

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.

```
1 #include<sys/types.h>
2 #include<sys/wait.h>
3 #include<unistd.h>
4 #include<stdio.h>
5 #include<stdlib.h>
6 #include<time.h>
7
8 int main()
9 {
10     pid_t pid1, pid2;
11     system("date +\"\\%T:\\%N\\\"");
12     printf("pid: %d \\n", getpid());
13     printf("parent pid: %d \\n\\n", getppid());
14
15     if ((pid1 = fork()) == 0)
16     {
17         system("date +\"\\%T:\\%N\\\"");
18         printf("pid: %d \\n", getpid());
19         printf("parent pid: %d \\n\\n", getppid());
20     }
21
22     if (pid1 > 0 && (pid2 = fork()) == 0)
23     {
24         system("date +\"\\%T:\\%N\\\"");
25         printf("pid: %d \\n", getpid());
26         printf("parent pid: %d \\n\\n", getppid());
27     }
28
29     if (pid1 != 0 && pid2 != 0){ system ("ps -x");}
30
31     return 0;
32 }
33 }
```

```

bb@bb-VirtualBox:~/Downloads/work1/121702/Vitkovskaya_Szalyhin$ g++ oc1lab.c -o
l2p1.exe
bb@bb-VirtualBox:~/Downloads/work1/121702/Vitkovskaya_Szalyhin$ ./l2p1.exe
16:38:48:577176497
pid: 66803
parent pid: 63777

16:38:48:580027918
pid: 66806
parent pid: 66803

16:38:48:581247269
pid: 66807
parent pid: 66803

  PID TTY      STAT   TIME COMMAND
 3268 ?        Ss      0:02 /lib/systemd/systemd --user
 3271 ?        S        0:00 (sd-pam)
 3280 ?        S<sl    1:41 /usr/bin/pulseaudio --daemonize=no --log-target=jou
 3282 ?        SNsl    0:00 /usr/libexec/tracker-miner-fs
 3286 ?        Ss      0:05 /usr/bin/dbus-daemon --session --address=systemd: -
 3288 ?        Ssl     0:00 /usr/libexec/gvfsd
 3294 ?        Sl      0:00 /usr/libexec/gvfsd-fuse /run/user/1000/gvfs -f -o b
 3304 ?        Sl      0:00 /usr/bin/gnome-keyring-daemon --daemonize --login
 3323 ?        Ssl     0:00 /usr/libexec/gvfs-udisks2-volume-monitor
 3334 tty2     Ssl+    0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_
 3340 ?        Ssl     0:02 /usr/libexec/gvfs-afc-volume-monitor
 3346 ?        Ssl     0:00 /usr/libexec/gvfs-gphoto2-volume-monitor

 63767 ?        Ssl     0:21 /usr/libexec/gnome-terminal-server
 63777 pts/0      Ss      0:00 bash
 63998 ?        Sl      0:00 /usr/lib/firefox/firefox -contentproc -childID 631
 64019 ?        Sl      0:00 /usr/lib/firefox/firefox -contentproc -childID 632
 64726 ?        Sll     0:29 telegram-desktop --
 66803 pts/0      S+      0:00 ./l2p1.exe
 66806 pts/0      Z+      0:00 [l2p1.exe] <defunct>
 66807 pts/0      Z+      0:00 [l2p1.exe] <defunct>
 66809 pts/0      S+      0:00 sh -c ps -x
 66812 pts/0      R+      0:00 ps -x

```

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_lt.cpp      *main.cpp

1#include <stdlib.h>
2#include <stdio.h>
3#include <sys/types.h>
4#include <dirent.h>
5#include <unistd.h>
6#include <fstream>
7#include <sys/wait.h>
8#include <sys/stat.h>
9#include <fcntl.h>
10#include <iostream>
11#define BUF_SIZE 4096
12char buffer [BUF_SIZE];
13using namespace std;
14
15void copy_file (const char *ininitable_orig, const char *pathetic_copy);
16
17int main (void)
18{
19    DIR *dir1, *dir2;
20    struct dirent *inp_dir, *out_dir;           //структура с информацией о каталоге
21    struct stat *buf;                           //структура с информацией о файле
22    int flag, flag_name, k, m, cur, N;
23    string in_str, out_str;
24    pid_t pids[1024], pid_end;
25    int file_o;
26    char dir_name1[80], dir_name2[80];
27
28    printf("Dir1: \n");      scanf("%s", dir_name1);
29    printf("Dir2: \n");      scanf("%s", dir_name2);
30    printf("Число процессов N: \n");  scanf("%d", &N); // Число процессов по адресу NPROC
31
32    dir1 = opendir (dir_name1);
33    dir2 = opendir (dir_name2);
34    if ((!dir1)||(!dir2))
35    {
36        printf("cannot open a directory");
37        return 1;
38    }
39
40    int i = 0;
41    while ((inp_dir = readdir(dir1)) != NULL)
42    {
43        flag = 1;
44        while ((out_dir = readdir(dir2)) != NULL)
45        {
46            for (flag_name = 1, k = 0; (k < MAXNAMLEN) && (inp_dir->d_name[k] != '\0'); k++)

```

```

47            {
48                if (flag_name == 1, k = 0; (k < MAXNAMLEN) && (inp_dir->d_name[k] != '\0'); k++)
49                {
50                    if (inp_dir->d_name[k] != out_dir->d_name[k])
51                        flag_name = 0;
52                }
53                if ((flag_name == 1) && (inp_dir->d_type == out_dir->d_type) && (inp_dir->d_reclen == out_dir->d_reclen))
54                    flag = 0;
55            }
56            rewinddir (dir2);
57            if (flag)
58            {
59                if (i == N)
60                {
61                    pid_end = wait(NULL);
62                    for (n = 0; (n < N) && (pids[n] != pid_end); n++);
63                    cur = n;
64                }
65                else cur = i;
66                pids[cur] = fork();
67                if (pids[cur] < 0)
68                {
69                    perror("fork");
70                    return 1;
71                }
72                else if ((pids[cur] && (i < N))
73                {
74                    if (inp_dir->d_type == DT_DIR) continue;
75                    i++;
76                    in_str = dir_name1; in_str += "/"; in_str += inp_dir->d_name;
77                    out_str = dir_name2; out_str += "/"; out_str += inp_dir->d_name;
78                    copy_file (in_str.c_str(), out_str.c_str());
79                    struct stat stat_buf;
80                    int inf = stat(in_str.c_str(), &stat_buf);
81                    int size = (inf == 0 ? stat_buf.st_size : -1);
82                    out = getpuid(1) << " " << inp_dir->d_name << " " << size << endl;
83                }
84            }
85            closedir (dir1);
86            closedir (dir2);
87            for (i = 0; i < N; i++)
88                waitpid (pids[i], NULL, 0);
89            return 0;
90        }
91    }
92}

```

```

93void copy_file (const char *ininitable_orig, const char *pathetic_copy)
94{
95    int input = open (ininitable_orig, O_RDONLY);
96    int output = open (pathetic_copy, O_WRONLY | O_CREAT | O_TRUNC, 0640); // O_CREAT -если не существует,то создать; 0640-владелец rw, группа -r
97    ssize_t bytes;
98    if (input == -1)
99    {
100        printf ("cannot open file\n");
101        exit(0);
102    }
103    if (output == -1)
104    {
105        printf ("cannot open output file\n");
106        exit(0);
107    }
108    while ((bytes = read(input, buffer, BUF_SIZE)) > 0)
109    {
110        write (output, buffer, bytes);
111    }
112    close (input);
113    close (output);
114}

```

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

```

1 #include<sys/types.h>
2 #include<sys/wait.h>
3 #include<unistd.h>
4 #include<stdio.h>
5 #include<stdlib.h>
6 #include<string.h>
7
8 int main()
9 {
10     pid_t pid2,pid3, pid4, pid5, pid6,pid7;
11     printf("process 1, its pid is: %d \nparent has a pid: %d \n", getpid(), getppid());
12     if((pid2 = fork())==0)
13     {
14         printf("process 2, its pid is: %d \nparent has a pid: %d \n", getpid(), getppid());
15
16
17
18         if ((pid3=fork()) == 0)
19         {
20             printf("process 3, its pid is: %d \nparent has a pid: %d \n", getpid(), getppid());
21             if ((pid5 = fork()) == 0)
22             {
23                 printf("process 5, its pid is: %d \nparent has a pid: %d \n", getpid(), getppid());
24                 printf("process 5 (pid : %d, parent pid : %d) finishes its work\n\n",getpid(),getppid());
25                 exit(0);
26             }
27             wait(0);
28             printf("process 3 (pid : %d, parent pid : %d) finishes its work\n\n",getpid(),getppid());
29             exit(0);
30         }
31         if((pid4=fork())==0)
32         {
33             printf("process 4, its pid is: %d \nparent has a pid: %d \n", getpid(), getppid());
34             if ((pid6 = fork()) == 0)
35             {
36                 printf("process 6, its pid is: %d \nparent has a pid: %d \n", getpid(), getppid());
37                 if ((pid7=fork())==0)
38                 {
39                     printf("process 7, its pid is: %d \nparent has a pid: %d \n", getpid(), getppid());
40                     printf("process 7 (pid : %d, parent pid : %d) finishes its work\n\n",getpid(),getppid());
41                     exit(0);
42                 }
43                 wait(0);
44                 printf("process 6 (pid : %d, parent pid : %d) finishes its work\n\n",getpid(),getppid());
45                 exit(0);
46             }
47             wait(0);
48             printf("process 4 (pid : %d, parent pid : %d) finishes its work\n\n",getpid(),getppid());
49             exit(0);
50         }
51         wait(0);
52         printf("process 2 (pid : %d, parent pid : %d) finishes its work\n\n",getpid(),getppid());
53         execl("/bin/ps", "ps", "-f", NULL);
54         exit(0);
55     }
56     wait(0);
57     printf("process 1 (pid : %d, parent pid : %d) finishes its work\n\n",getpid(),getppid());
58     exit(0);
59     return 0;
60 }

```

```
./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

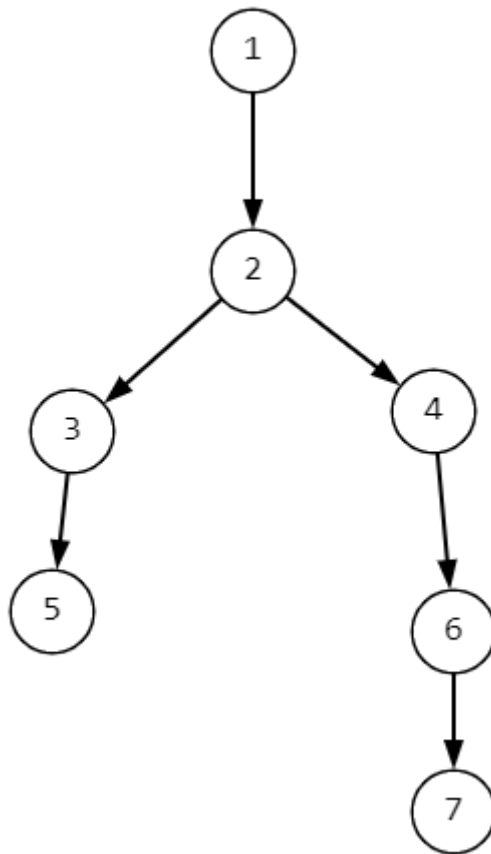
process 5 (pid : 70195, parent pid : 70193) finishes its work
process 3 (pid : 70193, parent pid : 70192) finishes its work
process 2 (pid : 70192, parent pid : 70191) finishes its work

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

```
#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){
```

```
klirik@klirik-VirtualBox:~$ ./proga
Process [1] was created by user, its pid is: 42035, its ppid is: 41357
Process[2] has id: 42036, parent has id: 42035
Process[5] has id: 42037, parent has id: 42036
Process[6] has id: 42038, parent has id: 42037
Process[6] with id: 42038, parent has id: 42037 finishes its work
Process[5] with id: 42037, parent has id: 42036 finishes its work
Process[2] with id: 42036, parent has id: 42035 finishes its work
Process[3] has id: 42039, parent has id: 42035
Process[7] has id: 42040, parent has id: 42039
Process[7] with id: 42040, parent has id: 42039 finishes its work
Process[3] with id: 42039, parent has id: 42035 finishes its work
Process[4] has id: 42041, parent has id: 42035, it calls exec()
/home/klirik
Process[1] with id: 42035, parent has id: 41357 finishes its work
```

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