завершается, иначе вновь создается аналогичный дочерний процесс. Главный процесс определяет количество созданных дочерних процессов.
6. Гомзяков Александр Андреевич	
7. Закиев Марат Рафисович	«Общий файл». Открыть файл и записать в него первое сообщение («Hello »). После разветвления процесса в дочернем записать в файл второе сообщение («World»). Выполнить программу, просмотреть созданный файл и объяснить результат. Что будет, если попытаться закрыть файл в дочернем процессе, а в родительском после ожидания wait(child_pid) записать в файл третье сообщение?
8. Зянгареев Урал Тимерзянович	
9. Косолапова Анна Андреевна	«Количество слов в файле». Программа принимает в качестве аргумента имя файла. Родительский процесс создает дочерний, через временный файл передает ему имя файла, принимает в качестве ответа количество слов в файле и выводит результат на дисплей.
10. Крысов Артём Владимирович	
11. Мерзляков Иван Олегович	«Количество символов в строке». Программа принимает в качестве аргумента строку. Родительский процесс создает дочерний, через временный файл передает ему строку, принимает в качестве ответа количество символов в строке и выводит результат на дисплей.
12. Фаттахова Гульназ Салимхановна	
13. Халиков Ильнур Фанзилевич	«Количество файлов в каталоге». Программа принимает в качестве аргумента полный путь к каталогу. Родительский процесс создает дочерний, через временный файл передает ему путь, принимает в качестве ответа количество файлов в каталоге и выводит результат на дисплей.
14. Хаматуллин Артур Рафикович	
15. Хамматов Артур Зинфирович	«Количество символов в строке». Программа принимает в качестве аргумента строку.

16. Храмцов Константин Алексеевич	Родительский процесс создает дочерний, через временный файл передает ему строку, принимает в качестве ответа количество символов в строке и выводит результат на дисплей.
17. Шведов Евгений Николаевич	«Ералаш». Родительский процесс последовательно запускает в дочерних все исполняемые файлы, которые он сможет найти в указанном каталоге. После запуска последнего происходит задержка в 10 секунд, после чего все дочерние процессы закрываются и выводится на дисплей их количество.
18. Юсупжанов Ринат Рафаэлевич	

6. Оформление отчета.

Составить отчет о проделанной работе, содержащий:

6.1. Цель работы, ход выполнения.

6.2. Тексты исследуемых программ с комментариями.

6.3. Скриншоты работы программ.

6.4. Выводы о возможностях, предоставляемых ОС GNU/Linux для создания процессов.