Государственное учреждение образования
“БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ”

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №2

по курсу “Операционные системы”

Создание и управление процессов в UNIX-подобных ОС

Выполнили:
студенты группы 121702
Витковская С.И.
Шершень К.А.

Проверил:
Цирук В.А.

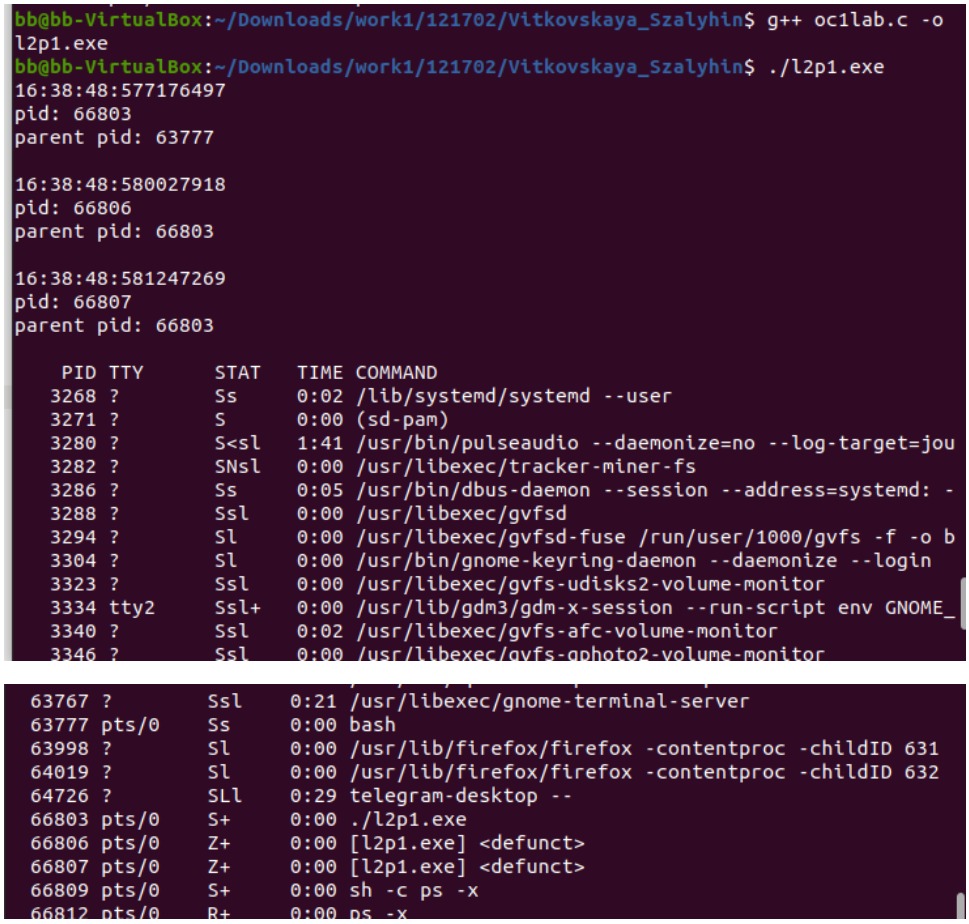
Минск, 2022

Цель: научиться создавать процессы и потоки, а также управлять ими.

Задание

1. Изучить теоретическую часть лабораторной работы.
2. Написать программу, создающую два дочерних процесса с использованием двух вызовов fork(). Родительский и два дочерних процесса должны выводить на экран свой pid и pid родительского процесса и текущее время в формате: часы : минуты : секунды : миллисекунды. Используя вызов system(), выполнить команду ps -x в родительском процессе. Найти свои процессы в списке запущенных процессов.

```c
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

int main()
{
    pid_t pid1,pid2;
    system("date +\"\%T:\%N\"");
    printf("pid: %d \n", getpid());
    printf("parent pid: %d \n\n", getppid());

    if ((pid1 = fork()) == 0)
    {
        system("date +\"\%T:\%N\"");
        printf("pid: %d \n", getpid());
        printf("parent pid: %d \n\n", getppid());
    }

    if (pid1 >0 && (pid2 =fork()) ==0)
    {
        system("date +\"\%T:\%N\"");
        printf("pid: %d \n", getpid());
        printf("parent pid: %d \n\n", getppid());

    }

    if (pid1 != 0 && pid2 != 0){ system ("ps -x");}

    return 0;
}
```

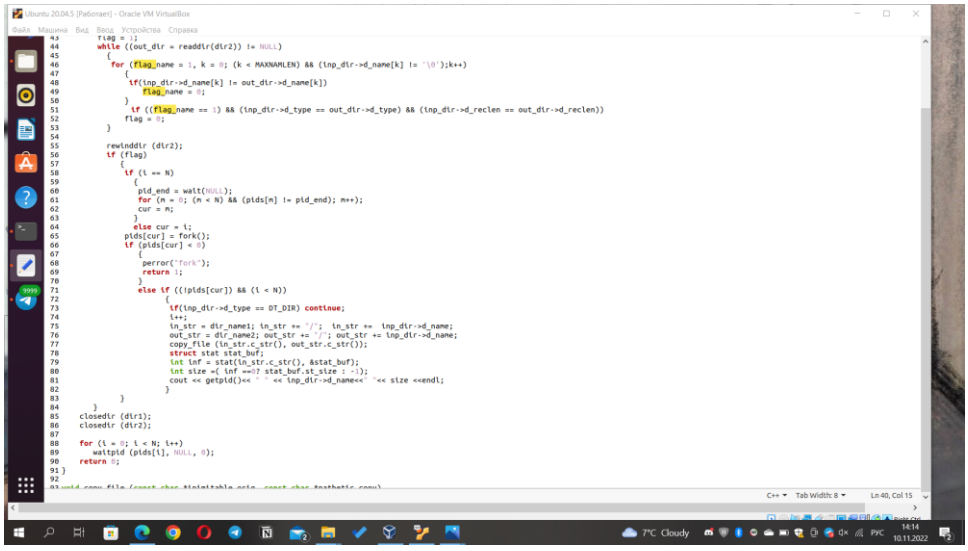


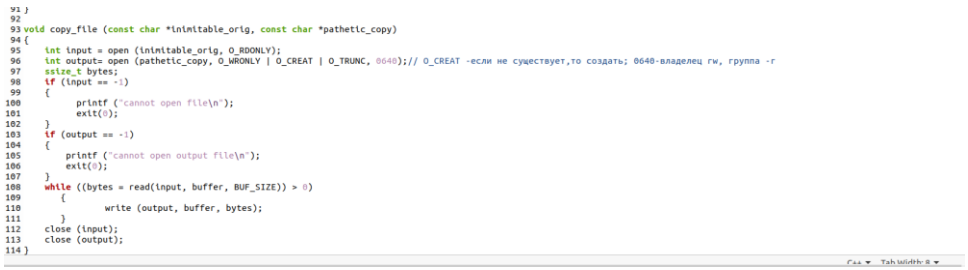
3. Выполнить индивидуальное задания.

Написать программу поиска одинаковых по их содержимому файлов в двух каталогов, например, Dir1 и Dir2. Пользователь задаёт имена Dir1 и Dir2. В результате работы программы файлы, имеющиеся в Dir1, сравниваются с файлами в Dir2 по их содержимому. Процедуры сравнения должны запускаться в отдельном процессе для каждой пары сравниваемых файлов. Каждый процесс выводит на экран свой pid, имя файла, общее число просмотренных байт и результаты сравнения. Число одновременно работающих процессов не должно превышать N (вводится пользователем).

l2_it.cpp *main.cpp

```cpp
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <dirent.h>
#include <unistd.h>
#include <fstream>
#include <sys/wait.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <iostream>
#define BUF_SIZE  4096
char buffer [BUF_SIZE];
using namespace std;

void copy_file (const char *inimitable_orig, const char *pathetic_copy);

int main (void)
{
    DIR *dir1, *dir2;
    struct dirent *inp_dir, *out_dir;         //структура с информацией о каталоге
    struct stat *buf;                         //структура с информацией о файле
    int flag, flag_name, k, m, cur, N;
    string in_str, out_str;
    pid_t pids[1024], pid_end;
    int file_o;
    char dir_name1[80], dir_name2[80];

    printf("Dir1: \n");               scanf("%s", dir_name1);
    printf("Dir2: \n");               scanf("%s", dir_name2);
    printf("Число процессов N: \n");  scanf("%d", &N); // Число процессов по адресу NPROC

    dir1 = opendir (dir_name1);
    dir2 = opendir (dir_name2);
    if ((!dir1)||(!dir2))
      {
        printf("cannot open a directory");
        return 1;
      }

    int i = 0;
    while ((inp_dir = readdir(dir1)) != NULL)
       {
        flag = 1;
        while ((out_dir = readdir(dir2)) != NULL)
          {
            for (flag_name = 1, k = 0; (k < MAXNAMLEN) && (inp_dir->d_name[k] != '\0');k++)
```





```cpp
}
void copy_file (const char *inimitable_orig, const char *pathetic_copy)
{
    int input = open (inimitable_orig, O_RDONLY);
    int output= open (pathetic_copy, O_WRONLY | O_CREAT | O_TRUNC, 0640);// O_CREAT -если не существует,то создать; 0640-владелец rw, группа -r
    ssize_t bytes;
    if (input == -1)
    {
        printf ("cannot open file\n");
        exit(0);
    }
    if (output == -1)
    {
        printf ("cannot open output file\n");
        exit(0);
    }
    while ((bytes = read(input, buffer, BUF_SIZE)) > 0)
        {
            write (output, buffer, bytes);
        }
    close (input);
    close (output);
}
```

4. Написать программу, которая будет реализовывать следующие функции:

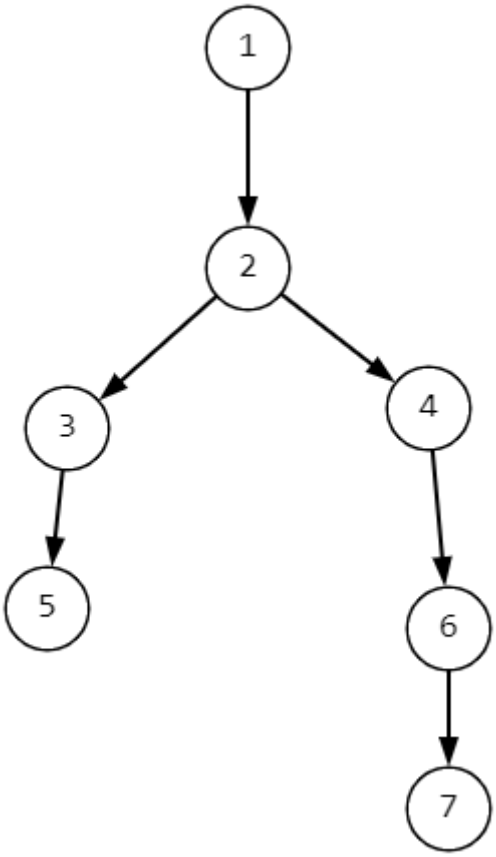
- сразу после запуска получает и сообщает свой ID и ID родительского процесса;
- перед каждым выводом сообщения об ID процесса и родительского процесса эта информация получается заново;
- порождает процессы, формируя генеалогическое дерево согласно варианту, сообщая, что "процесс с ID таким-то породил процесс с таким-то ID";
- перед завершением процесса сообщить, что "процесс с таким-то ID и таким- то ID родителя завершает работу";
- один из процессов должен вместо себя запустить программу, указанную в варианте задания.

На основании выходной информации программы предыдущего пункта изобразить генеалогическое дерево процессов (с указанием идентификаторов процессов).

4.1 вариант 2 - Витковская

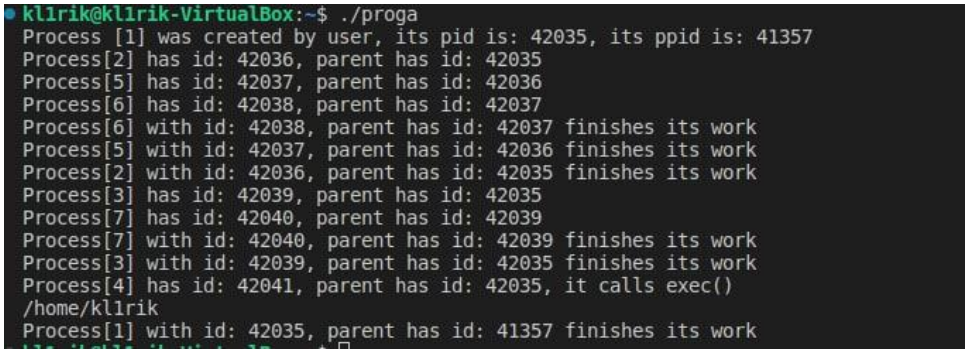
```c
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

int main()
{
    pid_t pid2,pid3, pid4, pid5, pid6,pid7;
    printf("process 1, its pid is: %d \nparent has a pid: %d \n\n", getpid(), getppid());
    if((pid2 = fork())==0)
    {
        printf("process 2, its pid is: %d \nparent has a pid: %d \n\n", getpid(), getppid());


        if ((pid3=fork()) == 0)
        {
                printf("process 3, its pid is: %d \nparent has a pid: %d \n\n", getpid(), getppid());
                if ((pid5 = fork()) == 0)
                {
                        printf("process 5, its pid is: %d \nparent has a pid: %d \n\n", getpid(), getppid());
                        printf("process 5 (pid : %d, parent pid : %d) finishes its work\n\n",getpid(),getppid());
                        exit(0);
                }
                wait(0);
                printf("process 3 (pid : %d, parent pid : %d) finishes its work\n\n",getpid(),getppid());
                exit(0);
        }
        if((pid4=fork())==0)
        {
                printf("process 4, its pid is: %d \nparent has a pid: %d \n\n", getpid(), getppid());
                if ((pid6 = fork()) == 0)
                {
                        printf("process 6, its pid is: %d \nparent has a pid: %d \n\n", getpid(), getppid());
                        if ((pid7=fork())==0)
                        {
                                printf("process 7, its pid is: %d \nparent has a pid: %d \n\n", getpid(), getppid());
                                printf("process 7 (pid : %d, parent pid : %d) finishes its work\n\n",getpid(),getppid());
                                exit(0);
                        }
                        wait(0);
                        printf("process 6 (pid : %d, parent pid : %d) finishes its work\n\n",getpid(),getppid());
                        exit(0);


                }
                wait(0);
                printf("process 4 (pid : %d, parent pid : %d) finishes its work\n\n",getpid(),getppid());
                exit(0);
        }
        wait(0);
        printf("process 2 (pid : %d, parent pid : %d) finishes its work\n\n",getpid(),getppid());
        execl("/bin/ps","ps", "-f",NULL);
        exit(0);
    }
    wait(0);
    printf("process 1 (pid : %d, parent pid : %d) finishes its work\n\n",getpid(),getppid());
    exit(0);
      return 0;
}
```

```
./l2_it.exe
process 1, its pid is: 70191
parent has a pid: 70014

process 2, its pid is: 70192
parent has a pid: 70191

process 3, its pid is: 70193
parent has a pid: 70192

process 4, its pid is: 70194
parent has a pid: 70192

process 5, its pid is: 70195
parent has a pid: 70193

process 5 (pid : 70195, parent pid : 70193) finishes its work

process 3 (pid : 70193, parent pid : 70192) finishes its work

process 2 (pid : 70192, parent pid : 70191) finishes its work

process 6, its pid is: 70196
parent has a pid: 70194

process 7, its pid is: 70197
parent has a pid: 70196

process 7 (pid : 70197, parent pid : 70196) finishes its work

process 6 (pid : 70196, parent pid : 70194) finishes its work

process 4 (pid : 70194, parent pid : 70192) finishes its work

UID          PID    PPID  C STIME TTY          TIME CMD
bb         70014   70004  0 03:45 pts/0    00:00:00 bash
bb         70191   70014  0 04:11 pts/0    00:00:00 ./l2_it.exe
bb         70192   70191  0 04:11 pts/0    00:00:00 ps -f
bb         70194   70192  0 04:11 pts/0    00:00:00 [l2_it.exe] <defunct>
process 1 (pid : 70191, parent pid : 70014) finishes its work
```



4.2 вариант 3- Шершень

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    pid_t pid;
    printf("Process [1] was created by user, its pid is: %d, its ppid is: %d\n", getpid(), getppid());
    if((pid = fork()) == 0){
        printf("Process[2] has id: %d, parent has id: %d\n", getpid(), getppid());
        if((pid = fork()) == 0){
            printf("Process[5] has id: %d, parent has id: %d\n", getpid(), getppid());
            if((pid = fork()) == 0){
                printf("Process[6] has id: %d, parent has id: %d\n", getpid(), getppid());
                printf("Process[6] with id: %d, parent has id: %d finishes its work\n", getpid(), getppid());
                exit(0);
            }
            wait(0);
            printf("Process[5] with id: %d, parent has id: %d finishes its work\n", getpid(), getppid());
            exit(0);
        }
        wait(0);
        printf("Process[2] with id: %d, parent has id: %d finishes its work\n", getpid(), getppid());
        exit(0);
    }
    wait(0);
    if((pid = fork()) == 0){
        printf("Process[3] has id: %d, parent has id: %d\n", getpid(), getppid());
        if((pid = fork()) == 0){
            printf("Process[7] has id: %d, parent has id: %d\n", getpid(), getppid());
            printf("Process[7] with id: %d, parent has id: %d finishes its work\n", getpid(), getppid());
            exit(0);
        }
        wait(0);
        printf("Process[3] with id: %d, parent has id: %d finishes its work\n", getpid(), getppid());
        exit(0);
    }
    wait(0);
    if((pid = fork()) == 0){
```



Вывод: научились создавать процессы и потоки с помощью вызова fork(), а также управлять ими. Выполнили задание синхронизации двух каталогов и построили деревья процессов.