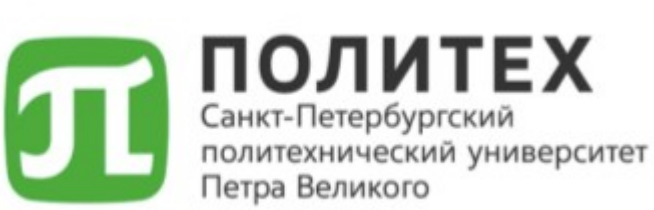
Федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический университет
Петра Великого»
Институт компьютерных наук и технологий
Фундаментальная информатика и информационные технологии



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

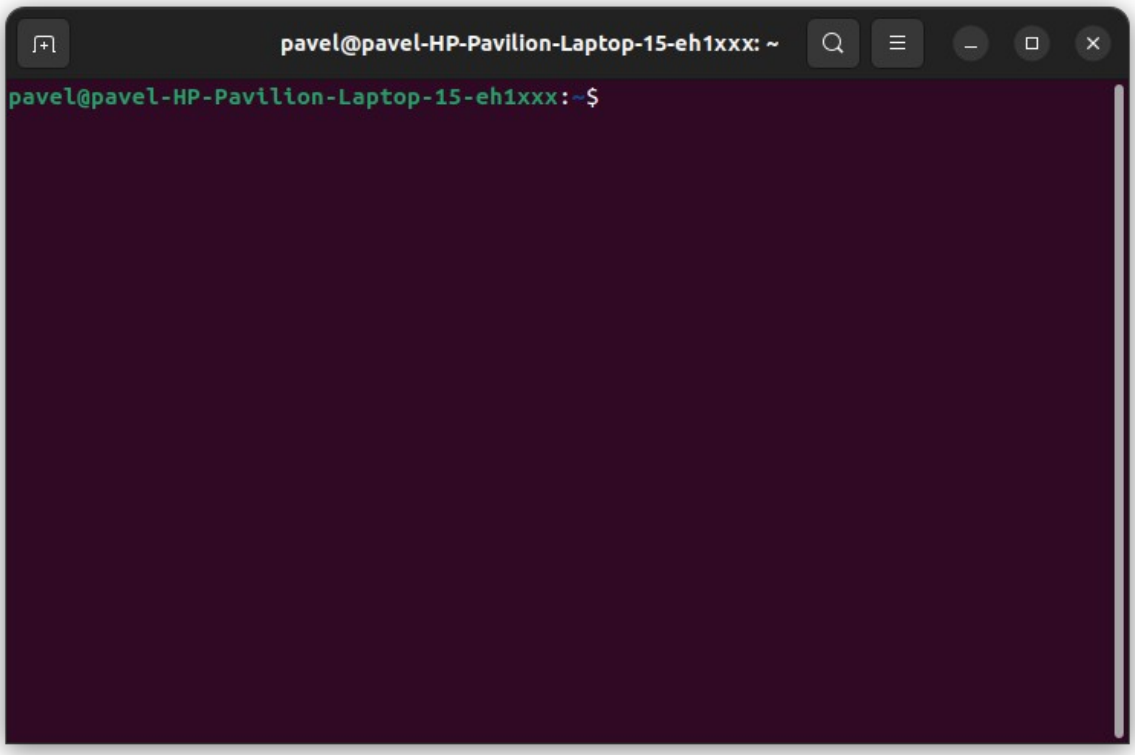
Оглавление:

- Лабораторная работа № 1 «БАЗОВЫЕ КОМАНДЫ ОС» стр 3-5
- Лабораторная работа № 2 «ЗАПУСК И ЗАВЕРШЕНИЕ ПРОЦЕССОВ» стр 6-8
- Лабораторная работа № 3 «ПРОГРАММНЫЕ КАНАЛЫ» стр 11-16
- Лабораторная работа № 4 «КОМАНДНЫЕ ФАЙЛЫ. ПЕРЕМЕННЫЕ ОКРУЖЕНИЯ» стр 17-20
- Лабораторная работа № 5 «УЧЕТНЫЕ ЗАПИСИ. ФОНОВЫЙ И ДИАЛОГОВЫЙ РЕЖИМЫ ИСПОЛНЕНИЯ ПРОЦЕССОВ» стр 21-25
- Лабораторная работа № 6 «ГЕНЕРАЦИЯ И ОБРАБОТКА СИГНАЛОВ» стр 26-31
- Лабораторная работа № 7 «СЕМАФОРЫ И СИНХРОНИЗАЦИЯ» стр 32-40
- Лабораторная работа № 8 «ОБМЕН ЧЕРЕЗ ОЧЕРЕДИ СООБЩЕНИЙ» стр 41-46
- Лабораторная работа № 9 «РАБОТА С РАЗДЕЛЯЕМОЙ ПАМЯТЬЮ» стр 47-51
- Лабораторная работа № 10 «СОЗДАНИЕ СОЕДИНЕНИЙ НА СОКЕТАХ» стр 52-55

- Лабораторная работа № 11 «ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ ПО СЕТИ» стр 56-62

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
```

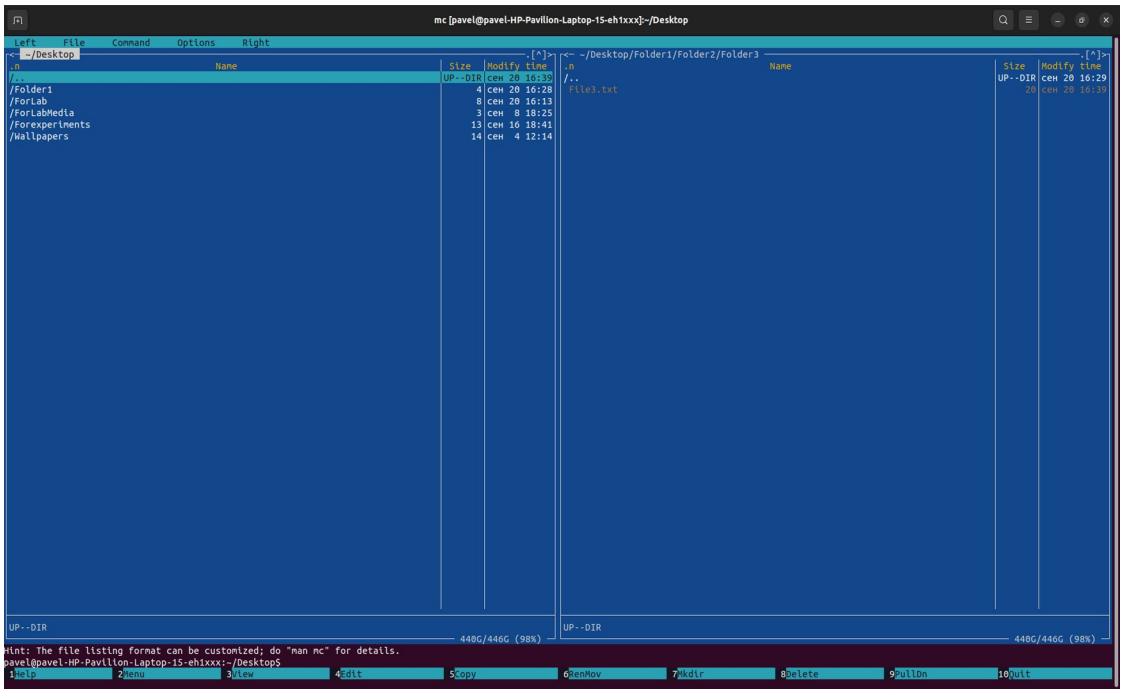
Полное описание синтаксиса и семантики этих и любых других команд можно увидеть в системе помощи ОС Linux, вызы-ваемой с терминала в виде man < интересующая вас команда >, или запускайте веб-браузер и используйте всю информационную мощь Интернета. +

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

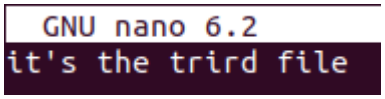
5. Создайте дерево каталогов глубиной вложения до трех уров-ней, а в самих каталогах создайте текстовые файлы. Примените различные способы создания новых файлов.

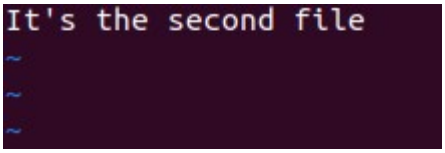
```
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop$ mkdir Folder1
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop$ cd Folder1
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/Folder1$ cat >File1.txt
It's first file
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/Folder1$ mkdir Folder2
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/Folder1$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

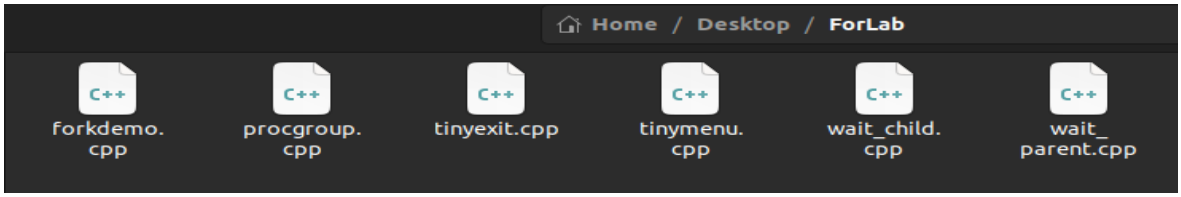

```
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab1$ mkdir darkroom
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab1$ cd darkroom
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab1/darkroom$ vim secretfile.txt
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab1/darkroom$ 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»

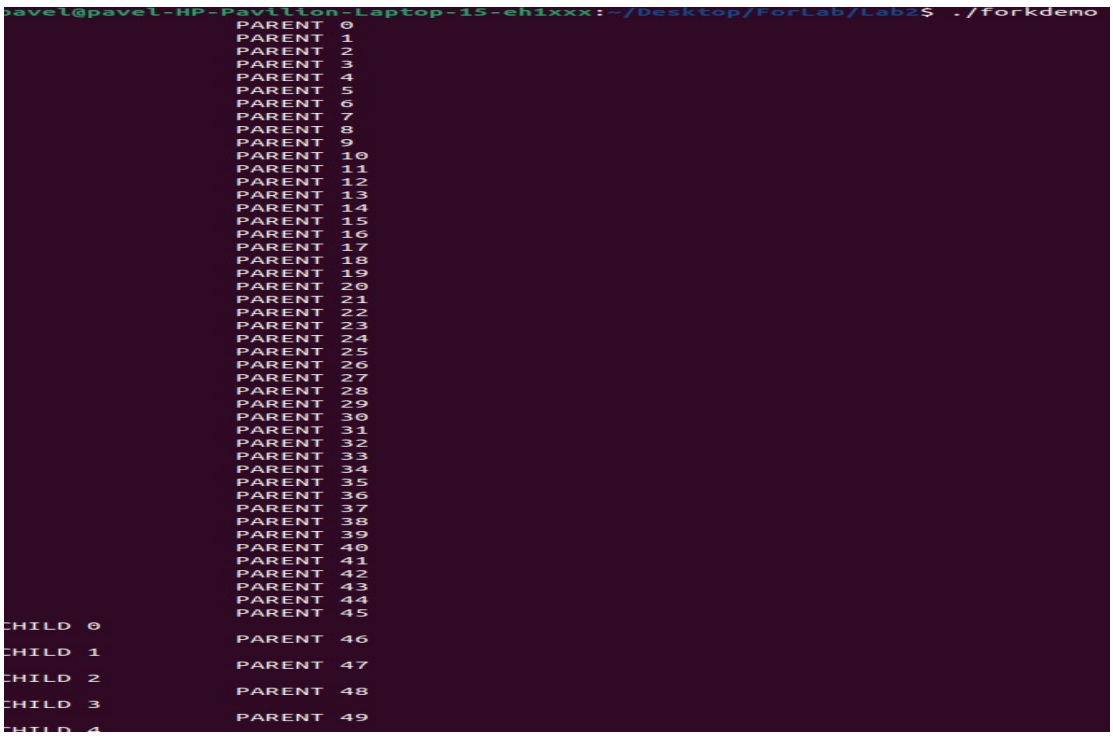
1. Войдите в систему и скопируйте в свой HOME-каталог с разделяемого ресурса набор исходных файлов для второй лабораторной работы. +



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.

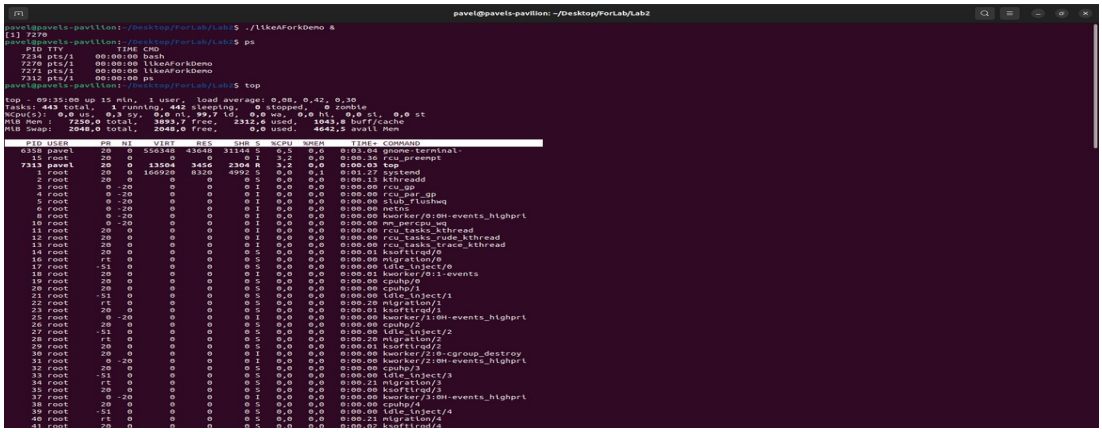
```
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab2$ g++ forkdemo.cpp -o forkdemo
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab2$ g++ tinymenu.cpp -o tinymenu
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab2$ g++ tinyexit.cpp -o tinyexit
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab2$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

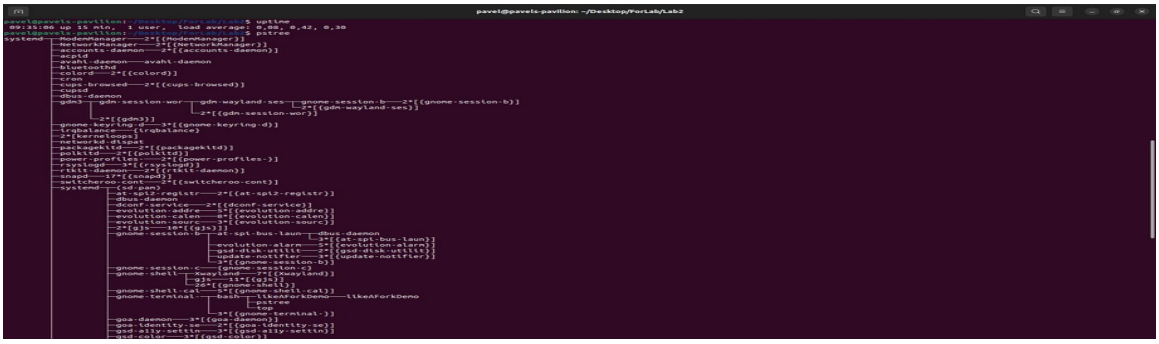


```
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab2$ ./tinymenu
0=who, 1=ls, 2=date:0
pavel    tty2         2023-09-20 15:46 (tty2)
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab2$ ./tinyexit
0=who, 1=ls, 2=date:0
pavel    tty2         2023-09-20 15:46 (tty2)
0=who, 1=ls, 2=date:1
2_processes.pdf  forkdemo  forkdemo.cpp  likeAForkDemo.cpp  procgroup  procgroup.cpp  tinyexit  tinyexit.cpp  tinymenu  tinymenu.cpp  wait_child  wait_child.cpp  wait_parent  wait_parent.cpp
0=who, 1=ls, 2=date:2
Ср 20 сен 2023 17:23:06 MSK
0=who, 1=ls, 2=date:^Z
[1]+  Stopped                 ./tinyexit
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab2$ ./procgroup


Initial process          PID  60465     PPID  60188     GID  60465

New process              PID  60466     PPID  60465     GID  60465
New process              PID  60467     PPID  60465     GID  60465
New process              PID  60468     PPID  60465     GID  60465
New process              PID  60469     PPID  60466     GID  60465
New process              PID  60470     PPID  60466     GID  60465
New process              PID  60471     PPID  60467     GID  60465
New process              PID  60472     PPID  60469     GID  60465
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab2$ ./wait_parent
Forked child 60595
Forked child 60596
Forked child 60597
Child 60595 is terminating with exit (00B3)
Child 60597 is terminating with exit (00B5)
Child 60596 is terminating with signal 0009
Wait on PID: 60595 returns status of: B300
Wait on PID: 60596 returns status of: 0009
Wait on PID: 60597 returns status of: B500
pavel@pavel-HP-Pavilion-Laptop-15-eh1xxx:~/Desktop/ForLab/Lab2$ ./wait_child
Segmentation fault (core dumped)
```

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;
}
```

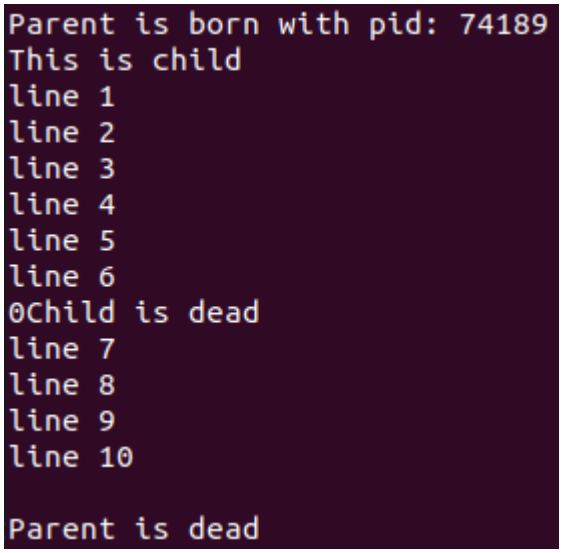
```
pavel@pavels-pavilion:~/Desktop/ForLab/Lab2$ g++ myProgram.cpp -o myProgram
pavel@pavels-pavilion:~/Desktop/ForLab/Lab2$ ./myProgram
Parent is born with pid: 53071
Child is born
^Z
[1]+  Остановлен    ./myProgram
pavel@pavels-pavilion:~/Desktop/ForLab/Lab2$ pwdx 53071
53071: /home/pavel/Desktop/ForLab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

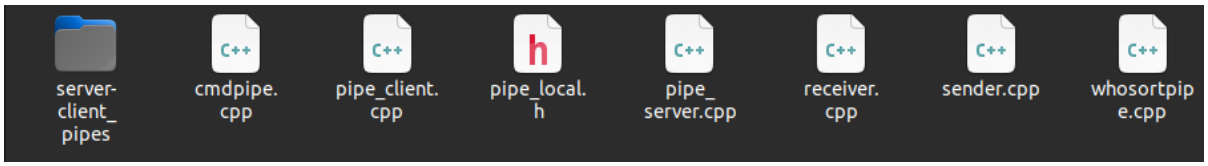
Вывод: *в ходе выполнения лабораторной работы были изучены взаимодействия родительских и потомственных процессов.*

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)
```

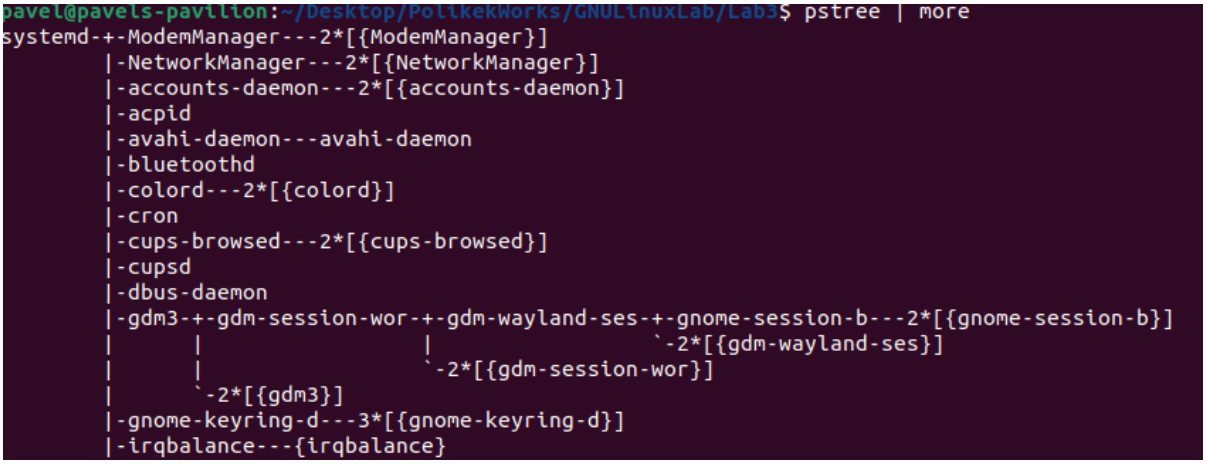
pstree и more).

```
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
```

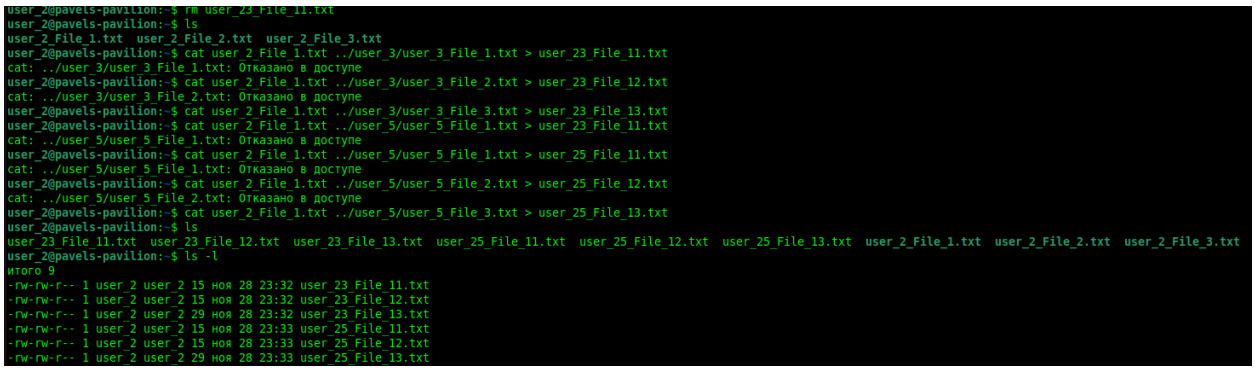
Заходя в систему под разными аккаунтами, создайте в соответствующих домашних каталогах файлы, варьируя при этом права доступа для пользователя, для группы, для всех.

```
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.



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

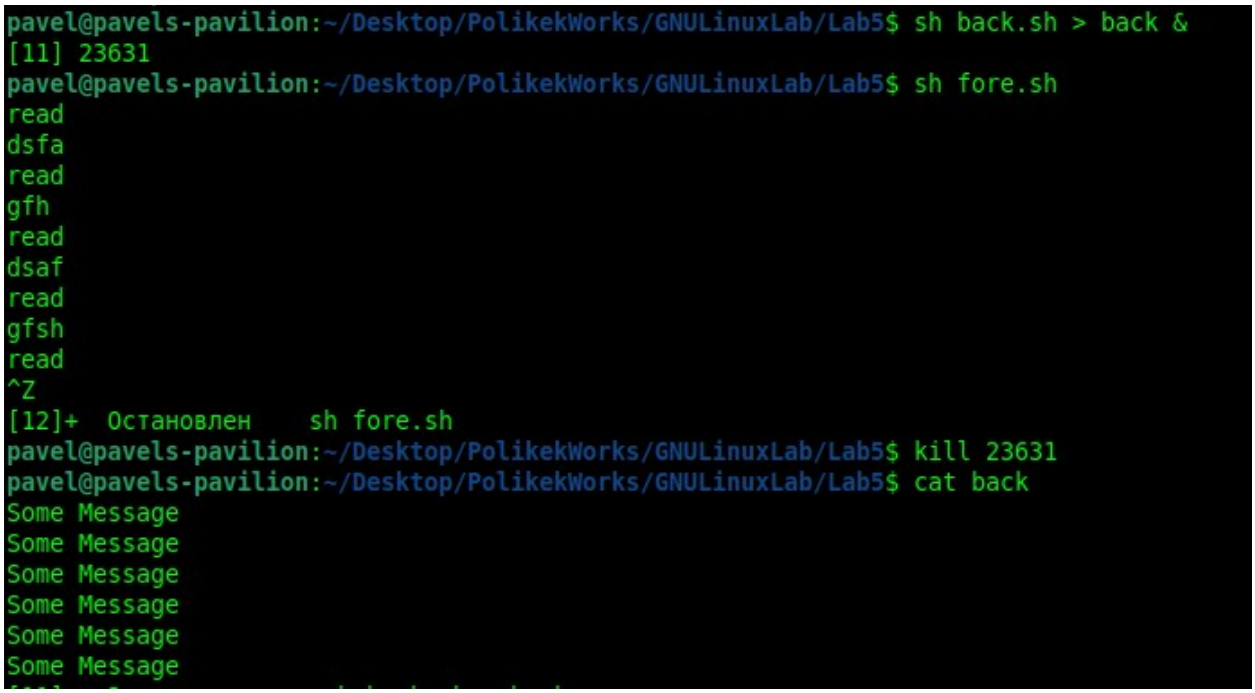
```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab5$ cat back.sh
while true
do
  echo "Some Message"
  sleep 5
done
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab5$ cat fore.sh
while true
do
  echo "read"
  read somevar
done
```

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

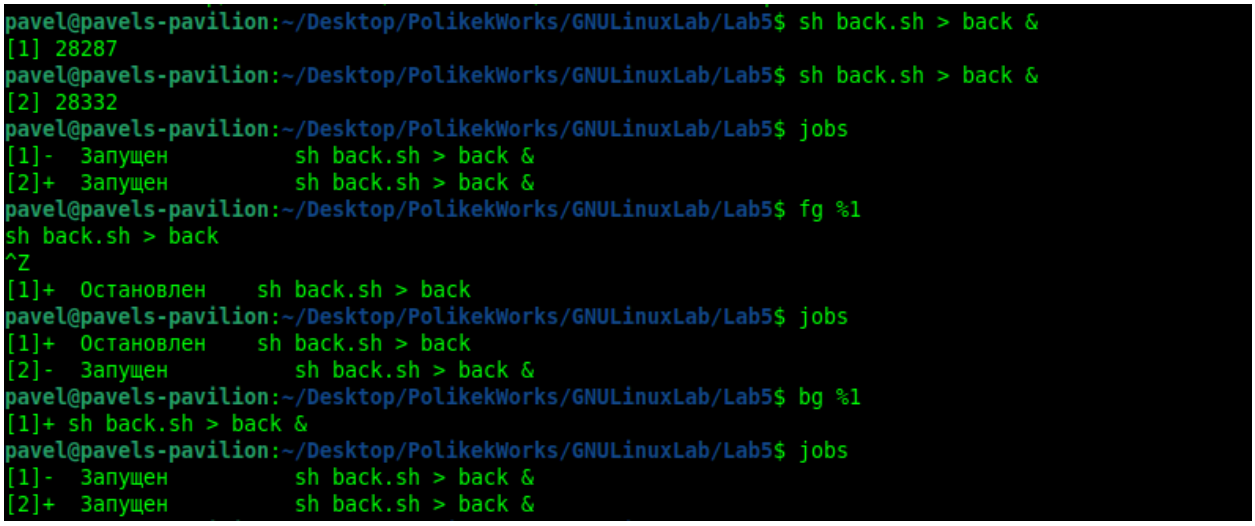
```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab5$ sh back.sh &
[8] 23114
pavel@pavels-pavilion:~/Desktop/PolikekWorks/GNULinuxLab/Lab5$ Some Message
sh fore.sh
read
Some Message
Some Message

read
Some Message

read
Some Message
```

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

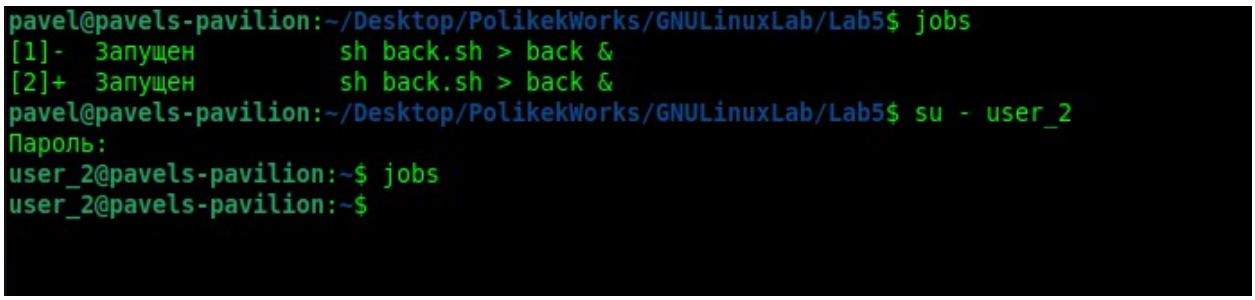
вывода в файл. Убедитесь, что теперь вывод двух процессов разделен.



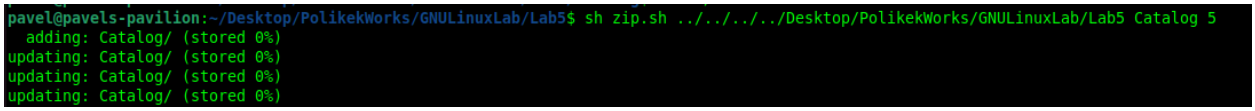
3. Доработайте предыдущее задание так, чтобы показать возможность перевода фонового процесса в диалоговый режим (foreground) для выполнения операции ввода с клавиатуры и затем возврата его обратно в фоновый (background) режим (команды fg, bg, jobs).



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



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.



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
```

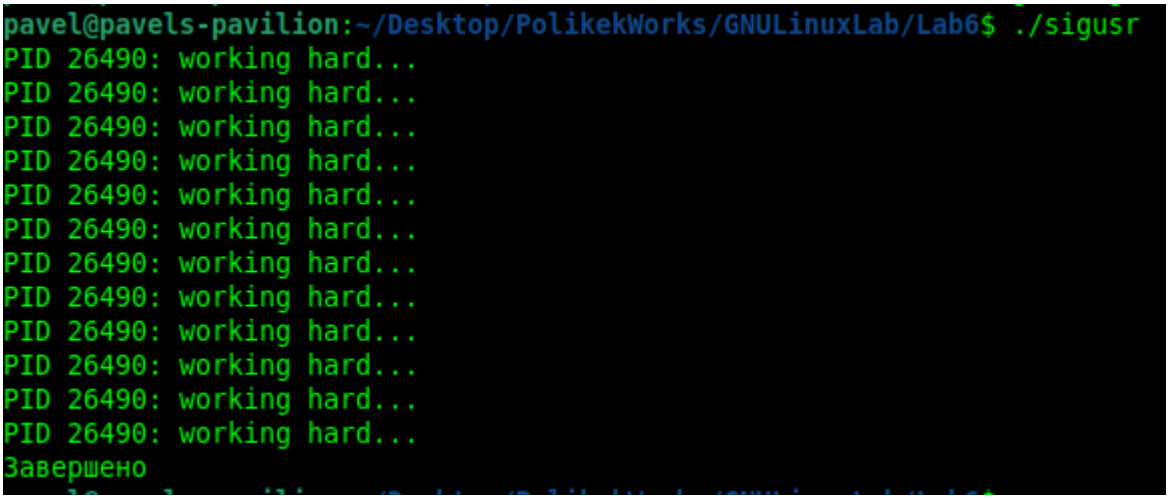
Результат: при нажатие CTRL + C программа выводит сообщение: «Signal 2 received».

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

При нажатие CTRL + Z программа останавливается.

При нажатие CTRL + Q не было замечено видимого результата.

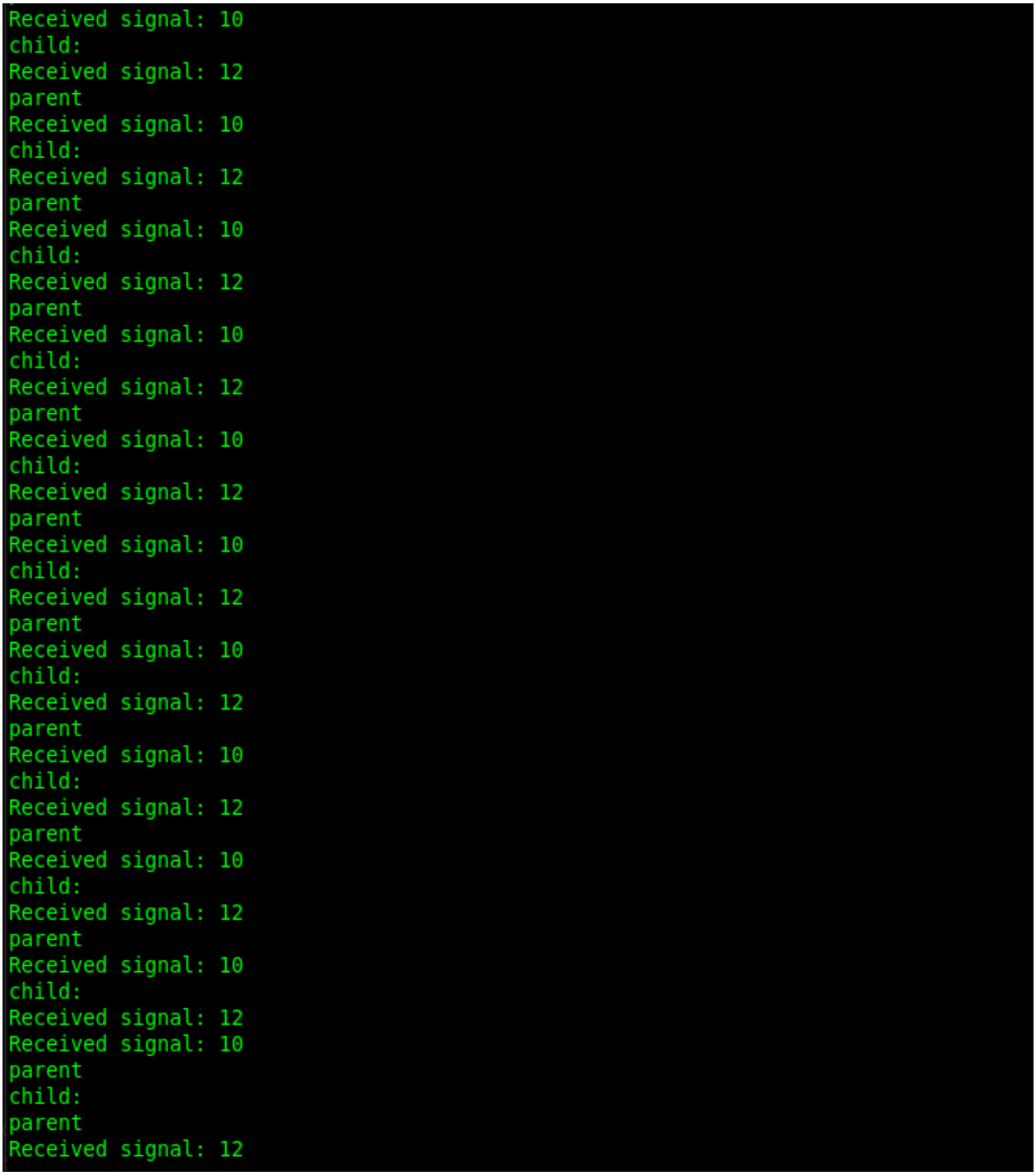
4. Скомпилируйте и запустите программу sigusr.cpp . Программа выводит на консоль значение ее PID и зацикливается, ожидая получения сигнала. Запустите второй терминал и, отправляя с него командой kill различные сигналы, в том числе и SIGUSR1 , проанализируйте реакцию на них.



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

Лабораторная работа №7
«СЕМАФОРЫ И СИНХРОНИЗАЦИЯ»

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

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab7$ ls
gener_sem.cpp  semdemo.cpp  sem_prod_con.cpp  semrm.cpp
```

2. Скомпилируйте и выполните программу gener_sem.cpp , иллюстрирующую создание наборов с семафорами или получение доступа к ним. Запустите программу несколько раз и после каждого ее завершения выполните команду ipcs -s . Поясните зависимость процедуры создания семафоров от используемых в вызове semget() флагов.

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab7$ g++ gener_sem.cpp -o gener_sem
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab7$ ./gener_sem
sem1 identifier is 4
semget: IPC_CREATE | IPC_EXCL | 0666: File exists
sem2 identifier is -1
sem3 identifier is 9
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab7$ ipcs -s

------ Массивы семафоров --------
ключ    semid      владелец права nsems
0xcc260022 2          pavel      600        1
0xcb260022 3          pavel      600        1
0x532605e8 4          pavel      666        3
0x00000000 5          pavel      600        3
0x00000000 6          pavel      600        3
0x00000000 7          pavel      600        3
0x00000000 8          pavel      600        3
0x00000000 9          pavel      600        3

pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab7$ ./gener_sem
sem1 identifier is 4
semget: IPC_CREATE | IPC_EXCL | 0666: File exists
sem2 identifier is -1
sem3 identifier is 10
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab7$ ipcs -s

------ Массивы семафоров --------
ключ    semid      владелец права nsems
0xcc260022 2          pavel      600        1
0xcb260022 3          pavel      600        1
0x532605e8 4          pavel      666        3
0x00000000 5          pavel      600        3
0x00000000 6          pavel      600        3
0x00000000 7          pavel      600        3
0x00000000 8          pavel      600        3
0x00000000 9          pavel      600        3
0x00000000 10         pavel      600        3

pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab7$ ./gener_sem
sem1 identifier is 4
semget: IPC_CREATE | IPC_EXCL | 0666: File exists
sem2 identifier is -1
sem3 identifier is 11
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab7$ ipcs -s

------ Массивы семафоров --------
ключ    semid      владелец права nsems
0xcc260022 2          pavel      600        1
0xcb260022 3          pavel      600        1
0x532605e8 4          pavel      666        3
0x00000000 5          pavel      600        3
0x00000000 6          pavel      600        3
0x00000000 7          pavel      600        3
0x00000000 8          pavel      600        3
0x00000000 9          pavel      600        3
0x00000000 10         pavel      600        3
0x00000000 11         pavel      600        3
```

При запуске данной программы многократно будет видно, что набор sem1 будет создан единожды, а затем каждая новая попытка будет всего лишь открывать доступ к уже существующему ресурсу; попытки создания набора sem2 на том же ключе всегда будут приводить к ошибке из-за наличия флагов IPC_CREATE | IPC_EXCL, не допускающих открытия ресурса вместо его создания; набор sem3 будет создаваться

при каждом новом запуске программы. Причем каждый раз с новым уникальным идентификатором.

3. Удалите созданные на предыдущем шаге семафоры с помощью команды ipcrm с соответствующей опцией и значением id семафора или ключа.

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab7$ ipcrm -a
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab7$ ipcs -s

------ Массивы семафоров --------
ключ   semid      владелец права nsems
```

4. Скомпилируйте semdemo.cpp , демонстрирующую организацию разделения доступа к общему ресурсу между несколькими процессами с помощью технологии семафоров. Запустите сразу несколько процессов на разных терминалах и проанализируйте их взаимодействие и соблюдение очередности в попытках получения общего ресурса.

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab7$ g++ semdemo.cpp -o semdemo
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab7$ ./semdemo
press Enter

Press Enter to lock:
Trying to lock...
Locked.
Press Enter to unlock:
Unlocked
```

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab7$ ./semdemo
Press Enter to lock:
Trying to lock...
Locked.
Press Enter to unlock:
Unlocked
```

При попытке одновременной блокировки из разных терминалов, сначала ожидалась разблокировка из первого терминала, после чего сразу же начиналась блокировка в следующем терминале

5. Скомпилируйте программу semrm.cpp и произведите с ее помощью удаление созданного на предыдущем шаге семафора. Поясните, почему данная программа удаляет только те семафоры, которые были созданы при выполнении программы semdemo.cpp .

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab7$ g++ semrm.cpp -o semrm
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab7$ ipcs -s

------ Массивы семафоров --------
ключ   semid      владелец права nsems
0x4a2605e8 12         pavel      666        1

pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab7$ ./semrm
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab7$ ipcs -s

------ Массивы семафоров --------
ключ   semid      владелец права nsems
```

semrm удаляет именно семафор созданный semdemo, т.к. в обоих программах используется одинаковый ключ

6. Попробуйте удалить семафор с помощью запуска semrm.cpp во время исполнения semdemo.cpp и проанализируйте ситуацию.

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab7$ ipcs -s

------ Массивы семафоров --------
ключ   semid      владелец права nsems
0x4a2605e8 13         pavel      666        1

pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab7$ ./semrm
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab7$ ipcs -s

------ Массивы семафоров --------
ключ   semid      владелец права nsems
```

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab7$ ./semdemo
press Enter

initsem: Invalid argument
```

При удаление семафора во время выполнения semdemo с помощью semrm, семофор успешно удаляется, но программа semdemo завершается с ошибкой

7. Попытайтесь улучшить программу semdemo.cpp , например, предоставив процессу возможность после освобождения ресурса становиться снова в очередь на повторное его занятие (а не завершаться), организовав при этом завершение процесса по вводу какого-либо символа.

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab7$ g++ semdemo.cpp -o semdemo
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab7$ ./semdemo
Press Enter to lock:
Trying to lock...
Locked.
Press Enter to unlock:
Unlocked

Press Enter to lock:
Trying to lock...
Locked.
Press Enter to unlock:
Unlocked

Press Enter to lock:
Trying to lock...
Locked.
Press Enter to unlock:
Unlocked

Press Enter to lock:
Trying to lock...
Locked.
Press Enter to unlock:
Unlocked
E
```

Код программы:

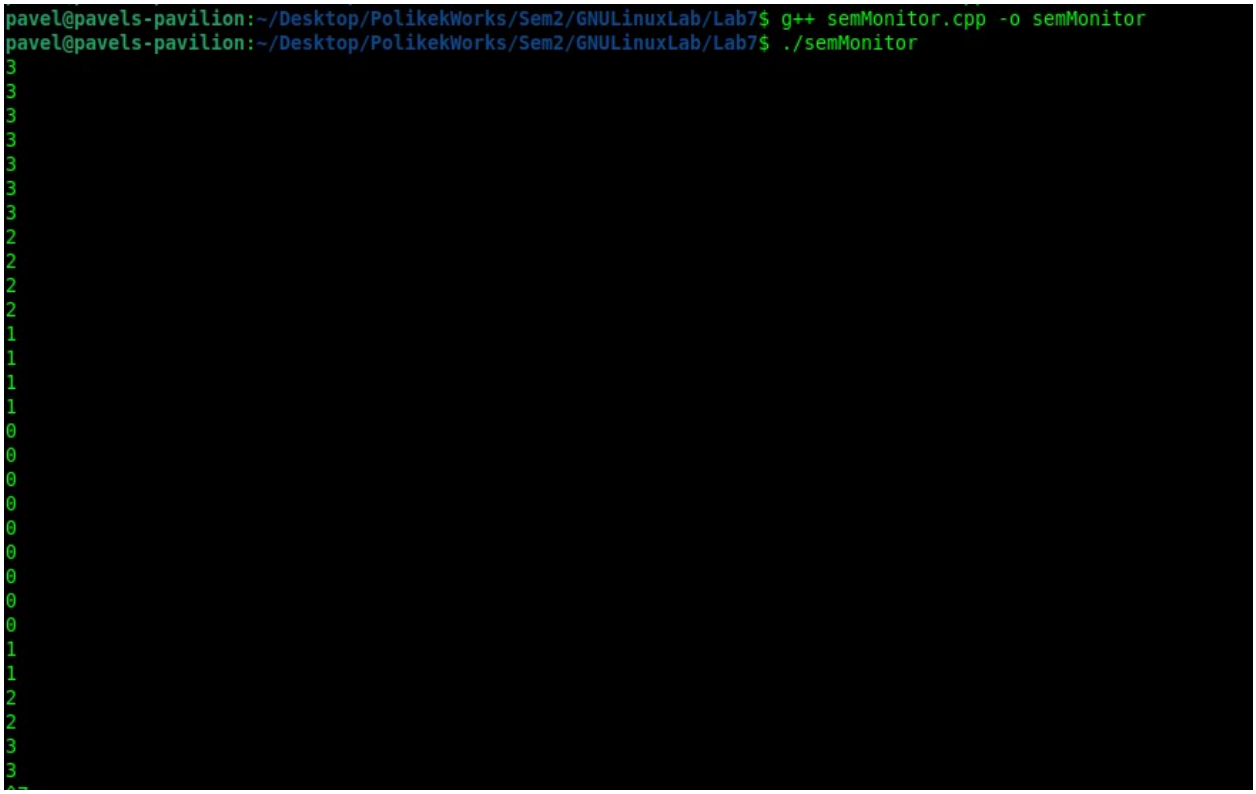
```
/*
** semdemo.cpp -- demonstrates semaphore use like a file locking
mechanism
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <iostream>

#define MAX_RETRIES 10

union semun {
     int val;
     struct semid_ds *buf;
     ushort *array;
};


int initsem(key_t key, int nsems)  /* key from ftok() */
{
     int i;
     union semun arg;
     struct semid_ds buf;
```

```
     struct sembuf sb;
     int semid;

     semid = semget(key, nsems, IPC_CREAT | IPC_EXCL | 0666);

     if (semid >= 0) { /* we got it first */
          sb.sem_op = 1; sb.sem_flg = 0;
          arg.val = 1;

          printf("press Enter\n"); getchar();

          for(sb.sem_num = 0; sb.sem_num < nsems; sb.sem_num++)
{
               /* do a semop() to "free" the semaphores. */
               /* this sets the sem_otime field, as needed below.
*/
               if (semop(semid, &sb, 1) == -1) {
                    int e = errno;
                    semctl(semid, 0, IPC_RMID); /* clean up */
                    errno = e;
                    return -1; /* error, check errno */
               }
          }

     } else if (errno == EEXIST) { /* someone else got it first
*/
          int ready = 0;

          semid = semget(key, nsems, 0); /* get the id */
          if (semid < 0) return semid; /* error, check errno */

          /* wait for other process to initialize the semaphore:
*/
          arg.buf = &buf;
          for(i = 0; i < MAX_RETRIES && !ready; i++) {
               semctl(semid, nsems-1, IPC_STAT, arg);
               if (arg.buf->sem_otime != 0) {
                    ready = 1;
               } else {
                    sleep(1);
               }
          }
          if (!ready) {
               errno = ETIME;
               return -1;
          }
     } else {
          return semid; /* error, check errno */
     }

     return semid;
```

```
}


int main(void)
{
     key_t key;
     int semid;
     char u_char = 'J';
     struct sembuf sb;

  char charForRead = 'A';

  while (charForRead != 'E')
     {
    sb.sem_num = 0;
    sb.sem_op = -1;  /* set to allocate resource */
    sb.sem_flg = SEM_UNDO;

    if ((key = ftok(".", u_char)) == -1)
    {
      perror("ftok");
      exit(1);
    }

    /* grab the semaphore set created by initsem: */
    if ((semid = initsem(key, 1)) == -1) {
      perror("initsem");
      exit(1);
    }

    printf("Press Enter to lock: ");
    getchar();
    printf("Trying to lock...\n");

    if (semop(semid, &sb, 1) == -1) {
      perror("semop");
      exit(1);
    }

    printf("Locked.\n");
    printf("Press Enter to unlock: ");
    getchar();

    sb.sem_op = 1; /* free resource */
    if (semop(semid, &sb, 1) == -1) {
      perror("semop");
      exit(1);
    }

    printf("Unlocked\n");
```

```
    charForRead = getchar();
  }
     return 0;
}
```

8. Составьте программу, позволяющую мониторить количество процессов (типа semdemo), находящихся в состоянии ожидания освобождения ресурса (Trying to lock...) в каждый момент времени. Программа строится на основе вызова semctl() с соответствующими параметрами и запускается на отдельном терминале.




```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <unistd.h>

int main(void)
{
     key_t key;
     int semid;

  while (1)
  {
    if ((key = ftok(".", 'J')) == -1)
```

```
    {
      perror("ftok");
      exit(1);
    }

    if ((semid = semget(key, 1, 0)) == -1)
    {
      perror("semget");
      exit(1);
    }

    int semNum = semctl(semid, 0, GETNCNT);
    printf("%i", semNum);
    sleep(1);
    printf("\n");
  }
     return 0;
}
```

Лабораторная работа №8
«ОБМЕН ЧЕРЕЗ ОЧЕРЕДИ СООБЩЕНИЙ»

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

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab8$ ls
gener_mq.cpp  receiver.cpp  sender.cpp
```

2. Скомпилируйте и выполните программу gener_mq.cpp , создающую несколько очередей сообщений. После завершения программы выполните команду ipcs и поясните отличие результата от того, что был при вызове подобной команды из программы.

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab8$ ./gener_mq

------ Очереди сообщений --------
ключ   msqid      владелец права исп. байты сообщения
0x4127de9a 0          pavel      660        0            0
0x4227de9a 1          pavel      660        0            0
0x4327de9a 2          pavel      660        0            0
0x4427de9a 3          pavel      660        0            0
0x4527de9a 4          pavel      660        0            0

------ Сегменты совм. исп. памяти --------
ключ   shmid      владелец права байты nattch     состояние
0x511b0896 2          pavel      600        16         0
0xca270022 3          pavel      600        65536      0

------ Массивы семафоров --------
ключ   semid      владелец права nsems
0xcc270022 2          pavel      600        1
0xcb270022 3          pavel      600        1

pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab8$ ipcs

------ Очереди сообщений --------
ключ   msqid      владелец права исп. байты сообщения

------ Сегменты совм. исп. памяти --------
ключ   shmid      владелец права байты nattch     состояние
0x511b0896 2          pavel      600        16         0
0xca270022 3          pavel      600        65536      0

------ Массивы семафоров --------
ключ   semid      владелец права nsems
0xcc270022 2          pavel      600        1
0xcb270022 3          pavel      600        1
```

В ходе выполнения программы было создано 5 очередей сообщений, они высвечиваються при выполнение программы. При завершение программа удаляет все 5 очередей, поэтому при использование команды ipcs мы видим отсутствие очередей

3. Скомпилируйте программы sender.cpp и receiver.cpp , задав соответствующим исполняемым файлам разные имена (g++ << имя .cpp

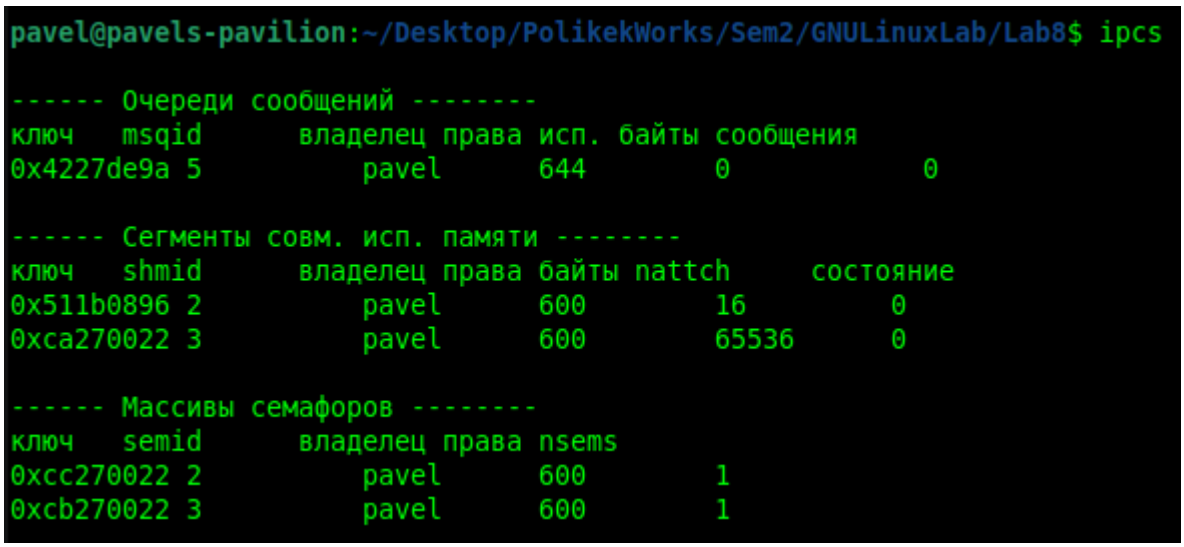
файла > -o < имя .out файла >). Запустите процессы на разных терминалах и передайте текстовые сообщения от процесса sender процессу receiver. Проанализируйте, что происходит с ресурсом Message Queue после завершения каждого из процессов (командой ipcs). При этом выполните различные виды завершения отправкой сигналов SIGQUIT и SIGINT (нажатием Ctrl-C).

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab8$ ipcs

------ Очереди сообщений --------
ключ   msqid      владелец права исп. байты сообщения

------ Сегменты совм. исп. памяти --------
ключ   shmid      владелец права байты nattch     состояние
0x511b0896 2          pavel      600        16         0
0xca270022 3          pavel      600        65536      0

------ Массивы семафоров --------
ключ   semid      владелец права nsems
0xcc270022 2          pavel      600        1
0xcb270022 3          pavel      600        1
```

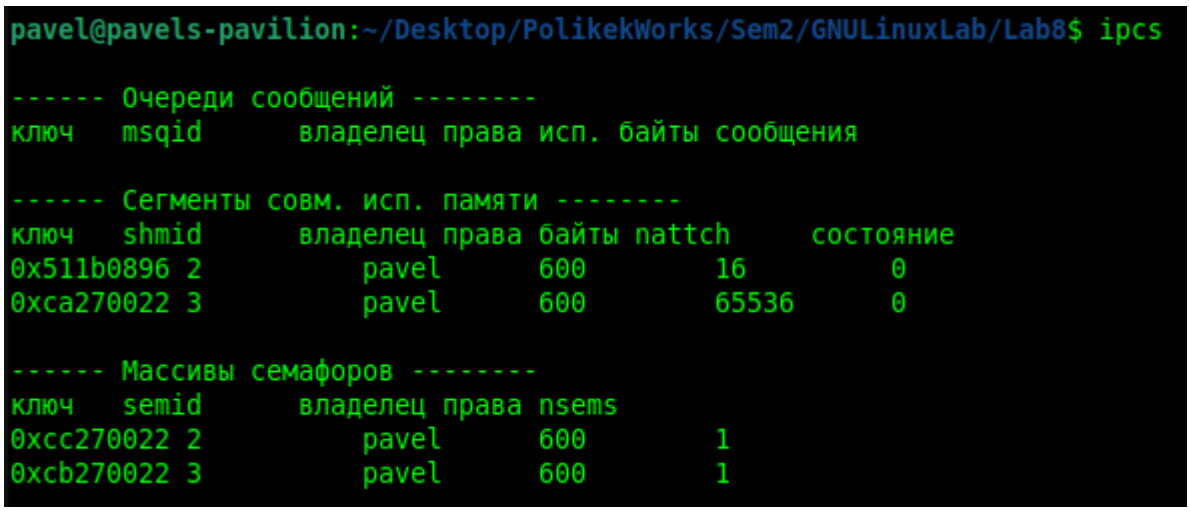
ipcs до запуска процессов

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab8$ ipcs

------ Очереди сообщений --------
ключ   msqid      владелец права исп. байты сообщения
0x4227de9a 5          pavel      644        0            0

------ Сегменты совм. исп. памяти --------
ключ   shmid      владелец права байты nattch     состояние
0x511b0896 2          pavel      600        16         0
0xca270022 3          pavel      600        65536      0

------ Массивы семафоров --------
ключ   semid      владелец права nsems
0xcc270022 2          pavel      600        1
0xcb270022 3          pavel      600        1
```

ipcs после запуска процессов



ipcs после завершения процессов (Ctrl + c)



ipcs после завершения процессов (Ctrl + d)

При завершение программы с помощью SIGINT (Ctrl + c) очередь остаётся существовать, при завершение процесса с помощью Ctrl + D очередь удаляется

4. Ответьте на вопрос: что происходит, если процесс receiver запускается уже после того, как процесс sender отправил в очередь одно или множество сообщений?

В случае, если receiver запускается после отправки сообщений. То он последовательно выводит эти сообщения, от самого старого к самому новому

5. Запустите несколько процессов receiver на различных терминалах и, отправляя сообщения процессом sender , проанализируйте ситуацию.

При запуске нескольких процессов receiver, сообщения отправляются поочерёдно:

первое сообщение – первый процесс

второе сообщение – второй процесс

третье сообщение – третий процесс

четвёртое сообщение – первый процесс

...

P.s в примере было запущено 3 процесса receiver

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab8$ ./sender
Enter lines of text, ^D to quit:
Some text
Some text2
Some text3
1
2
3
```

sender

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab8$ ./receiver
spock: ready to receive messages, captain.
spock: "Some text"
spock: "1"
```

receiver 1

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab8$ ./receiver
spock: ready to receive messages, captain.
spock: "Some text2"
spock: "2"
```

receiver 2

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab8$ ./receiver
spock: ready to receive messages, captain.
spock: "Some text3"
spock: "3"
```

receiver 3

6. Модифицируйте программы sender.cpp и receiver.cpp так, чтобы организовать отправку сообщений двух типов через одну и ту же очередь для двух различных процессов получателей. Для этого необходимо управлять параметром в поле mtype структуры my_msgbuf на передающей стороне и параметром msgtyp в системном вызове msgrcv() на приемной стороне.

sender2.cpp :

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
```

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

struct my_msgbuf
{
     long mtype;
     char mtext[200];
};

int main(void)
{
     struct my_msgbuf buf;
     int msqid;
     key_t key;

     if ((key = ftok(".", 'B')) == -1)
  {
          perror("ftok");
          exit(1);
     }

     if ((msqid = msgget(key, 0644 | IPC_CREAT)) == -1)
  {
          perror("msgget");
          exit(1);
     }

     printf("Enter lines of text, ^D to quit:\n");

     buf.mtype = 1;

     while(fgets(buf.mtext, sizeof buf.mtext, stdin) != NULL)
  {
          int len = strlen(buf.mtext);

          /* ditch newline at end, if it exists */
    if (buf.mtext[0] == '1')
    {
      buf.mtype = 1;
    }
    else if (buf.mtext[0] == '2')
    {
      buf.mtype = 2;
    }

          if (buf.mtext[len - 1] == '\n') buf.mtext[len - 1] =
'\0';

          if (msgsnd(msqid, &buf, len + 1, 0) == -1) /* +1 for
'\0' */
```

```
                perror("msgsnd");
     }

     if (msgctl(msqid, IPC_RMID, NULL) == -1)
  {
          perror("msgctl");
          exit(1);
     }

     return 0;
}
```

receiver2.cpp :

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <iostream>

struct my_msgbuf
{
     long mtype;
     char mtext[200];
};

int main(void)
{
     struct my_msgbuf buf;
     int msqid;
     key_t key;

     if ((key = ftok(".", 'B')) == -1)
  {  /* same key as sender.cpp */
          perror("ftok");
          exit(1);
     }

     if ((msqid = msgget(key, 0644)) == -1)
  { /* connect to the queue */
          perror("msgget");
          exit(1);
     }

     printf("write num of receiver2 procces:\n");

  char receiverNum = '1';
  std::cin >> receiverNum;

     for(;;)
```

```
  { /* Spock never quits! */
          if   (msgrcv(msqid,   &buf,   sizeof(buf.mtext),   (2   -
(receiverNum == '1')), 0) == -1)
    {
                perror("msgrcv");
                exit(1);
          }

          printf("spock: \"%s\"\n", buf.mtext);
     }

     return 0;
}
```

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab8$ ./sender2
Enter lines of text, ^D to quit:
1
Fisrt
Also first
First too
2
Second
Also Second
Second too
1
Again fisrt
2
Again second
```

sender2

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab8$ ./receiver2
write num of receiver2 procces:
1
spock: "1"
spock: "Fisrt"
spock: "Also first"
spock: "First too"
spock: "1"
spock: "Again fisrt"
```

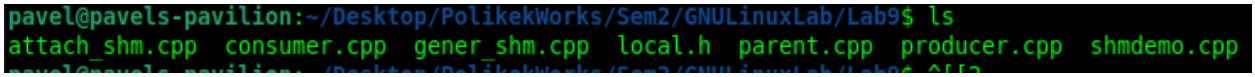
receiver2.1

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab8$ ./receiver2
write num of receiver2 procces:
2
spock: "2"
spock: "Second"
spock: "Also Second"
spock: "Second too"
spock: "2"
spock: "Again second"
```

receiver2.2

Лабораторная работа №9
«РАБОТА С РАЗДЕЛЯЕМОЙ ПАМЯТЬЮ»

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



2. Скомпилируйте и выполните программу gener_shm.cpp демонстрирующую создание сегментов разделяемой памяти. Запустите программу несколько раз и после каждого ее завершения выполните команду ipcs -m . Поясните зависимость процедуры создания сегментов разделяемой памяти от используемых в вызове shmget() флагов.

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ./gener_shm
First memory identifire is 7
Second shared memory identifire is 8
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ipcs -m

------ Сегменты совм. исп. памяти --------
ключ   shmid      владелец права байты nattch     состояние
0x511b1906 2          pavel      600        16         0
0xca260022 3          pavel      600        65536      0
0x0000000f 7          pavel      644        1000       0
0x00000000 8          pavel      644        20         0

pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ./gener_shm
First memory identifire is 7
Second shared memory identifire is 9
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ipcs -m

------ Сегменты совм. исп. памяти --------
ключ   shmid      владелец права байты nattch     состояние
0x511b1906 2          pavel      600        16         0
0xca260022 3          pavel      600        65536      0
0x0000000f 7          pavel      644        1000       0
0x00000000 8          pavel      644        20         0
0x00000000 9          pavel      644        20         0

pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ./gener_shm
First memory identifire is 7
Second shared memory identifire is 10
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ipcs -m

------ Сегменты совм. исп. памяти --------
ключ   shmid      владелец права байты nattch     состояние
0x511b1906 2          pavel      600        16         0
0xca260022 3          pavel      600        65536      0
0x0000000f 7          pavel      644        1000       0
0x00000000 8          pavel      644        20         0
0x00000000 9          pavel      644        20         0
0x00000000 10         pavel      644        20         0

pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ./gener_shm
First memory identifire is 7
Second shared memory identifire is 11
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ipcs -m

------ Сегменты совм. исп. памяти --------
ключ   shmid      владелец права байты nattch     состояние
0x511b1906 2          pavel      600        16         0
0xca260022 3          pavel      600        65536      0
0x0000000f 7          pavel      644        1000       0
0x00000000 8          pavel      644        20         0
0x00000000 9          pavel      644        20         0
0x00000000 10         pavel      644        20         0
0x00000000 11         pavel      644        20         0
```

Т.к в функции shmget() использует флаг IPC_CREAT при каждом запуске программы выделяется новый сегмент памяти

3. Удалите созданные на предыдущем шаге сегменты разделяемой памяти с помощью команды ipcrm с соответствующей опцией и значением id сегмента или ключа.

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ipcrm -m 11
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ipcrm -m 10
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ipcrm -m 9
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ipcrm -m 8
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ipcrm -m 8
ipcrm: неверный id (8)
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ipcrm -m 7
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ipcs -m

------ Сегменты совм. исп. памяти --------
ключ   shmid      владелец права байты nattch     состояние
0x511b1906 2          pavel      600        16         0
0xca260022 3          pavel      600        65536      0
```

4. Скомпилируйте shmdemo.cpp , осуществляющую операции записи в разделяемую память без разделения доступа к этому общему ресурсу. Символы, записываемые в общую память, передаются в качестве параметра командной строки при запуске процесса shmdemo . Запуск этого процесса без параметров приводит к выводу на консоль текущего содержимого сегмента общей памяти.

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ./shmdemo
segment contains: ""
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ./shmdemo Sometext
writing to segment: "Sometext"
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ./shmdemo
segment contains: "Sometext"
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ./shmdemo Newtext
writing to segment: "Newtext"
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ./shmdemo
segment contains: "Newtext"
```

5. Запустите несколько раз процессы типа shmdemo с различными значениями параметров и проиллюстрируйте возможности чтения и записи в сегмент общей памяти независимо исполняемыми процессами. Затем удалите сегмент памяти командой ipcrm .

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ./shmdemo FirstTerminal
writing to segment: "FirstTerminal"
```

First Terminal

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ./shmdemo SecondTerminal
writing to segment: "SecondTerminal"
```

Second Terminal

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ./shmdemo
segment contains: "FirstTerminal"
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ./shmdemo
segment contains: "SecondTerminal"
```

Third Terminal

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ipcs -m

------ Сегменты совм. исп. памяти --------
ключ   shmid      владелец права байты nattch     состояние
0x511b1906 2          pavel      600        16         0
0xca260022 3          pavel      600        65536      0
0x52268d02 12         pavel      644        1024       0

pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ipcrm -m 12
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ipcs -m

------ Сегменты совм. исп. памяти --------
ключ   shmid      владелец права байты nattch     состояние
0x511b1906 2          pavel      600        16         0
0xca260022 3          pavel      600        65536      0
```

6. Скомпилируйте и выполните программу attach_shm.cpp , иллюстрирующую передачу символьной информации между двумя процессами (родственными) через сегмент общей памяти с модификацией этой информации. Проанализируйте значения выводимой информации о границах сегментов в системной памяти. За счет чего после завершения данной программы сегмент общей памяти уже не присутствует в системе?

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab9$ ./attach_shm
Addresses in parent

shared mem: 0x7eff8ff4b000
 program text (etext): 0x564813a42471
 initialized data (edata): 0x564813a45010
 uninitialized data (end): 0x564813a45018

In parent before fork, memory is : ABCDEFGHIJKLMNOPQRSTUVWXYZ
In child after fork, memory is : ABCDEFGHIJKLMNOPQRSTUVWXYZ

In parent after fork, memory is  : abcdefghijklmnopqrstuvwxyz
Parent removing shared memory
```

После завершения программы сегмент общей памяти не присутствует в системе из-за 49 строки программы, а именно:
*shmctl(shmid, IPC_RMID, (struct shmid_ds *) 0);*

В данной строке функция *shmctl* **используется с флагом** *IPC_RMID***, что приводит к удалению системной структуры**

7. Составьте программу, создающую три разделяемых сегмента памяти размером 1023 байта каждый. Укажите в вызове shmat() параметр shmaddr =

0 при привязке сегментов. Разместит ли система сегменты в последовательных участках? Позволит ли система ссылку или изменение 1024-го байта любого из этих участков?

Сегменты размещены не в последовательных участках, система позволяет изменить 1024 байт участков

Лабораторная работа №10
«СОЗДАНИЕ СОЕДИНЕНИЙ НА СОКЕТАХ»

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

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab10$ ls
echo_client.cpp  echo_server.cpp  get_host.cpp  get_serv.cpp  local_c_i.h  sock_c_i_clt.cpp  sock_c_i_srv.cpp  sock_c_u_clt.cpp  sock_c_u_srv.cpp  socketpair.cpp  spair.c
```

2. Скомпилируйте и выполните программу socketpair.cpp , иллюстрирующую создание простейшего вида сокета и обмен данными двух родственных процессов. Проанализируйте вывод на консоль. Существует ли зависимость обмена от различных соотношений величин временных задержек (в вызовах sleep()) в процессе-родителе и в процессе-потомке?

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab10$ ./socketpair
p->c:0
c->p: 1
p->c:2
c->p: 3
p->c:4
c->p: 5
p->c:6
c->p: 7
p->c:8
c->p: 9
```

На консоль выводятся сообщения числа посылаемые родителем потомку и обратно. Обмен не зависит от задержек

3. Скомпилируйте программы echo_server.cpp и echo_client.cpp , задавая им при компиляции разные имена (размещаем файлы в одном каталоге). Запустите программы сервера и клиента на разных терминалах. Введите символьную информацию в окне клиента и проанализируйте вывод. Какой разновидности принадлежат сокеты, используемые в данном примере клиент-серверного взаимодействия? С чем связано создание специального файла в текущем каталоге во время исполнения программ?

```
Trying to connect...
Connected.
> Some message
echo> Some message
```

echo_client

```
Waiting for a connection...
Connected.
Waiting for a connection...
Connected.
```

echo_server

Разновидность сокетов: *datagram sockets*
Создание временного сокета связанно с тем, что в программе echo_server существует основной сокет, который создаёт дочерние сокеты для связи с клиентами

4. Скомпилируйте с разными именами программы sock_c_i_srv .cpp и sock_c_i_clt.cpp (в них используется общий include файл local_c_i.h). Запустите программы сервера и клиента на разных терминалах. При запуске клиента указывайте в качестве параметра командной строки имя хоста localhost . Введите символьную информацию в окне клиента и поясните вывод. Какой разновидности принадлежат сокеты, используемые в данном примере клиент-серверного взаимодействия?

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab10$ ./sock_clt localhost
> Some message
SOME MESSAGE
> other message
OTHER MESSAGE
> CAPS MESSAGE
CAPS MESSAGE
```

catsock_c_i_clt

Разновидность сокетов: *stream sockets*

Сервер возвращает сообщение от клиента в верхнем регистре

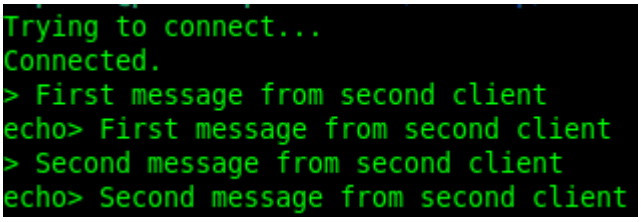
5. Модифицируйте программу echo_server.cpp так, чтобы при ответе на запросы клиента что-либо выводилось в окне сервера. Испытайте работу эхо-сервера при одновременной работе с несколькими клиентами.

```
Waiting for a connection...
Connected.
Message:
First message from first client
is send to client №4
Message:
Second message from first client
is send to client №4
Message:
Second message from first client
is send to client №4
Waiting for a connection...
Connected.
Message:
First message from second client
is send to client №4
Message:
Second message from second client
is send to client №4
Message:
Second message from second client
is send to client №4
Waiting for a connection...
```

echo_server2

```
Trying to connect...
Connected.
> First message from first client
echo> First message from first client
> Second message from first client
echo> Second message from first client
```

echo_client №1



echo_client №2

P.S. как мы видим программа echo_server2 не может одновременно принимать сообщение от двух клиентов, поэтому после завершения работы с первым клиентом его сокет закрывается и открывшийся сокет для работы со вторым клиентом приобретает тот же номер

echo_server2

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>

#define SOCK_PATH "echo_socket"

int main(void)
{
     int s, s2, t, len;
     struct sockaddr_un local, remote;
     char str[100];

     if ((s = socket(AF_UNIX, SOCK_STREAM, 0)) == -1)
  {
          perror("socket");
          exit(1);
     }

     local.sun_family = AF_UNIX;
     strcpy(local.sun_path, SOCK_PATH);
     unlink(local.sun_path);
     len = strlen(local.sun_path) + sizeof(local.sun_family);

     if (bind(s, (struct sockaddr *)&local, len) == -1)
  {
          perror("bind");
          exit(1);
     }
```

```
     if (listen(s, 5) == -1)
  {
          perror("listen");
          exit(1);
     }

     for(;;)
  {
          int done, n;
          printf("Waiting for a connection...\n");
          t = sizeof(remote);
          if  ((s2  =  accept(s,  (struct  sockaddr  *)&remote,
(socklen_t *)&t)) == -1)
    {
               perror("accept");
               exit(1);
          }

          printf("Connected.\n");

          done = 0;
          do
    {
               n = recv(s2, str, 100, 0);
      printf("Message:\n%sis send to client №%i\n", str, s2);
               if (n <= 0)
      {
                    if (n < 0) perror("recv");
                    done = 1;
               }

               if (!done)
                    if (send(s2, str, n, 0) < 0)
        {
                         perror("send");
                         done = 1;
                    }
          } while (!done);
          close(s2);
     }
     return 0;
}
```

Лабораторная работа №11
«ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ ПО СЕТИ»

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

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab11$ ls
Concur_code.txt  server_game.cpp
```

2. Скомпилируйте и запустите программу server_game.cpp , иллюстрирующую обмен данными с клиентскими приложениями по итеративной схеме.

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab11$ ./server_game
start to listen
```

3. Запустите другой терминал и проверьте с него наличие в системе созданного сервером сокета и то, что он находится в состоянии LISTEN. Для этого выполните команду netstat -a | grep 1066 . Проанализируйте вывод данной команды и объясните ее смысл.

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab11$ netstat -a | grep 1066
tcp        0      0 0.0.0.0:1066            0.0.0.0:*               LISTEN
```

I. В программе используется протокол TCP
II. Локальный адрес: 0.0.0.0:0
III. Внешний адрес: 0.0.0.0
IV. Состояние

4. Запустите в качестве клиентского процесса утилиту telnet с параметрами: telnet localhost 1066 . При организации коммуникации по сети на разных компьютерах вместо localhost при запуске клиента указывается IP-адрес компьютера, на котором был запущен сервер.

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab11$ telnet localhost 1066
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Glaying on host: @
 0:

 -----   12
```

5. Диалог с сервером заключается в угадывании слова. Оно вводится по буквам с клиентского терминала. При этом сервер вместо неугаданных букв выдает символы ”-” , а также считает число оставшихся неудачных попыток (всего их предусмотрено 12).

```
 -----   12
S
 -----   11
O
 -----   10
o
 -----   9
m
 -----   8
e
 --ee-   8
a
 --ee-   7
b
 --ee-   6
o
 --ee-   5
i
 --ee-   4
k
 --ee-   3
l
 --ee-   2
j
 --ee-   1
u
 green   0
Connection closed by foreign host.
```

6. Завершите серверное приложение с помощью сигнала kill, и затем определите командой netstat -a | grep 1066 , когда исчезает из системы соединение на сокетах. Во время сеанса обмена также примените команду netstat -a | grep 1066 , чтобы исследовать состояние соединения.

```
tcp        0      0 0.0.0.0:1066            0.0.0.0:*               LISTEN
tcp        0      0 localhost:44568         localhost:1066          ESTABLISHED
tcp        0      0 localhost:1066          localhost:44568         ESTABLISHED
```

netstat -a | grep 1066 во время сеанса обмена

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab11$ netstat -a | grep 1066
tcp        0      0 localhost:1066          localhost:44568         TIME_WAIT
```

netstat -a | grep 1066 после применения kill

7. Проделайте все заново, но запускайте не одно клиентское приложение (в виде telnet), а несколько экземпляров с разных терминалов, и попытайтесь работать с них одновременно. Проанализируйте, как сервер будет обслуживать запросы в этом случае.

```
pavel@pavels-pavilion:~/Desktop/PolikekWorks/Sem2/GNULinuxLab/Lab11$ netstat -a | grep 1066
tcp        1      0 0.0.0.0:1066            0.0.0.0:*               LISTEN
tcp        0      0 localhost:1066          localhost:40394         ESTABLISHED
tcp        0      0 localhost:40410         localhost:1066          ESTABLISHED
tcp        0      0 localhost:40394         localhost:1066          ESTABLISHED
tcp        0      0 localhost:1066          localhost:40410         ESTABLISHED
```

netstat -a | grep 1066 при запуске нескольких telnet

Сервер обслужит сначала первое клиентское приложение и только после его завершения перейдёт к следующему

8. Модифицируйте программу server_game.cpp так, чтобы запросы от каждого из клиентов могли обслуживаться конкурентно (путем запуска для каждого нового соединения собственного нового процесса на сервере. Возможно также улучшить качество самой игровой функции guess_word() сервера. Проанализируйте, как обслуживаются запросы в случае конкурентной схемы работы сервера.

server_game2

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>
#include <syslog.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <errno.h>

int maxlives = 12;

#define MAXLEN 80        /* max size of string */
#define SERVER_GAME_TCP_PORT  1066


void guess_word(int, int);

int main(void)
{
  int sock, fd, client_len;
  struct sockaddr_in server, client;

  sock = socket(AF_INET, SOCK_STREAM, 0);
  if (sock < 0)     {
    perror("creating stream socket");
    exit(1);
```

```
     }

  server.sin_family = AF_INET;
  server.sin_addr.s_addr = htonl(INADDR_ANY);
  server.sin_port   = htons(SERVER_GAME_TCP_PORT);

  if (bind(sock, (struct sockaddr *) &server, sizeof (server)) <
0) {
    perror("binding socket");
    exit(2);
     }

  listen(sock, 5);
  printf("start to listen\n");

  while (1)
  {
    client_len = sizeof(client);
       if ((fd = accept(sock, (struct sockaddr *) &client,
(socklen_t *)&client_len)) < 0)
    {
      perror("accepting connection");
      exit(3);
          }

    switch(fork())
    {
      case -1:
        perror("fork");
        break;

      case 0:
        printf("new guess_word start\n");
        close(sock);
        guess_word(fd, fd);
        close(fd);
        _exit(0);

      default:
        close(fd);
    }
  }
  close(sock);
}

/*----------------------------        guess_word()        function
----------------------------*/

void guess_word(int in, int out)
{
     char word_[MAXLEN];
```

```
     word_[0]= 'L';
     word_[1]= 'i';
     word_[2]= 'n';
     word_[3]= 'u';
     word_[4]= 'x';

     char    *whole_word,    part_word[MAXLEN],    guess[MAXLEN],
outbuf[MAXLEN + 20];
     int lives = maxlives;    /* Number of lives left */
     int game_state = 'I';    /* I => Incomplete, W => User Won,
L => User Lost */
     int i, good_guess, word_length;
     char hostname[MAXLEN];

     gethostbyname(hostname);
     sprintf(outbuf, "Playing on host: %s:\n\n", hostname);
     write (out, outbuf, strlen(outbuf));

               /* Pick a word */

     whole_word = word_;
     word_length = 5;
     syslog(LOG_USER|LOG_INFO,  "server_game  chose  word  %s",
whole_word);

     /* No letters are guessed initially */
     for (i = 0; i < word_length; i++)
          part_word[i] = '-';
     part_word[i] = '\0';

     sprintf(outbuf, " %s   %d\n", part_word, lives);
     write(out, outbuf, strlen(outbuf));

     while (game_state == 'I')
          /* Get guess letters from User */
  {
    while (read(in, guess, MAXLEN) < 0)
          {
      if (errno != EINTR)
        exit(4);
      printf("re-starting the read\n");
    }     /* Re-start read() if interrupted by signal */
    good_guess = 0;
    for (i = 0; i < word_length; i++)
    {
      if (guess[0] == whole_word[i])
      {
        good_guess = 1;
        part_word[i] = whole_word[i];
      }
    }
```

```
    if (! good_guess) lives--;
    if (strcmp(whole_word, part_word) == 0)
          game_state = 'W';   /* W => User Won  */
    else if (lives == 0)
    {
      game_state = 'L';  /* L => User Lost */
      strcpy(part_word, whole_word);  /* Show User the word */
    }
    sprintf (outbuf, " %s   %d\n", part_word, lives);
    write(out, outbuf, strlen(outbuf));
     }
}
```