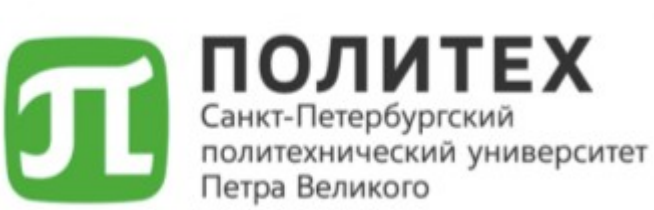
Федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический университет
Петра Великого»
Институт компьютерных наук и технологий
Фундаментальная информатика и информационные технологии



Отсчет по лабораторным работам

по дисциплине «Системное программное обеспечение GNU/Linux»

Студент гр. 5130904/30008 Ребдев П.А

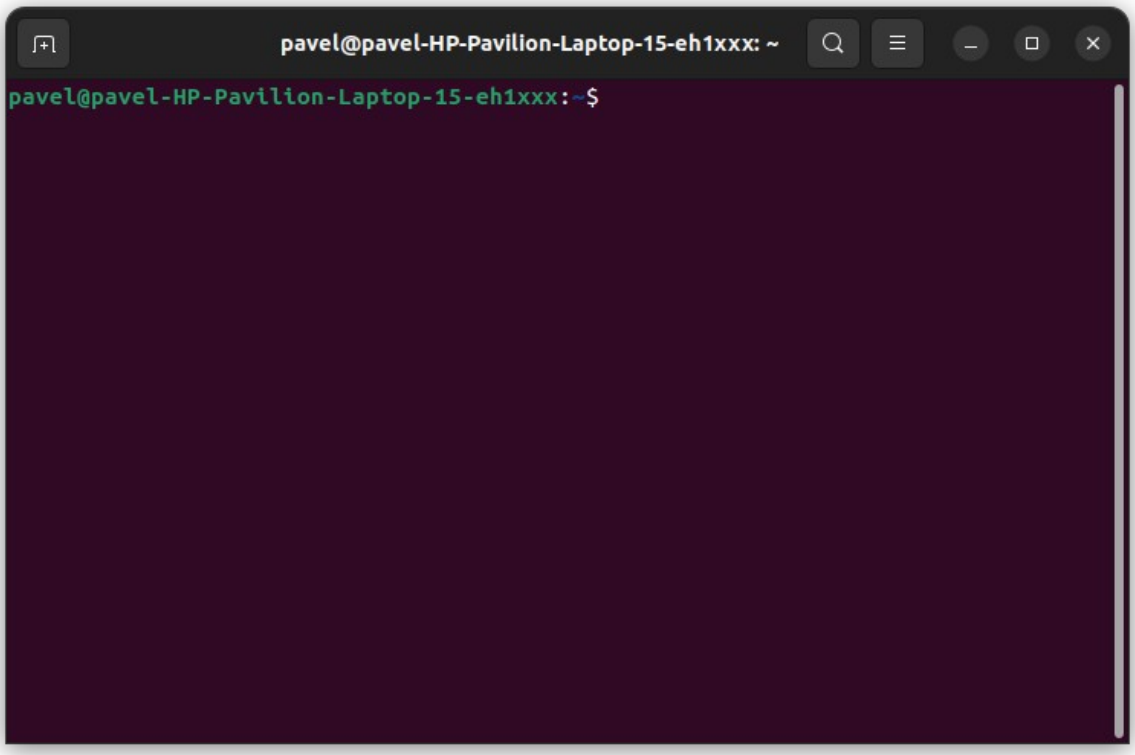
Руководитель: доц. Шмаков В.Э

Санкт-Петербург
2023

Оглавление:

Лабораторная работа №1
«БАЗОВЫЕ КОМАНДЫ ОС»

1. Войдите в систему под логином вашей учебной группы, получив необходимый пароль у преподавателя. +
2. Запустите терминал нажатием комбинации клавиш Ctrl + Alt + t . +



3. Выполните на терминале команды shell , рассмотренные в материалах лекций. Такие, как pwd , who , ls , cd , mkdir , rm , chmod.

```
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop$ pwd
/home/pavel/Desktop
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop$ who
pavel    tty2         2023-09-20 15:46 (tty2)
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop$ ls
Folder1  Forexperiments  ForLab  ForLabMedia  Wallpapers
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop$ cd ~/Desktop
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop$
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop$ mkdir Folder1
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop$ rm file1.txt
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop$ chmod u+w Folder1
```

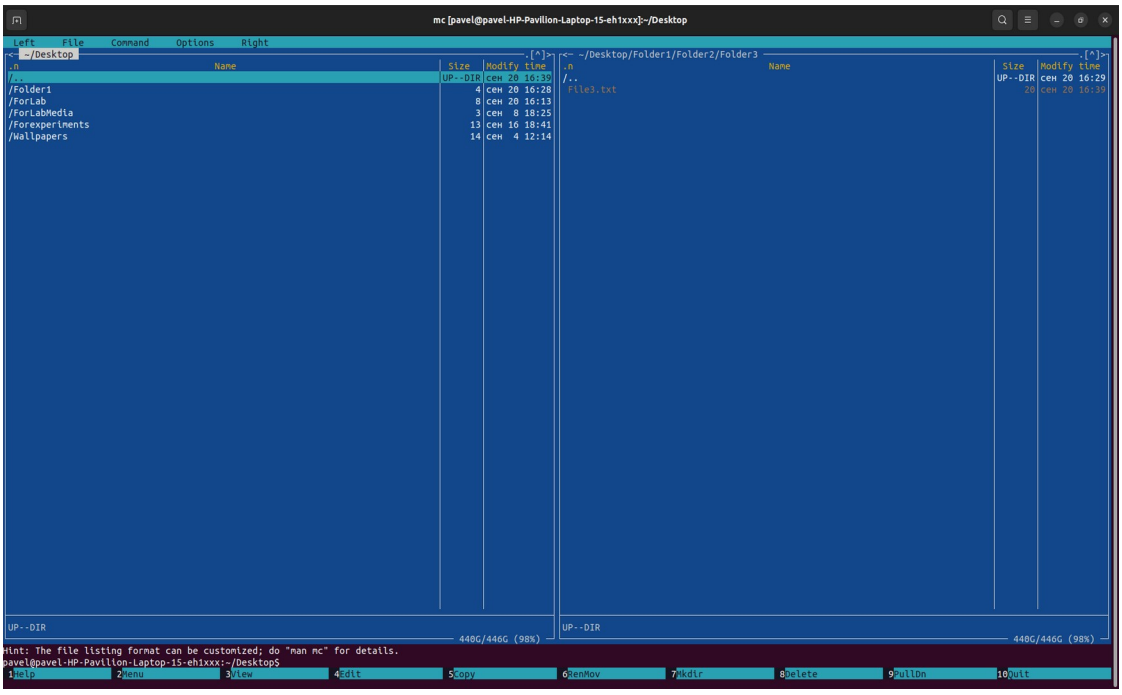
Полное описание синтаксиса и семантики этих и любых других команд можно увидеть в системе помощи ОС Linux, вызы-ваемой с терминала в виде man < интересующая вас команда >, или запускайте веб-браузер и используйте всю информационную мощь Интернета. +

4. Проанализируйте результаты выполнения команд. Наиболее значимые скриншоты (снимаются нажатием клавиш Alt + Prnt Scrn) поместите в отчет. +

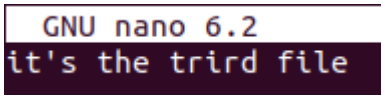
5. Создайте дерево каталогов глубиной вложения до трех уров-ней, а в самих каталогах создайте текстовые файлы. Примените различные способы создания новых файлов.

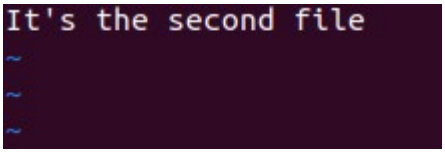
```
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop$ mkdir Folder1
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop$ cd Folder1
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/Folder1$ cat >File1.txt
It's first file
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/Folder1$ mkdir Folder2
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/Folder1$ cd Folder2
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/Folder1/Folder2$ echo> File2.txt
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/Folder1/Folder2$ mkdir Folder3
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/Folder1/Folder2$ cd Folder3
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/Folder1/Folder2/Folder3$ vim File3.txt
```

6. Запустите с терминала Midnight Commander Midnight Commander вводом команды mc и ознакомьтесь с его основными возможностями по работе с файловой системой.



Наполните созданные на предыдущем шаге файлы каким-либо содержанием. Для этого можно использовать любой редактор, от vim , встроенного в ОС, до графического редак-тора gedit , вызываемого из графической оболочки ОС.






```
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop$ cd Folder1
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/Folder1$ cat File1.txt Folder2/File2.txt
It's first file
It's the second file
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/Folder1$ cp File1.txt File1D.txt
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/Folder1$ find . -name "File3.txt"
./Folder2/Folder3/File3.txt
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/Folder1$ link File1.txt linkToFile1.txt
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/Folder1$ chmod u+x File1.txt
```

7. Выполните на терминале вторую серию команд cat , cp , find ,link , chmod , рассмотренных в лекциях. Для манипуляций с помощью этих команд используйте текстовые файлы, созданные и на-полненные на предыдущем шаге.
8. Попытайтесь создать на своем дереве какой-нибудь каталог с правами доступа, аналогичными каталогу darkroom , рассмотренному в лекциях.

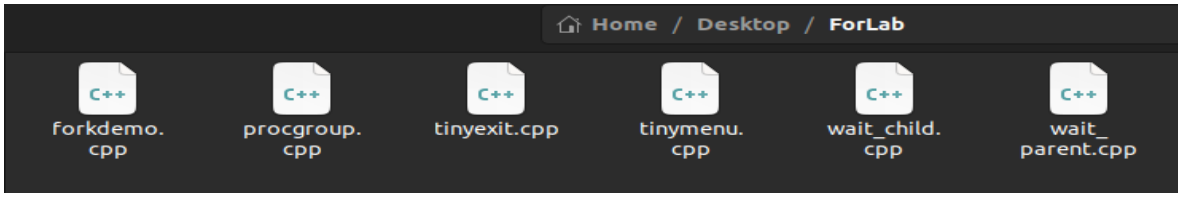

```
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab1$ mkdir darkroom
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab1$ cd darkroom
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab1/darkroom$ vim secretfile.txt
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab1/darkroom$ chmod a+r-w-x secretfile.txt
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab1/darkroom$ chmod u+w+r secretfile.txt
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab1/darkroom$ cd ../
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab1$ chmod a-r-w-x darkroom
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab1$ cd darkroom
bash: cd: darkroom: Permission denied
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab1$ cat darkroom/secretfile.txt
cat: darkroom/secretfile.txt: Permission denied
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab1$ chmod u+x darkroom
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab1$ cat darkroom/secretfile.txt
secret file, from darkroom
```

Вывод: *были изучены основные консольные команды для работы с файлами и процессами терминала Linux. Была рассмотрена концепция «darkroom»*

Лабораторная работа №2

«ЗАПУСК И ЗАВЕРШЕНИЕ ПРОЦЕССОВ»

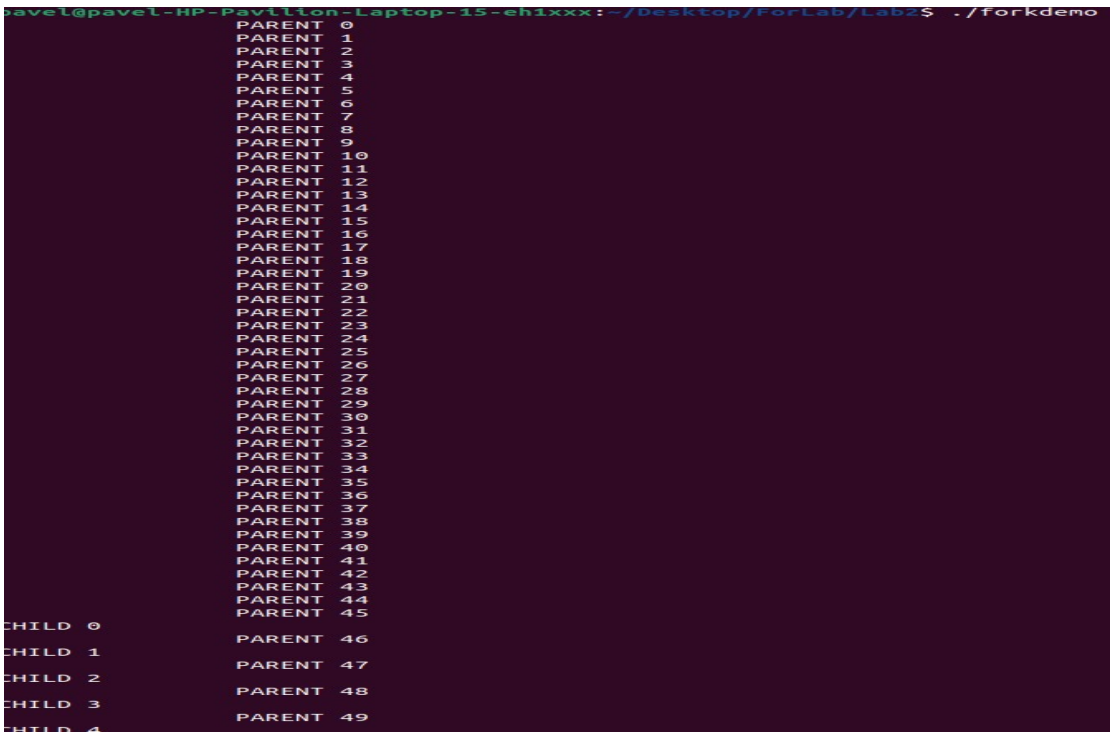
1. Войдите в систему и скопируйте в свой HOME-каталог с разделяемого ресурса набор исходных файлов для второй лабораторной работы. +

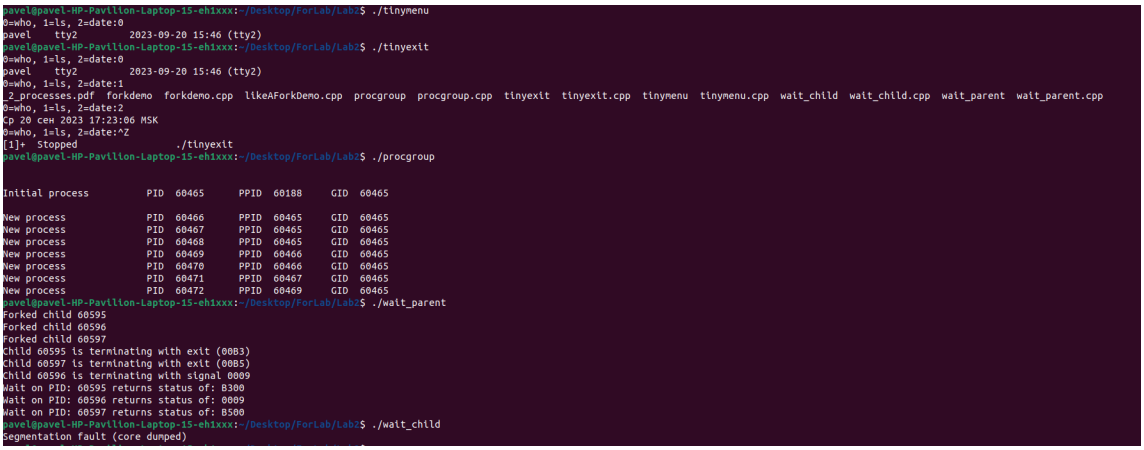


2. Скомпилируйте и выполните примеры программ forkdemo.cpp , tinymenu.cpp , tinyexit.cpp , procgroup.cpp , wait_parent.cpp. Процесс wait_parent при исполнении запускает процесс wait_child.

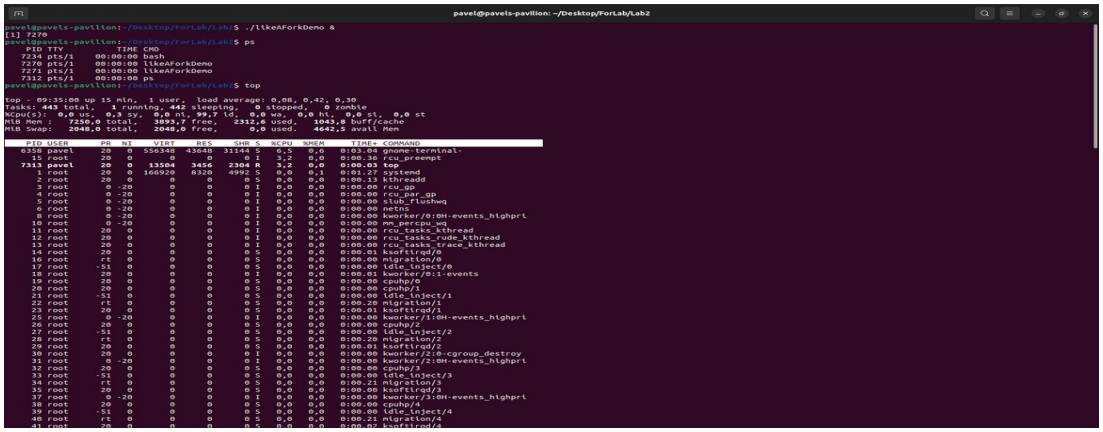
```
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab2$ g++ forkdemo.cpp -o forkdemo
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab2$ g++ tinymenu.cpp -o tinymenu
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab2$ g++ tinyexit.cpp -o tinyexit
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab2$ g++ procgroup.cpp -o procgroup
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab2$ g++ wait_parent.cpp -o wait_parent
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab2$ g++ wait_child.cpp -o wait_child
```

Программа wait_child.cpp компилируется с опцией: g+ +wait_child.cpp-o wait_child . Пояснения к данным программам можно найти в тексте лекций. При необходимости конвертации текстовых файлов из формата DOS в Linux и наоборот используйте команды dos2unix и unix2dos.

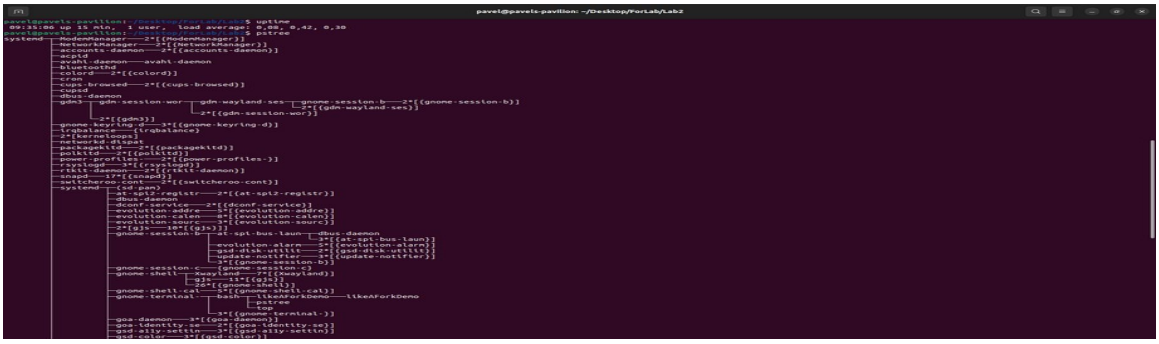




3. Модифицируйте программу forkdemo.cpp (или создайте собственную), так чтобы ввод/вывод на терминал отсутствовал, а при проходе по циклу была временная задержка, например sleep (7). Запустите эту программу в фоновом режиме (background), введя при запуске символ & после пробела и зафиксировав значение PID, назначенное системой фоновому процессу при запуске. Выполните на терминале команды ps , top , uptime , pstre . Снимите свой фоновый процесс командой kill с соответствующими



параметрами. Скриншоты вместе с пояснениями к выполнению процессов и команд, а также исходные тексты программ, составленных вами самостоятельно, приведите в отчете.



4. Исследуйте, что произойдет, если процесс-потомок сменит текущий каталог, будет ли изменен текущий каталог для родителя? Создайте программу, подтверждающую ответ и приведите в отчете.+

```cpp
#include<unistd.h>
#include<time.h>
#include<iostream>

int main(){

        long long int processID = fork();
        switch(processID){
        case(-1):
        std::cout << "Error, please restart";
        break;

        case(0):
        std::cout << "Child is born";

        chdir("./..");
        sleep(100);

        std::cout << "Child is dead";
        break;

        default:
        std::cout << "Parent is born";

        sleep(100);

        std::cout << "Parent is dead";
        }
        return 0;
}
```

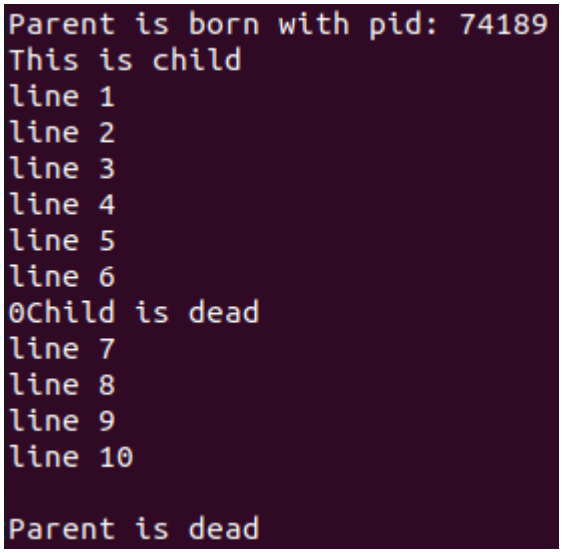
```
pavel@pavels-pavilion:~/Desktop/ForLab/Lab2$ g++ myProgram.cpp -o myProgram
pavel@pavels-pavilion:~/Desktop/ForLab/Lab2$ ./myProgram
Parent is born with pid: 53071
Child is born
^Z
[1]+  Остановлен    ./myProgram
pavel@pavels-pavilion:~/Desktop/ForLab/Lab2$ pwdx 53071
53071: /home/pavel/Desktop/ForLab
pavel@pavels-pavilion:~/Desktop/ForLab/Lab2$ pwdx 53070
53070: /home/pavel/Desktop/ForLab/Lab2
```

Если ребёнок сменит каталог, родитель останется на прежнем месте

5. Проиллюстрируйте как процесс-родитель и процесс-потомок разделяют один и тот же дескриптор и смещение

текстового файла. Для этого составьте программу, в которой процесс-родитель должен открывать текстовый файл и запускать потомка. Потомок должен читать порцию данных из открытого файла и выводить на консоль. По завершению потомка родитель должен читать из того же файла и выводить результат на консоль. Можете использовать вызов sleep() для синхронизации доступа родителя и потомка к файлу+

```
#include<unistd.h>
#include<time.h>
#include<iostream>
#include<fstream>

int main(){

        std::fstream txtFile;
        txtFile.open("txtFile.txt");

        long long int processID = fork();
        char a = '0';
        switch(processID){
        case(-1):
        std::cout << "Error, please restart" << '\n';
        break;

        case(0):
        std::cout << "This is child"<< '\n';

        sleep(2);
        while(!txtFile.eof()){
                txtFile.get(a);
                std::cout<<a;
        }

        std::cout << "Child is dead"<< '\n';
        break;

        default:
        std::cout << "Parent is born with pid: " << processID << '\n';

        sleep(2);
        while(!txtFile.eof()){
                txtFile.get(a);
                std::cout<<a;
        }

        std::cout << "Parent is dead"<< '\n';
        }
        return 0;
}
```



Как мы видим из последнего изображения процесс родитель и процесс потомок могут одновременно считывать данные из одного и того же файла, при этом они работают с одним образцом файла, а не с дубликатами, поэтому если родитель считает 50 символов, то ребёнок сможет начать считывание с 51 символа.

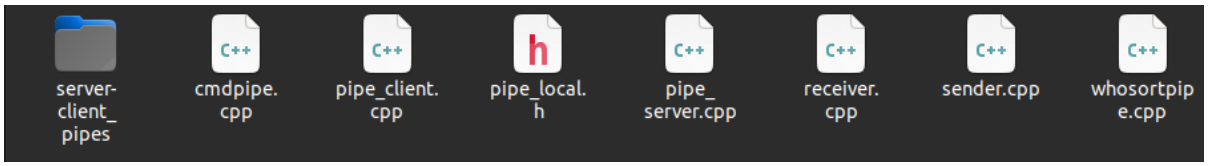
Вывод: *в ходе выполнения лабораторной работы были изучены взаимодействия родительских и потомственных процессов.*

Было изучено поведение родительского процесса при изменение директории потомственного процесса и одновременное взаимодействие с файлом родительского и потомственного процессов.

Лабораторная работа №3

«ПРОГРАММНЫЕ КАНАЛЫ»

1. Войдите в систему и скопируйте с разделяемого ресурса в свой HOME-каталог набор исходных файлов для третьего занятия. +



2. Скомпилируйте и выполните программу whosortpipe.cpp .

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab3$ g++ whosortpipe.cpp -o whosortpipe
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab3$ ./whosortpipe
pavel    tty2         2023-10-18 10:40 (tty2)
```

Сопоставьте результат выполнения программы с выполнением этих же двух команд из shell в конвейерном режиме (|).

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab3$ who | sort
pavel    tty2         2023-10-20 13:37 (tty2)
```

Не забывайте приводить в отчете анализ результатов работы этой программы (как и всех последующих) с соответствующими скриншотами.

Вывод: в ходе выполнения пункта 2 мы выясняли, что программа whosortpipe.cpp выполняет такое же действие команде who | sort

3. Программу cmdpipe.cpp запускайте после компиляции, задавая ей при стартах в качестве параметров командной строки пары команд shell для конвейеризации (who и sort ; last и sort ;

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab3$ ./cmdpipe who sort
pavel    tty2         2023-11-01 10:24 (tty2)
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab3$ ./cmdpipe last sort

pavel    tty2         tty2             Fri Oct 13 09:19 - down   (01:57)
pavel    tty2         tty2             Fri Oct 13 11:52 - down   (03:42)
pavel    tty2         tty2             Fri Oct 13 15:35 - down   (02:15)
pavel    tty2         tty2             Fri Oct 13 17:54 - down   (00:00)
pavel    tty2         tty2             Fri Oct 20 09:36 - down   (00:19)
pavel    tty2         tty2             Fri Oct 20 09:56 - down   (01:30)
```

last и more ;

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab3$ ./cmdpipe last more
pavel    tty2         tty2             Wed Nov  1 10:24   still logged in
reboot   system boot  6.2.0-35-generic Wed Nov  1 10:24   still running
pavel    tty2         tty2             Tue Oct 31 12:05 - down   (10:34)
reboot   system boot  6.2.0-35-generic Tue Oct 31 12:05 - 22:40  (10:34)
pavel    tty2         tty2             Tue Oct 31 10:24 - down   (00:37)
reboot   system boot  6.2.0-35-generic Tue Oct 31 10:24 - 11:02  (00:37)
pavel    tty2         tty2             Sun Oct 29 12:10 - down  (1+10:24)
reboot   system boot  6.2.0-35-generic Sun Oct 29 12:10 - 22:35 (1+10:24)
pavel    tty2         tty2             Fri Oct 27 15:56 - down   (01:57)
reboot   system boot  6.2.0-35-generic Fri Oct 27 15:56 - 17:53  (01:57)
```

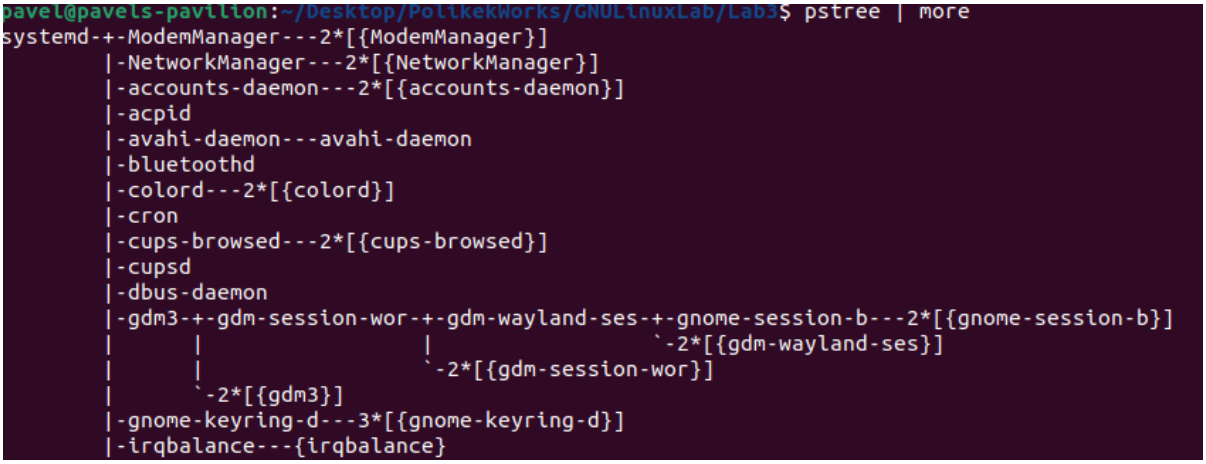
pstree и more).

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab3$ ./cmdpipe pstree more
systemd-+-ModemManager---2*[{ModemManager}]
        |-NetworkManager---2*[{NetworkManager}]
        |-accounts-daemon---2*[{accounts-daemon}]
        |-acpid
        |-avahi-daemon---avahi-daemon
        |-bluetoothd
        |-colord---2*[{colord}]
        |-cron
        |-cups-browsed---2*[{cups-browsed}]
        |-cupsd
        |-dbus-daemon
        |-gdm3-+-gdm-session-wor-+-gdm-wayland-ses-+-gnome-session-b---2*[{gnome-session-b}]
        |      |                 |                 `-2*[{gdm-wayland-ses}]
        |      |                 `-2*[{gdm-session-wor}]
        |      `-2*[{gdm3}]
        |-gnome-keyring-d---3*[{gnome-keyring-d}]
        |-irqbalance---{irqbalance}
```

Сопоставьте результаты запусков программы с выполнением тех же пар команд из shell в конвейерном режиме.

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab3$ who | sort
pavel    tty2         2023-11-01 10:24 (tty2)
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab3$ last | sort

pavel    tty2         tty2             Fri Oct 13 09:19 - down   (01:57)
pavel    tty2         tty2             Fri Oct 13 11:52 - down   (03:42)
pavel    tty2         tty2             Fri Oct 13 15:35 - down   (02:15)
pavel    tty2         tty2             Fri Oct 13 17:54 - down   (00:00)
pavel    tty2         tty2             Fri Oct 20 09:36 - down   (00:19)
pavel    tty2         tty2             Fri Oct 20 09:56 - down   (01:30)
```

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab3$ last | more
pavel    tty2         tty2             Wed Nov  1 10:24   still logged in
reboot   system boot  6.2.0-35-generic Wed Nov  1 10:24   still running
pavel    tty2         tty2             Tue Oct 31 12:05 - down   (10:34)
reboot   system boot  6.2.0-35-generic Tue Oct 31 12:05 - 22:40  (10:34)
pavel    tty2         tty2             Tue Oct 31 10:24 - down   (00:37)
reboot   system boot  6.2.0-35-generic Tue Oct 31 10:24 - 11:02  (00:37)
pavel    tty2         tty2             Sun Oct 29 12:10 - down  (1+10:24)
reboot   system boot  6.2.0-35-generic Sun Oct 29 12:10 - 22:35 (1+10:24)
pavel    tty2         tty2             Fri Oct 27 15:56 - down   (01:57)
reboot   system boot  6.2.0-35-generic Fri Oct 27 15:56 - 17:53  (01:57)
```



Можно ли с помощью вызова popen() создать программу, организующую конвейер из трех команд shell , передаваемых ей в качестве параметров командуной строки при запуске? Если да, то создайте такую программу, если нет, дайте обоснованный ответ, почему нельзя.

Нет, с помощью popen() нельзя создать программу, организующую конвейер из трех команд shell, т.к popen() принимает либо аргумент r либо аргумент w, а для конвейера из трёх процессов один из них должен сначала считать информацию из первого процесса, а потом отправить её в другой процесс.

4. Напишите программу (например, на основе вызовов pipe()), воспринимающую варьируемое количество команд, передаваемых ей при запуске в качестве параметров. Каждая последующая команда должна быть соединена с предыдущей с помощью конвейера. Так, при запуске программы $ ./a.out last sort more должны выполняться действия, эквивалентные запуску команд из shell :$ last | sort | more.

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<limits.h>
#include<iostream>

int main(int argc, char *argv[])
{
  FILE  *fin, *fout;
  char  buffer[PIPE_BUF];

  int n = 0;
  for (int i = 2; i < argc; ++i)
  {
    fin  = popen(argv[i-1], "r");
    fout = popen(argv[i], "w");

    while ((n = read(fileno(fin), buffer, PIPE_BUF)) > 0)
    {
      write(fileno(fout), buffer, n);
    }
    pclose(fin);
    pclose(fout);
  }

  return 0;
}
```

5. Разберите и выполните пример клиент-серверного взаимодействия, организованного на конвейерах различного типа. Исходный текст примера содержится в файлах pipe_server.cpp , pipe_client.cpp и pipe_local.h и разобран в материалах лекций. Сервер запускается в фоновом режиме. Проанализируйте результаты функционирования данной системы и ее недостатки. Программа сервер этого примера исполняет каждый командный запрос поочередно. Если какой-либо запрос потребует много времени, все остальные клиентские процессы будут ожидать обслуживания.

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab3$ ./pipe_server &
[14] 120044
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab3$ ./pipe_client

cmd>last sort


wtmp begins Mon Sep  4 09:45:45 2023

cmd>last

pavel    tty2         tty2             Wed Nov  1 10:24   still logged in
reboot   system boot  6.2.0-35-generic Wed Nov  1 10:24   still running
pavel    tty2         tty2             Tue Oct 31 12:05 - down   (10:34)
reboot   system boot  6.2.0-35-generic Tue Oct 31 12:05 - 22:40  (10:34)
pavel    tty2         tty2             Tue Oct 31 10:24 - down   (00:37)
reboot   system boot  6.2.0-35-generic Tue Oct 31 10:24 - 11:02  (00:37)
pavel    tty2         tty2             Sun Oct 29 12:10 - down  (1+10:24)
reboot   system boot  6.2.0-35-generic Sun Oct 29 12:10 - 22:35 (1+10:24)
pavel    tty2         tty2             Fri Oct 27 15:56 - down   (01:57)
reboot   system boot  6.2.0-35-generic Fri Oct 27 15:56 - 17:53  (01:57)
pavel    tty2         tty2             Fri Oct 27 15:10 - down   (00:45)
reboot   system boot  6.2.0-35-generic Fri Oct 27 15:10 - 15:55  (00:45)
pavel    tty2         tty2             Fri Oct 27 12:22 - down   (01:11)
reboot   system boot  6.2.0-35-generic Fri Oct 27 12:22 - 13:33  (01:11)
pavel    tty2         tty2             Fri Oct 27 09:33 - down   (01:47)
reboot   system boot  6.2.0-35-generic Fri Oct 27 09:33 - 11:21  (01:47)
pavel    tty2         tty2             Thu Oct 26 19:30 - down   (00:18)
reboot   system boot  6.2.0-35-generic Thu Oct 26 19:29 - 19:48  (00:18)
pavel    tty2         tty2             Thu Oct 26 09:58 - down   (08:07)
```

6. Модифицируйте прогрмму pipe_server.cpp так, чтобы при получении нового сообщения от очередного клиента сервер порождал очередной дочерний процесс для выполнения задачи обслуживания данного запроса (выполнения переданной от клиента команды и переправки клиенту результата).

```c
/* The server program  pipe_server.cpp */
#include"pipe_local.h"
int main(void)
{
int     n, done, dummyfifo, privatefifo, publicfifo;
static char buffer[PIPE_BUF];
FILE *fin;
struct message msg;
        /* Generate the public FIFO */
mknod(PUBLIC, S_IFIFO | 0666, 0);
        /* OPEN the public FIFO for reading and writing */
if (((publicfifo=open(PUBLIC, O_RDONLY))==-1) ||
     (dummyfifo=open(PUBLIC, O_WRONLY | O_NDELAY))==-1){
    perror(PUBLIC);
    exit(1);
    }
    /* Message can be read from the PUBLIC pipe */
    long long int Pid = fork();
    if(Pid == 0){
    while(read(publicfifo, (char *) &msg, sizeof(msg))>0){
    n = done = 0; /* Clear counters | flags */
    do{         /* Try OPEN of private FIFO */
        if ((privatefifo=open(msg.fifo_name, O_WRONLY | O_NDELAY))==-1)
        sleep(3);       /* Sleep a while  */
        else{   /* OPEN succesful */
            fin = popen(msg.cmd_line, "r");     /* Execute the cmd */
            write(privatefifo, "\n", 1);        /* Keep output pretty */
            while((n=read(fileno(fin), buffer, PIPE_BUF))>0){
                write(privatefifo, buffer, n);  /*to private FIFO */
                memset(buffer, 0x0, PIPE_BUF);  /* Clear it out */
                }
                pclose(fin);
                close(privatefifo);
                done = 1;       /* Record succes */
            }
        }while(++n<5 && !done);

        if(!done)               /* Indicate failure */
            write(fileno(stderr), "\nNOTE: SERVER ** NEVER ** accessed private FIFO\n", 48);
    }
    }
}
```

Вывод: *были изучены такие понятия как: конвейер, тунель. Мы узнали разницу между потоком и процессом. Были изучены pipe и named pipes.*

Лабораторная работа №4

«КОМАНДНЫЕ ФАЙЛЫ. ПЕРЕМЕННЫЕ ОКРУЖЕНИЯ»

Цель работы. Знакомство с важным атрибутом любой операционной системы – переменными среды (или переменными окружения) и с возможностями их использования в Linux. Освоение языка для составления командных сценариев и написание набора полезных для системного администрирования скриптов.

Последовательность выполнения работы:

1. Создайте несколько символьных переменных среды (переменных окружения). Составьте командный файл (сценарий bash), выводящий на консоль значения этих переменных. Выполните операцию конкатенации (склеивания) значений переменных и выведите полученный результат на консоль. Выделите из конкатенированной переменной среды подстроку и выведите ее на консоль. Замените выделенную подстроку на какое-либо другое значение и выведите измененное значение переменной среды на консоль.

```
1 echo "firstVariable: ${firstVariable} | secondVariable: ${secondVariable}"
2
3 concatenation=$firstVariable$secondVariable
4 echo "Конкатенация: ${concatenation}"
5
6 substring=$(echo $concatenation | cut -c 1-5)
7 echo "Подстрока: ${substring}"
8
9 substring="new string"
10 echo "Изменёная строка: ${substring}"
```

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4$ export firstVariable="first"
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4$ export secondVariable="second"
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4$ ./firstTask.sh
firstVariable: first | secondVariable: second
Конкатенация: firstsecond
Подстрока: first
Изменёная строка: new string
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4$
```

2. Создайте несколько переменных среды в интерпретации, как числовые переменные. В новом командном файле выполните с этими числовыми переменными все допустимые

арифметические операции, выводя на консоль результаты операций и соответствующие комментарии.

```
let sum=firstNumber+secondNumber
echo "Сумма: ${sum}"

let diff=firstNumber-secondNumber
echo "Разница: ${diff}"

let prod=firstNumber*secondNumber
echo "Произведение: ${prod}"

let rat=firstNumber/secondNumber
echo "Отношение: ${rat}"
```

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4$ export firstNumber=10
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4$ export secondNumber=5
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4$ ./secondTask.sh
Сумма: 15
Разница: 5
Произведение: 50
Отношение: 2
```

3. Создайте командный файл (основной), выдающий при старте сообщение и затем вызывающий другой командный файл (его имя задается при старте основного файла в качестве параметра командной строки), который выдает свое сообщение и приостанавливается до нажатия любой клавиши. При возврате управления в вызывающий (основной) файл из него должно выдаваться еще одно сообщение, подтверждающее возврат.

```
echo "Parent file is live"

./"${1}.sh"

echo "Parent is dead"
```

```
echo "Child is live"

read B

echo "Child is dead"
```

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4$ ./trirdTaskParentFile.sh trirdTaskChildFile
Parent file is live
Child is live

Child is dead
Parent is dead
```

4. Составьте командный файл, выводящий на экран различия содержимого двух каталогов, имена которых передаются в

качестве параметров. Отличия искать в именах файлов, их размерах и атрибутах.

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4/firstDir$ ls
4  FirstFile  SecondFile
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4/firstDir$ cat FirstFile
FirstFile
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4/firstDir$ cat SecondFile
SecondFile
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4/firstDir$ cat 4
```

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4/secondDir$ ls
3  FirstFile  SecondFile
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4/secondDir$ cat FirstFile
FirstFile
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4/secondDir$ cat SecondFile
NSecondFile
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4/secondDir$ cat 3
dsadsadsasdd
```

```
1 diff -r $1 $2
```

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4$ ./differenceBetweenTwoDirectories.sh "firstDir" "secondDir"
Только в secondDir: 3
Только в firstDir: 4
diff -r firstDir/SecondFile secondDir/SecondFile
1c1
< SecondFile
---
> NSecondFile
```

5 && 6. Разработайте командный файл сценария для поиска текстовых файлов, содержащих заданную последовательность символов. Эта последовательность передается при запуске в качестве первого параметра командной строки. В качестве второго параметра передается имя файла результатов, который должен быть создан в сценарии для записи в него имен найденных текстовых файлов и номеров их строк, в которых содержится заданная последовательность символов.

```
1 grep $1 * -n >> $2
```

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4$ ./findSubstr.sh echo output.txt
grep: firstDir: Это каталог
grep: secondDir: Это каталог
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4$ cat output.txt
firstTask.sh:1:echo "firstVariable: ${firstVariable} | secondVariable: ${secondVariable}"
firstTask.sh:4:echo "Конкатенация: ${concatenation}"
firstTask.sh:6:substring=$(echo $concatenation | cut -c 1-5)
firstTask.sh:7:echo "Подстрока: ${substring}"
firstTask.sh:10:echo "Изменёная строка: ${substring}"
secondTask.sh:2:echo "Сумма: ${sum}"
secondTask.sh:5:echo "Разница: ${diff}"
secondTask.sh:8:echo "Произведение: ${prod}"
secondTask.sh:11:echo "Отношение: ${rat}"
trirdTaskChildFile.sh:1:echo "Child is live"
trirdTaskChildFile.sh:5:echo "Child is dead"
trirdTaskParentFile.sh:1:echo "Parent file is live"
trirdTaskParentFile.sh:5:echo "Parent is dead"
```

7. Создайте командный файл, который синхронизирует содержимое заданного каталога с эталонным. После запуска и

отработки командного файла в заданном каталоге должен оказаться тот же набор файлов, что и в эталонном (если файла нет – он копируется из эталонного каталога, если найдется файл, которого нет в эталонном, – удаляется). Если файл с некоторым именем есть и в заданном и в эталонном каталогах, то он перезаписывается только в том случае, если в эталонном имеется более новая версия файла. Имена обоих каталогов должны при запуске передаваться командному файлу параметрами командной строки.

```
1 rsync -avu --delete $1 $2
```

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4$ cd firstDir/
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4/firstDir$ ls
4  FirstFile  SecondFile
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4/firstDir$ cd ./..
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4$ cd ThirdDir/
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4/ThirdDir$ ls
```

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4$ ./synchronization.sh firstDir ThirdDir
sending incremental file list
firstDir/
firstDir/4
firstDir/FirstFile
firstDir/SecondFile

sent 295 bytes  received 77 bytes  744,00 bytes/sec
total size is 21  speedup is 0,06
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4$ cd firstDir/
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4/firstDir$ ls
4  FirstFile  SecondFile
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4/firstDir$ cd ./..
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4$ cd ThirdDir/
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab4/ThirdDir$ ls
firstDir
```

Вывод: *были изучены такие основы по работе с shall. Мы научились разрабатывать простые shall -скрипты. Были изучены методы создания переменных среды*

Лабораторная работа № 5

«УЧЕТНЫЕ ЗАПИСИ. ФОНОВЫЙ И ДИАЛОГОВЫЙ РЕЖИМЫ ИСПОЛНЕНИЯ ПРОЦЕССОВ»

1. Создайте учетные записи для нескольких пользователей (не задавая им прав администратора) и объедините их в две группы.

```
pavel:x:1000:1000:Pavel,,,:/home/pavel:/bin/bash
user_2:x:1001:1001:User2,,,:/home/user_2:/bin/bash
user_3:x:1002:1002:User3,,,:/home/user_3:/bin/bash
user_4:x:1003:1003:User4,,,:/home/user_4:/bin/bash
user_5:x:1004:1004:User5,,,:/home/user_5:/bin/bash
```

```
pavel@pavels-pavilion:~$ sudo addgroup newgroup
Добавляется группа «newgroup» (GID 1005) ...
Готово.
pavel@pavels-pavilion:~$ sudo addgroup newgroup2
Добавляется группа «newgroup2» (GID 1006) ...
Готово.
```

```
pavel@pavels-pavilion:~$ sudo usermod -a -G newgroup user_2
pavel@pavels-pavilion:~$ sudo usermod -a -G newgroup user_3
pavel@pavels-pavilion:~$ sudo usermod -a -G newgroup2 user_4
pavel@pavels-pavilion:~$ sudo usermod -a -G newgroup2 user_5
```

Заходя в систему под разными аккаунтами, создайте в соответствующих домашних каталогах файлы, варьируя при этом права доступа для пользователя, для группы, для всех.

```
user_2@pavels-pavilion:~$ vim user_2_File_1.txt
user_2@pavels-pavilion:~$ vim user_2_File_2.txt
user_2@pavels-pavilion:~$ vim user_2_File_3.txt
user_2@pavels-pavilion:~$ cat user_2_File_1.txt
User 2, File 1
user_2@pavels-pavilion:~$ cat user_2_File_2.txt
User 2 File 2
user_2@pavels-pavilion:~$ cat user_2_File_3.txt
User 2 File 3
```

```
user_2@pavels-pavilion:~$ chmod a-w-r-x user_2_File_1.txt
user_2@pavels-pavilion:~$ chmod u+w+r+x user_2_File_1.txt
user_2@pavels-pavilion:~$ chmod a-w-r-x user_2_File_2.txt
user_2@pavels-pavilion:~$ chmod g+w+r+x user_2_File_2.txt
user_2@pavels-pavilion:~$ chmod a+w+r+x user_2_File_3.txt
```

```
user_3@pavels-pavilion:~$ vim user_3_File_1.txt
user_3@pavels-pavilion:~$ vim user_3_File_2.txt
user_3@pavels-pavilion:~$ vim user_3_File_3.txt
user_3@pavels-pavilion:~$ cat user_3_File_1.txt
User 3 File 1
user_3@pavels-pavilion:~$ cat user_3_File_2.txt
User 3 File 2
user_3@pavels-pavilion:~$ cat user_3_File_3.txt
User 3 File 3
user_3@pavels-pavilion:~$ chmod a-w-r-x user_3_File_1.txt
user_3@pavels-pavilion:~$ chmod u+w+r+x user_3_File_1.txt
user_3@pavels-pavilion:~$ chmod a-w-r-x user_3_File_2.txt
user_3@pavels-pavilion:~$ chmod g+w+r+x user_3_File_2.txt
user_3@pavels-pavilion:~$ chmod a+w+r+x user_3_File_3.txt
user_3@pavels-pavilion:~$ ls
user_3_File_1.txt  user_3_File_2.txt  user_3_File_3.txt
user_3@pavels-pavilion:~$ su - user_4
Пароль:
user_4@pavels-pavilion:~$ vim user_4_File1.txt
user_4@pavels-pavilion:~$ vim user_4_File2.txt
user_4@pavels-pavilion:~$ vim user_4_File3.txt
user_4@pavels-pavilion:~$ cat user_4_File1.txt
User 4 File 1
user_4@pavels-pavilion:~$ cat user_4_File2.txt
User 4 File 2

user_4@pavels-pavilion:~$ cat user_4_File3.txt
User 4 File 4
user_4@pavels-pavilion:~$ chmod a-w-r-x user_4_File_1.txt
chmod: невозможно получить доступ к 'user_4_File_1.txt': Нет такого файла или каталога
user_4@pavels-pavilion:~$ chmod a-w-r-x user_4_File_1.txt
chmod: невозможно получить доступ к 'user_4_File_1.txt': Нет такого файла или каталога
user_4@pavels-pavilion:~$ chmod a-w-r-x user_4_File1.txt
user_4@pavels-pavilion:~$ chmod u+w+r+x user_4_File1.txt
user_4@pavels-pavilion:~$ chmod a-w-r-x user_4_File2.txt
user_4@pavels-pavilion:~$ chmod g+w+r+x user_4_File2.txt
user_4@pavels-pavilion:~$ chmod a-w-r-x user_4_File3.txt
user_4@pavels-pavilion:~$ su - user_5
Пароль:
user_5@pavels-pavilion:~$ vim user_5_File_1.txt
user_5@pavels-pavilion:~$ vim user_5_File_2.txt
user_5@pavels-pavilion:~$ vim user_5_File_3.txt
user_5@pavels-pavilion:~$ cat user_5_File_1.txt
User 5 File 1
user_5@pavels-pavilion:~$ cat user_5_File_2.txt
User 5 File 2
user_5@pavels-pavilion:~$ cat user_5_File_3.txt
User 5 File 3
user_5@pavels-pavilion:~$ chmod a-w-r-x user_5_File_1.txt
user_5@pavels-pavilion:~$ chmod u+w+r+x user_5_File_1.txt
user_5@pavels-pavilion:~$ chmod a-w-r-x user_5_File_2.txt
user_5@pavels-pavilion:~$ chmod g+w+r+x user_5_File_2.txt
user_5@pavels-pavilion:~$ chmod a+w+r+x user_5_File_3.txt
user_5@pavels-pavilion:~$
```

Убедитесь, что права доступа разделяются в соответствии с тем, как это задано. Проведите операцию слияния файлов с различными правами доступа и проверьте, какие при этом получаются права у результирующего файла.

```
user_2@pavels-pavilion:~$ rm user_23_File_11.txt
user_2@pavels-pavilion:~$ ls
user_2_File_1.txt  user_2_File_2.txt  user_2_File_3.txt
user_2@pavels-pavilion:~$ cat user_2_File_1.txt ../user_3/user_3_File_1.txt > user_23_File_11.txt
cat: ../user_3/user_3_File_1.txt: Отказано в доступе
user_2@pavels-pavilion:~$ cat user_2_File_1.txt ../user_3/user_3_File_2.txt > user_23_File_12.txt
cat: ../user_3/user_3_File_2.txt: Отказано в доступе
user_2@pavels-pavilion:~$ cat user_2_File_1.txt ../user_3/user_3_File_3.txt > user_23_File_13.txt
user_2@pavels-pavilion:~$ cat user_2_File_1.txt ../user_5/user_5_File_1.txt > user_23_File_11.txt
cat: ../user_5/user_5_File_1.txt: Отказано в доступе
user_2@pavels-pavilion:~$ cat user_2_File_1.txt ../user_5/user_5_File_1.txt > user_25_File_11.txt
cat: ../user_5/user_5_File_1.txt: Отказано в доступе
user_2@pavels-pavilion:~$ cat user_2_File_1.txt ../user_5/user_5_File_2.txt > user_25_File_12.txt
cat: ../user_5/user_5_File_2.txt: Отказано в доступе
user_2@pavels-pavilion:~$ cat user_2_File_1.txt ../user_5/user_5_File_3.txt > user_25_File_13.txt
user_2@pavels-pavilion:~$ ls
user_23_File_11.txt  user_23_File_12.txt  user_23_File_13.txt  user_25_File_11.txt  user_25_File_12.txt  user_25_File_13.txt  user_2_File_1.txt  user_2_File_2.txt  user_2_File_3.txt
user_2@pavels-pavilion:~$ ls -l
итого 9
-rw-rw-r-- 1 user_2 user_2 15 ноя 28 23:32 user_23_File_11.txt
-rw-rw-r-- 1 user_2 user_2 15 ноя 28 23:32 user_23_File_12.txt
-rw-rw-r-- 1 user_2 user_2 29 ноя 28 23:32 user_23_File_13.txt
-rw-rw-r-- 1 user_2 user_2 15 ноя 28 23:33 user_25_File_11.txt
-rw-rw-r-- 1 user_2 user_2 15 ноя 28 23:33 user_25_File_12.txt
-rw-rw-r-- 1 user_2 user_2 29 ноя 28 23:33 user_25_File_13.txt
```

2. Запустите в фоновом (background) режиме командный файл (процесс), выдающий в цикле с некоторой задержкой сообщение на консоль.

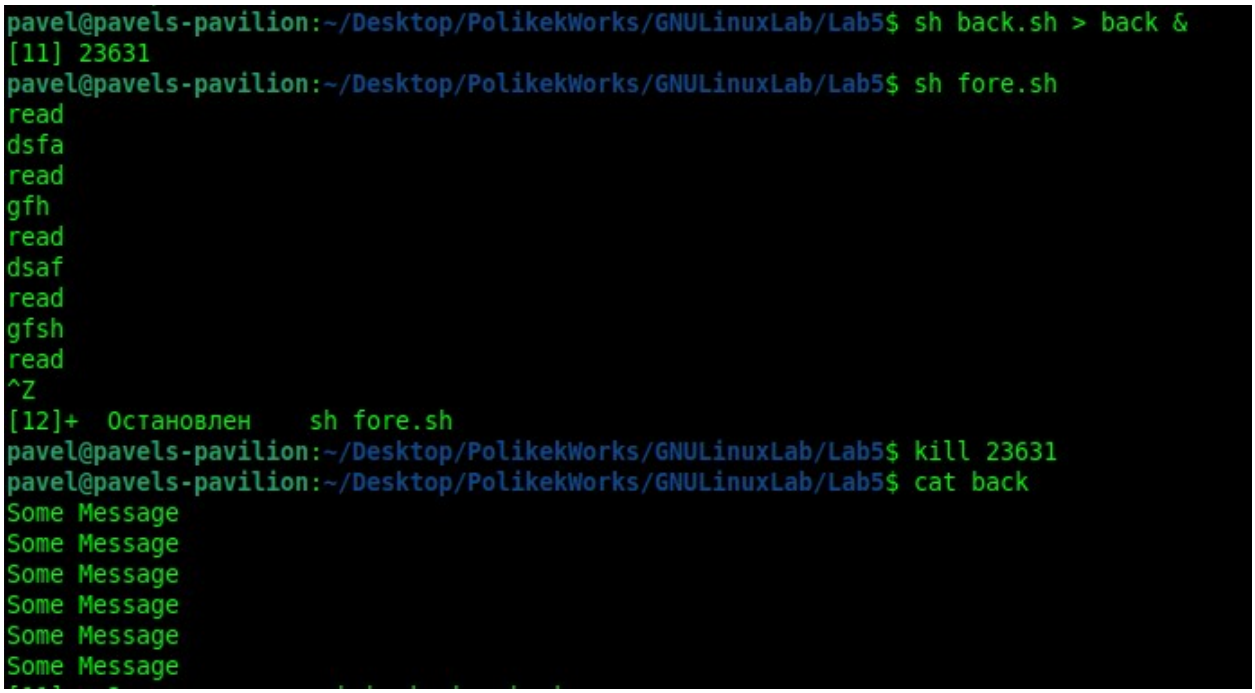
```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab5$ cat back.sh
while true
do
  echo "Some Message"
  sleep 5
done
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab5$ cat fore.sh
while true
do
  echo "read"
  read somevar
done
```

Запустите другой командный файл (процесс), требующий диалога, в обычном режиме (foreground). Убедитесь в том, что вывод этих двух процессов на консоль перемежается.

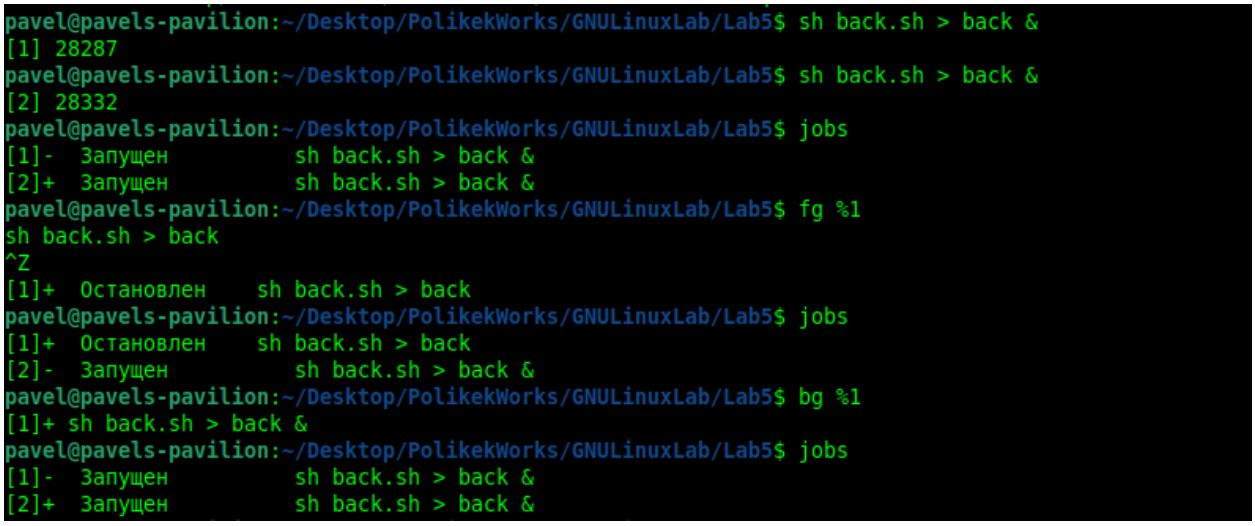
```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab5$ sh back.sh &
[8] 23114
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab5$ Some Message
sh fore.sh
read
Some Message
Some Message

read
Some Message

read
Some Message
```

Остановите фоновый процесс сигналом kill . Запустите его снова, организовав предварительно перенаправление его

вывода в файл. Убедитесь, что теперь вывод двух процессов разделен.



3. Доработайте предыдущее задание так, чтобы показать возможность перевода фонового процесса в диалоговый режим (foreground) для выполнения операции ввода с клавиатуры и затем возврата его обратно в фоновый (background) режим (команды fg, bg, jobs).



Продемонстрируйте возможность оставления фонового процесса на исполнение после завершения пользовательского сеанса работы в ОС.

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab5$ jobs
[1]-  Запущен          sh back.sh > back &
[2]+  Запущен          sh back.sh > back &
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab5$ su - user_2
Пароль:
user_2@pavels-pavilion:~$ jobs
user_2@pavels-pavilion:~$
```

4. Разработайте командный файл для выполнения архивации каталога через определенные интервалы времени. Запустите командный файл в режиме background . Имя архивируемого каталога, местоположение архива и время (период) архивации передаются при запуске командного файла в виде параметров командной строки.

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab5$ sh zip.sh ../../../../Desktop/PolikekWorks/GNULinuxLab/Lab5 Catalog 5
  adding: Catalog/ (stored 0%)
updating: Catalog/ (stored 0%)
updating: Catalog/ (stored 0%)
updating: Catalog/ (stored 0%)
```

Вывод: в ходе выполнения работы были изучены способы создание новых учётных записей и создания групп. Были изучены методы по перемещению процессов в back и fore ground

Лабораторная работа № 6

«ГЕНЕРАЦИЯ И ОБРАБОТКА СИГНАЛОВ»

1. Войдите в систему и скопируйте с разделяемого ресурса в свой HOME-каталог набор исходных файлов для шестого занятия.

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab6$ ls
rlimit.cpp  sigint.cpp  signal_alarm.cpp  signal_catch.cpp  sigusr.cpp
```

2. Программа sigint.cpp осуществляет ввод символов со стандартного ввода. Скомпилируйте и запустите программу и отправьте ей сигналы SIGINT (нажатием Ctrl-C) и SIGQUIT (нажатием Ctrl-\). Проанализируйте результаты.

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab6$ g++ sigint.cpp -o sigint
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab6$ ./sigint
Enter a string:
^CAhhh! SIGINT!
fgets: Interrupted system call
```

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab6$ ./sigint
Enter a string:
^\Выход (образ памяти сброшен на диск)
```

Результат: при нажатие CTRL + C программа выводит сообщение: «ahhh! SIGINT!». При нажатие CTRL + \ программа завершается с ошибкой.

3. Запустите программу signal_catch.cpp , выполняющую вывод на консоль. Отправьте процессу сигналы SIGINT и SIGQUIT , а также SIGSTOP (нажатием Ctrl-Z) и SIGCONT (нажатием Ctrl-Q) . Проанализируйте поведение процесса и вывод на консоль, а также сравните с программой из предыдущего пункта.

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab6$ ./signal_catch
0
1
2
3
^C
Signal 2 received.
4
5
^\
Signal 3 received.
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab6$ ./signal_catch
0
1
2
3
^Z
[1]+  Остановлен    ./signal_catch
```

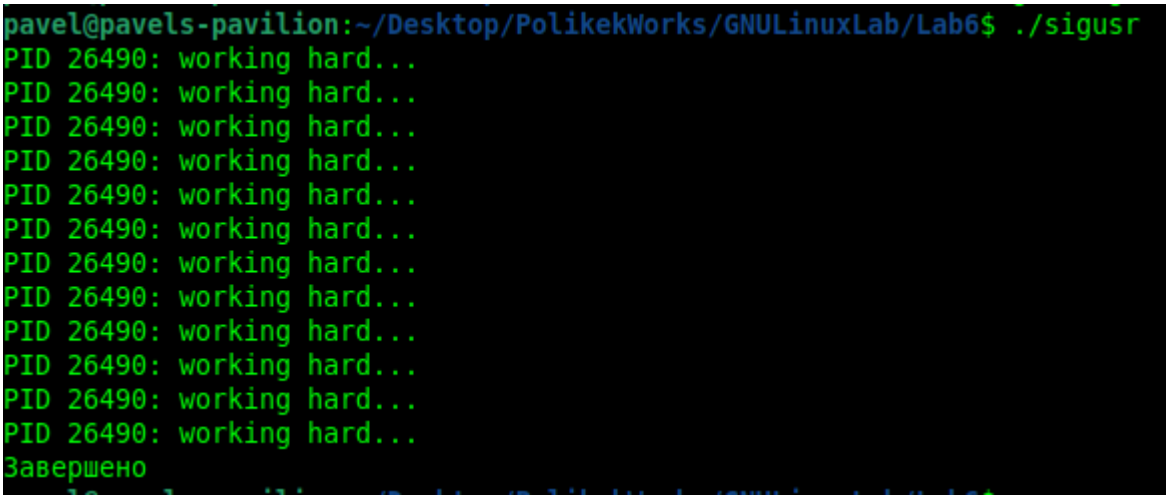
Результат: при нажатие CTRL + C программа выводит сообщение: «Signal 2 received».

При нажатие CTRL + \ программа выводит сообщение: «Signal 3 received».

При нажатие CTRL + Z программа останавливается.

При нажатие CTRL + Q не было замечено видимого результата.

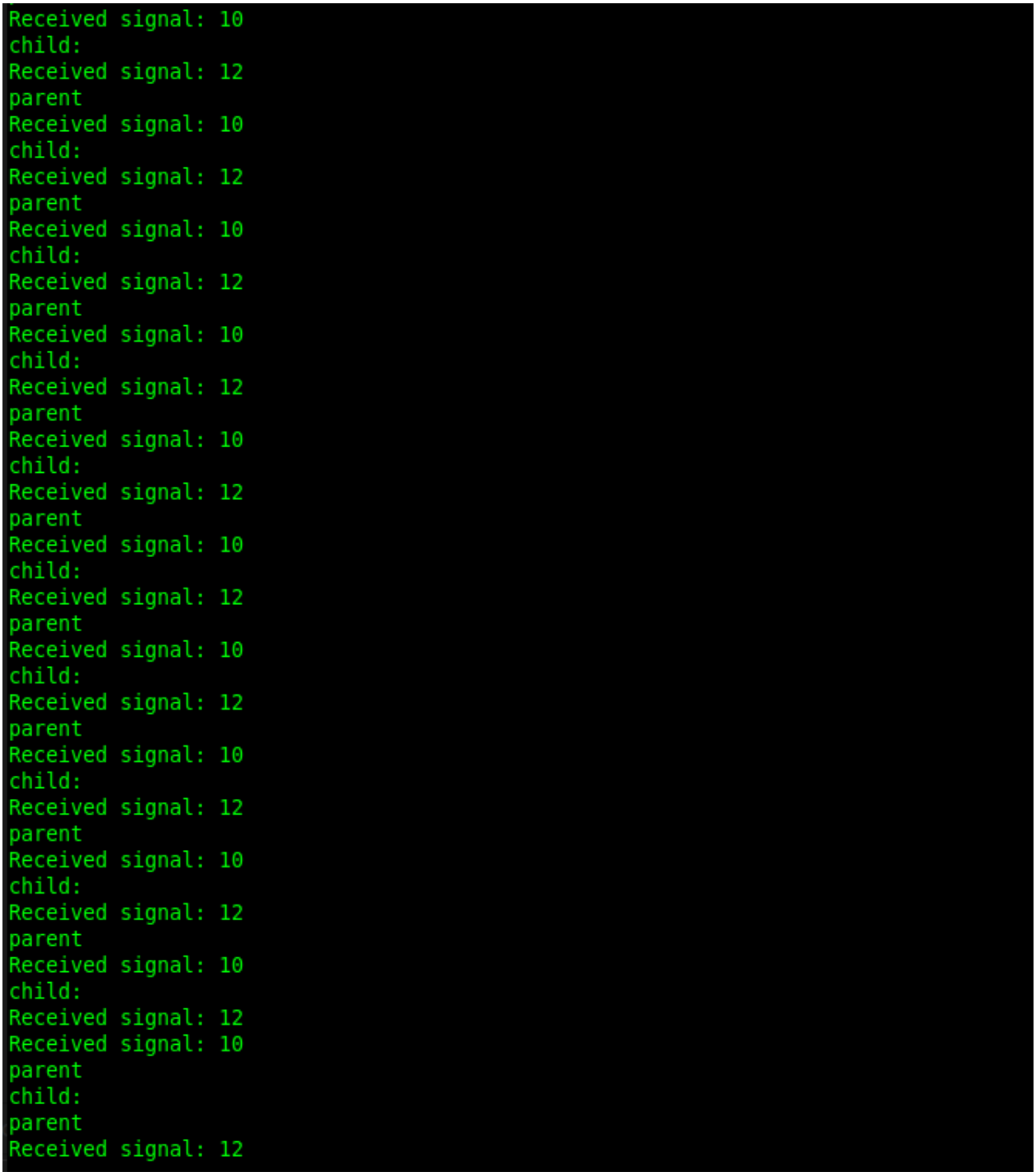
4. Скомпилируйте и запустите программу sigusr.cpp . Программа выводит на консоль значение ее PID и зацикливается, ожидая получения сигнала. Запустите второй терминал и, отправляя с него командой kill различные сигналы, в том числе и SIGUSR1 , проанализируйте реакцию на них.



Результат: после выполнения команды kill с PID работающего процесса из второго терминала, процесс завершает свою работу

5. Составьте программу, запускающую процесс-потомок. Процесс-родитель и процесс-потомок должны генерировать (можно случайным образом) и отправлять друг другу сигналы (например, SIGUSR1, SIGUSR2). Каждый из процессов должен выводить на консоль информацию об отправленном и о полученном сигналах.

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab6$ cat myProgramm.cpp
#include <unistd.h>
#include <csignal>
#include <iostream>

void signalHandler(int signum) {
    std::cout << "Received signal: " << signum << '\n';
}

int main()
{
  signal(SIGINT, signalHandler);

  long long int PID = fork();
  switch (PID)
  {
    case (-1):
      std::cerr << "Error!";
      return 1;

    case (0):
      //child
      signal(SIGUSR1, signalHandler);

      while (true)
      {
        std::cout << "child:\n";
        sleep(1000);
        kill(getppid(), SIGUSR2);
      }
      return 0;

    default:
    //parent
    signal(SIGUSR2, signalHandler);

    while (true) {
      std::cout << "parent\n";
      sleep(1000);
      kill(PID, SIGUSR1);
    }
    return 0;
  }
}
```



6. Для организации обработчиков сигналов предпочтительно использовать системный вызов sigaction() и соответствующую структуру данных. Обеспечьте корректное завершение процессов.

```cpp
#include <iostream>
#include <cstdlib>
#include <unistd.h>
#include <signal.h>

void signalHandler(int signal) {
    // Обработчик сигналов
    std::cout << "Процесс " << getpid() << " получил сигнал: " << signal << std::endl;
}

int main() {
    // Установка обработчика сигналов
    struct sigaction sa;
    sa.sa_handler = signalHandler;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    sigaction(SIGUSR1, &sa, nullptr);

    // Создание процесса-потомка
    pid_t pid = fork();

    if (pid == -1) {
        std::cerr << "Ошибка при создании процесса-потомка" << std::endl;
        return 1;
    } else if (pid == 0) {
        // Код для процесса-потомка
        srand(getpid());

        for (int i = 0; i < 5; i++) {
            // Генерация случайного сигнала
            int signal = rand() % 10 + 1;

            // Отправка сигнала процессу-родителю
            kill(getppid(), signal);
            std::cout << "Процесс-потомок " << getpid() << " отправил сигнал " << signal << std::endl;

            // Задержка
            sleep(1);
        }
    } else {
        // Код для процесса-родителя
        srand(getpid());

        for (int i = 0; i < 5; i++) {
            // Генерация случайного сигнала
            int signal = rand() % 10 + 1;

            // Отправка сигнала процессу-потомку
            kill(pid, signal);
            std::cout << "Процесс-родитель " << getpid() << " отправил сигнал " << signal << std::endl;

            // Задержка
            sleep(1);
        }
    }

    return 0;
}
```

```
Процесс-родитель 101669 отправил сигнал 6
Процесс-родитель 101669 отправил сигнал 3
Процесс-родитель 101669 отправил сигнал 10
Процесс-родитель 101669 отправил сигнал 2
Процесс-родитель 101669 отправил сигнал 6
```

Вывод: были изучены основы создания и обработки сигналов. Были написаны практические программы взаимодействующие с сигналами