

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Вычислительной техники

ОТЧЁТ

по лабораторной работе №3

по дисциплине «Организация процессов и программирования в среде Linux»

Тема: СОЗДАНИЕ И ИДЕНТИФИКАЦИЯ ПРОЦЕССОВ

Студент гр. 1335

Преподаватель

Максимов Ю Е

Разумовский Г.В.

Санкт-Петербург,

2024

Оглавление

1. Введение	3
1.1. Введение.....	3
1.2. Порядок выполнения работы	3
1.3. Содержание отчёта	3
2. Тексты программ с комментариями	4
2.1. main.cpp	4
2.2. executable.cpp	7
3. Распечатки файлов, содержащих параметры вызова программы и атрибуты процессов.....	8
3. Вывод	17
4. Список использованных источников.....	18

1. Введение

1.1. Введение

Тема работы: Создание и идентификация процессов.

Цель работы: Изучение и использование системных функций, обеспечивающих порождение и идентификацию процессов.

1.2. Порядок выполнения работы

1. Разработать программу, которая порождает 2 потомка. Первый потомок порождается с помощью `fork`, второй – с помощью `vfork` с последующей заменой на другую программу. Все 3 процесса должны вывести в один файл свои атрибуты с предварительным указанием имени процесса (например: Предок, Потомок1, Потомок2). Имя выходного файла задаётся при запуске программы. Порядок вывода атрибутов в файл должен определяться задержками процессов, которые задаются в качестве параметров программы и выводятся в начало файла.

2. Откомпилировать программу и запустить её 3 раза с различными сочетаниями задержек.

1.3. Содержание отчёта

Отчёт по лабораторной работе должен содержать:

1. Цель и задание.
2. Тексты программ с комментариями.
3. Распечатки файлов, содержащих параметры вызова программы и атрибуты процессов.

2. Тексты программ с комментариями

2.1. main.cpp

```
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h> // error
#include <unistd.h> // delay, process attributes
#include <time.h> // delay
#include <sys/types.h> // process attributes
#include <sys/wait.h> // pid_t types

using namespace std;

int main()
{
    int delay_parent; // delay seconds f/ parent
    int delay_child_1; // delay seconds f/ child_1
    int pid_1_status; // status pid_1
    int pid_2_status; // status pid_2
    char delay_child_2[10]; // delay seconds f/ child_2 (char f/ sending as argument)
    char file_name[80]; // output file name
    pid_t pid_1; // "child1" process, created w/ fork()
    pid_t pid_2; // "child2" process, created w/ vfork()
    ofstream file_stream; // file stream

    cout << "Filename: ";
    cin >> file_name;
    cout << "Parent, child_1 & child_2 delays (separate w/ space): ";
    cin >> delay_parent >> delay_child_1 >> delay_child_2;

    file_stream.open(file_name, file_stream.out | file_stream.app);
    file_stream << "Parent, child_1 & child_2 delays: " << delay_parent << " " << delay_child_1 << " " <<
delay_child_2 << "\n";
    file_stream.close();

    pid_1 = fork(); // creating "child1" process w/ "fork()"
```

```

// making process output w/ if-else statement because of more understandable pattern
if (pid_1 == -1) // if unsuccessful
{
    perror("ERROR w/ \"child_1\" process creating w/ \"fork()\" function"); // creating error
    // void exit(int exitCode)
    exit(1); // exit code '1' means error w/ "fork()" "child_1"
}
else if (pid_1 == 0) // if creation succeeded
{
    // void mdelay(unsigned long milliseconds);
    //mdelay(delay_child_1);
    // int sleep(unsigned sec);
    sleep(delay_child_1); // child_1 delay
    file_stream.open(file_name, file_stream.out | file_stream.app); // file open & write
    file_stream << "\n***CHILD_1***\n\n"
    << "Process ID (PID): " << getpid() << "\n"
    << "Parent process ID (PPID): " << getppid() << "\n"
    << "Session ID (SID): " << getsid(getpid()) << "\n"
    << "Process group ID (PGID): " << getpgid(getpid()) << "\n"
    << "[real] User ID (UID): " << getuid() << "\n"
    << "Effective user ID (EUID): " << geteuid() << "\n"
    << "[real] Group ID (GID): " << getgid() << "\n"
    << "Effective group ID (EGID): " << getegid() << "\n";
    file_stream.close();
    // void exit(int exitCode)
    exit(EXIT_SUCCESS); // successful exit
}
else
{
    pid_2 = vfork(); // creating "child2" process w/ "vfork()" (this one will change executable program)

    if (pid_2 == -1) // if unsuccessful
    {
        perror("ERROR w/ \"child_2\" process creating w/ \"vfork()\" function");
        // void exit(int exitCode)
        exit(2); // exit code '1' means error w/ "vfork()" "child_2"
    }
    else if (pid_2 == 0) // process "child2" can switch to executing another program stored in a file on disk
    {

```

```

        // int execl(const char *path, const char *arg, ...);
        execl("executable", file_name, delay_child_2, NULL); // execute another program and
sending arguments

        // void exit(int exitCode)
        exit(EXIT_SUCCESS); // successful exit
    }
    else
    {

        // void mdelay(unsigned long milliseconds);
        //mdelay(delay_parent);
        // int sleep(unsigned sec);
        sleep(delay_parent); // parent delay
        file_stream.open(file_name, file_stream.out | file_stream.app); // file open & write
        file_stream<<"\n***PARENT***\n\n"
        << "Process ID (PID): " << getpid() << "\n"
        << "Parent process ID (PPID): " << getppid() << "\n"
        << "Session ID (SID): " << getsid(getpid()) << "\n"
        << "Process group ID (PGID): " << getpgid(getpid()) << "\n"
        << "[real] User ID (UID): " << getuid() << "\n"
        << "Effective user ID (EUID): " << geteuid() << "\n"
        << "[real] Group ID (GID): " << getgid() << "\n"
        << "Effective group ID (EGID): " << getegid() << "\n";
        file_stream.close();
        waitpid(pid_1, &pid_1_status, 0); // child_1 waiting f/ correct ending
        waitpid(pid_2, &pid_2_status, 0); // child_2 waiting f/ correct ending
    }
}

return 0;
}

```

2.2. executable.cpp

```
#include <iostream>
#include <fstream>
#include <unistd.h> // delay, process attributes
#include <sys/types.h> // process attributes
#include <sys/wait.h> // pid_t types

using namespace std;

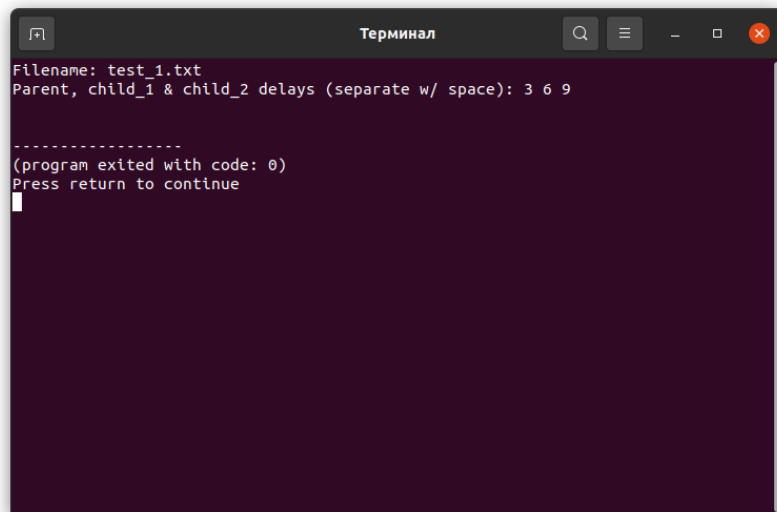
int main(int argc, char *argv[])
{
    ofstream file_stream; // file stream
    file_stream.open(argv[0], file_stream.out | file_stream.app); // file open & write
    sleep(atoi(argv[1])); // "child_2" delay

    file_stream << "\n***CHILD_2***\n\n"
    << "Process ID (PID): " << getpid() << "\n"
    << "Parent process ID (PPID): " << getppid() << "\n"
    << "Session ID (SID): " << getsid(getpid()) << "\n"
    << "Process group ID (PGID): " << getpgid(getpid()) << "\n"
    << "[real] User ID (UID): " << getuid() << "\n"
    << "Effective user ID (EUID): " << geteuid() << "\n"
    << "[real] Group ID (GID): " << getgid() << "\n"
    << "Effective group ID (EGID): " << getegid() << "\n";

    file_stream.close();
    return 0;
}
```

3. Распечатки файлов, содержащих параметры вызова программы и атрибуты процессов

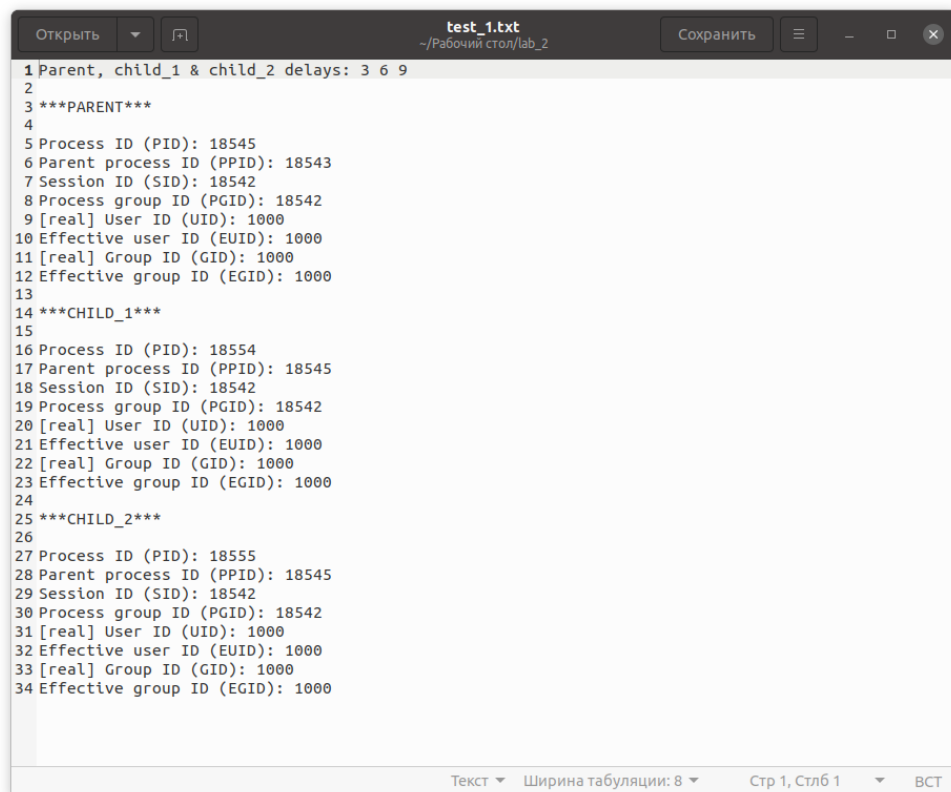
Проведём 3 теста на различные времена задержек процессов.



```
Терминал
Filename: test_1.txt
Parent, child_1 & child_2 delays (separate w/ space): 3 6 9

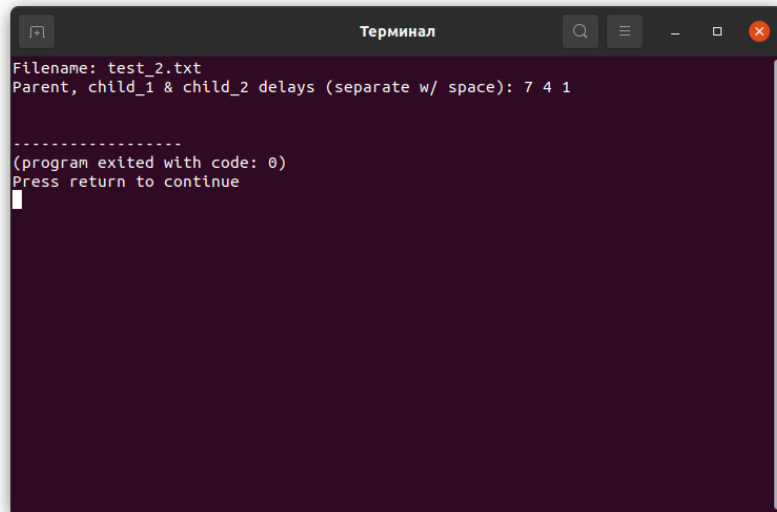
-----
(program exited with code: 0)
Press return to continue
```

Рисунок 1. Тест 1: Задержки для родительского процесса, дочернего процесса 1 и дочернего процесса 2 (меняющим исполняемую программу) равны 3, 6 и 9 секунд соответственно



```
1 Parent, child_1 & child_2 delays: 3 6 9
2
3 ***PARENT***
4
5 Process ID (PID): 18545
6 Parent process ID (PPID): 18543
7 Session ID (SID): 18542
8 Process group ID (PGID): 18542
9 [real] User ID (UID): 1000
10 Effective user ID (EUID): 1000
11 [real] Group ID (GID): 1000
12 Effective group ID (EGID): 1000
13
14 ***CHILD_1***
15
16 Process ID (PID): 18554
17 Parent process ID (PPID): 18545
18 Session ID (SID): 18542
19 Process group ID (PGID): 18542
20 [real] User ID (UID): 1000
21 Effective user ID (EUID): 1000
22 [real] Group ID (GID): 1000
23 Effective group ID (EGID): 1000
24
25 ***CHILD_2***
26
27 Process ID (PID): 18555
28 Parent process ID (PPID): 18545
29 Session ID (SID): 18542
30 Process group ID (PGID): 18542
31 [real] User ID (UID): 1000
32 Effective user ID (EUID): 1000
33 [real] Group ID (GID): 1000
34 Effective group ID (EGID): 1000
```

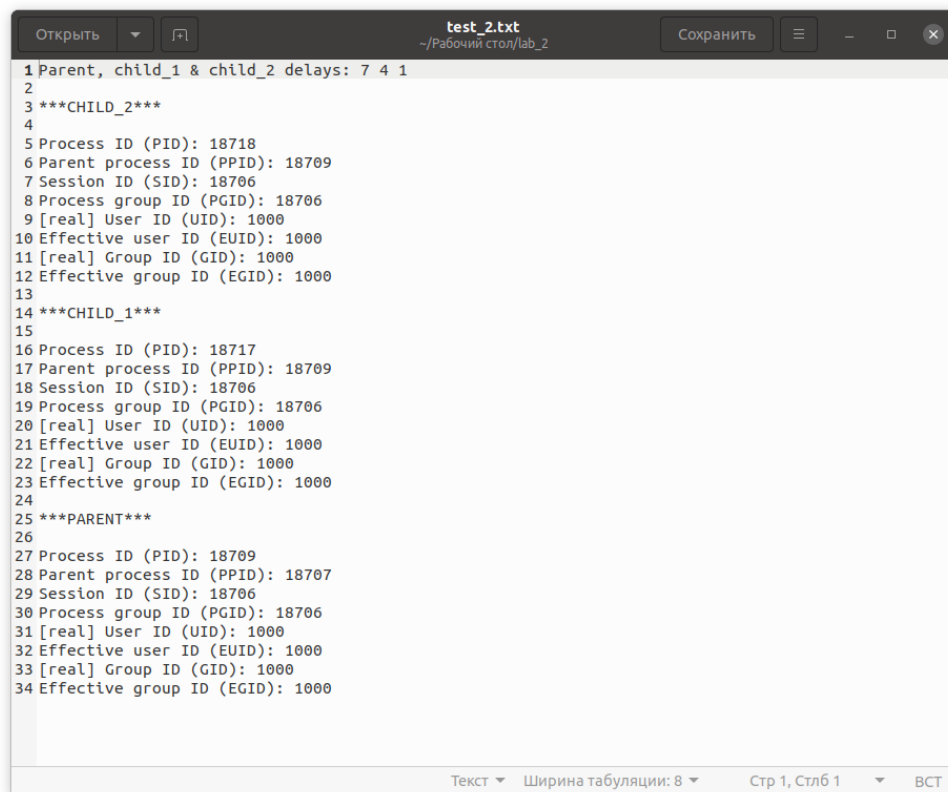
Рисунок 2. Тест 1: Порядок вывода процессов в файл – родительский процесс, дочерний процесс 1, дочерний процесс 2 (меняющий исполняемую программу)



```
Терминал
Filename: test_2.txt
Parent, child_1 & child_2 delays (separate w/ space): 7 4 1

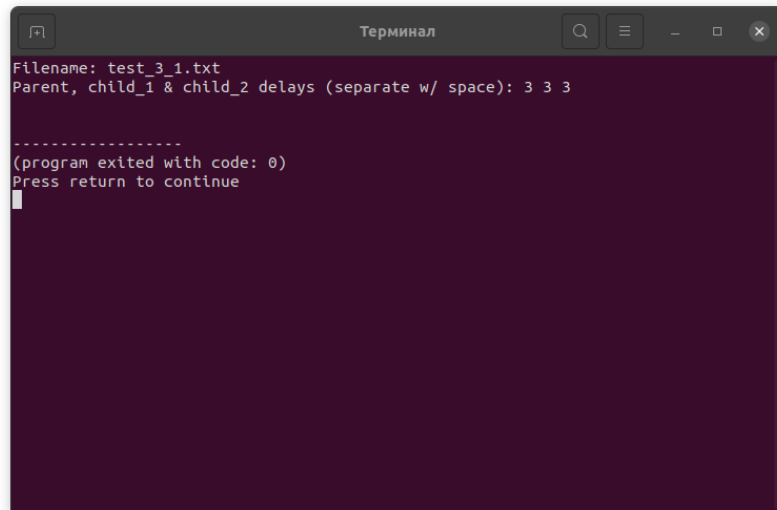
-----
(program exited with code: 0)
Press return to continue
```

Рисунок 3. Тест 2: Задержки для родительского процесса, дочернего процесса 1 и дочернего процесса 2 (меняющим исполняемую программу) равны 7, 4 и 1 секунд соответственно



```
1 Parent, child_1 & child_2 delays: 7 4 1
2
3 ***CHILD_2***
4
5 Process ID (PID): 18718
6 Parent process ID (PPID): 18709
7 Session ID (SID): 18706
8 Process group ID (PGID): 18706
9 [real] User ID (UID): 1000
10 Effective user ID (EUID): 1000
11 [real] Group ID (GID): 1000
12 Effective group ID (EGID): 1000
13
14 ***CHILD_1***
15
16 Process ID (PID): 18717
17 Parent process ID (PPID): 18709
18 Session ID (SID): 18706
19 Process group ID (PGID): 18706
20 [real] User ID (UID): 1000
21 Effective user ID (EUID): 1000
22 [real] Group ID (GID): 1000
23 Effective group ID (EGID): 1000
24
25 ***PARENT***
26
27 Process ID (PID): 18709
28 Parent process ID (PPID): 18707
29 Session ID (SID): 18706
30 Process group ID (PGID): 18706
31 [real] User ID (UID): 1000
32 Effective user ID (EUID): 1000
33 [real] Group ID (GID): 1000
34 Effective group ID (EGID): 1000
```

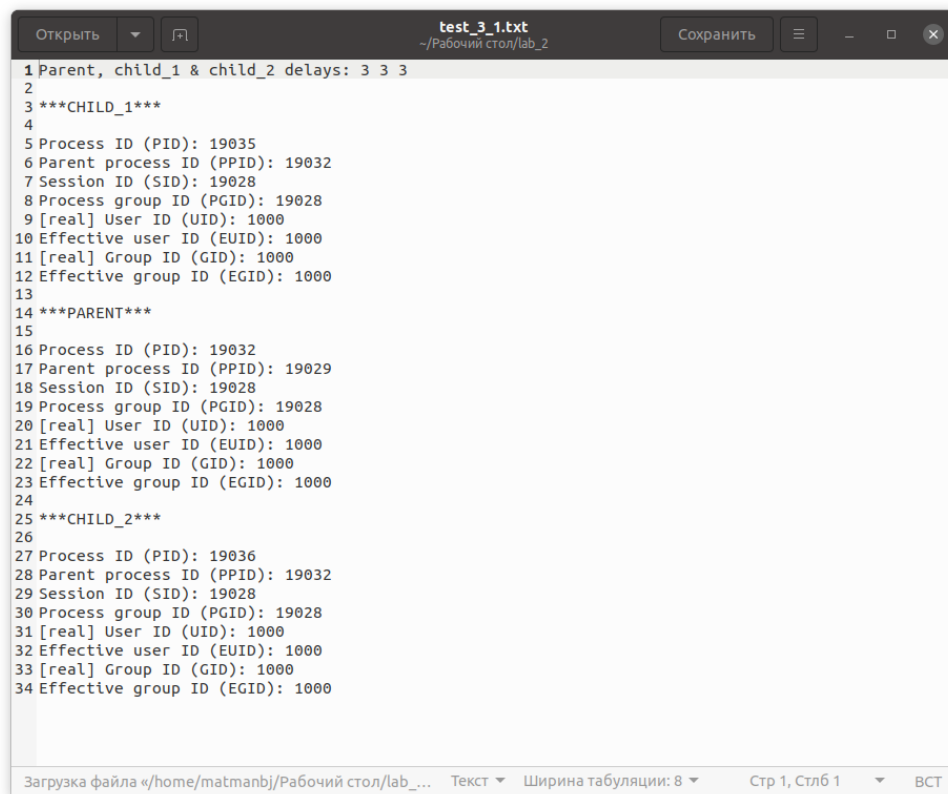
Рисунок 4. Тест 2: Порядок вывода процессов в файл – дочерний процесс 2 (меняющий исполняемую программу), дочерний процесс 1, родительский процесс



```
Терминал
Filename: test_3_1.txt
Parent, child_1 & child_2 delays (separate w/ space): 3 3 3

-----
(program exited with code: 0)
Press return to continue
█
```

Рисунок 5. Тест 3.1: Задержки для родительского процесса, дочернего процесса 1 и дочернего процесса 2 (меняющим исполняемую программу) равны 3, 3 и 3 секунд соответственно

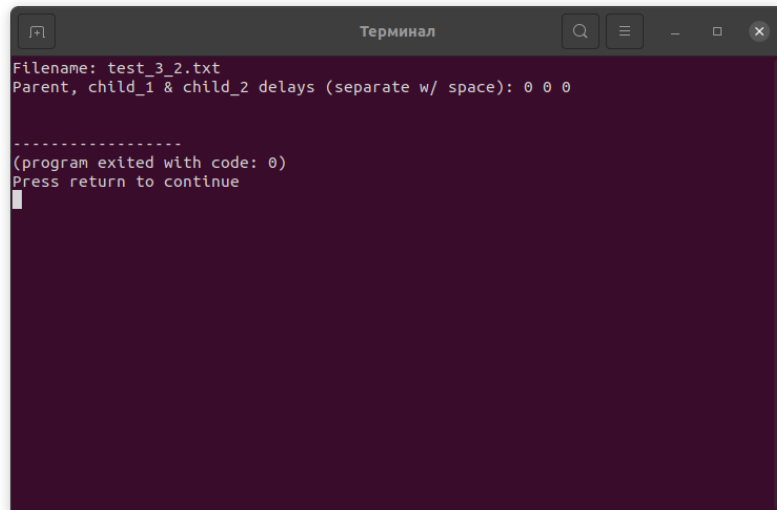


The screenshot shows a terminal window titled "test_3_1.txt" with a path "~/Рабочий стол/lab_2". The window contains a script that prints process details for a parent and two children. The output is as follows:

```
1 Parent, child_1 & child_2 delays: 3 3 3
2
3 ***CHILD_1***
4
5 Process ID (PID): 19035
6 Parent process ID (PPID): 19032
7 Session ID (SID): 19028
8 Process group ID (PGID): 19028
9 [real] User ID (UID): 1000
10 Effective user ID (EUID): 1000
11 [real] Group ID (GID): 1000
12 Effective group ID (EGID): 1000
13
14 ***PARENT***
15
16 Process ID (PID): 19032
17 Parent process ID (PPID): 19029
18 Session ID (SID): 19028
19 Process group ID (PGID): 19028
20 [real] User ID (UID): 1000
21 Effective user ID (EUID): 1000
22 [real] Group ID (GID): 1000
23 Effective group ID (EGID): 1000
24
25 ***CHILD_2***
26
27 Process ID (PID): 19036
28 Parent process ID (PPID): 19032
29 Session ID (SID): 19028
30 Process group ID (PGID): 19028
31 [real] User ID (UID): 1000
32 Effective user ID (EUID): 1000
33 [real] Group ID (GID): 1000
34 Effective group ID (EGID): 1000
```

The terminal window has a status bar at the bottom showing "Загрузка файла «/home/matmanbj/Рабочий стол/lab_... Текст ▾ Ширина табуляции: 8 ▾ Стр 1, Стлб 1 ▾ ВСТ".

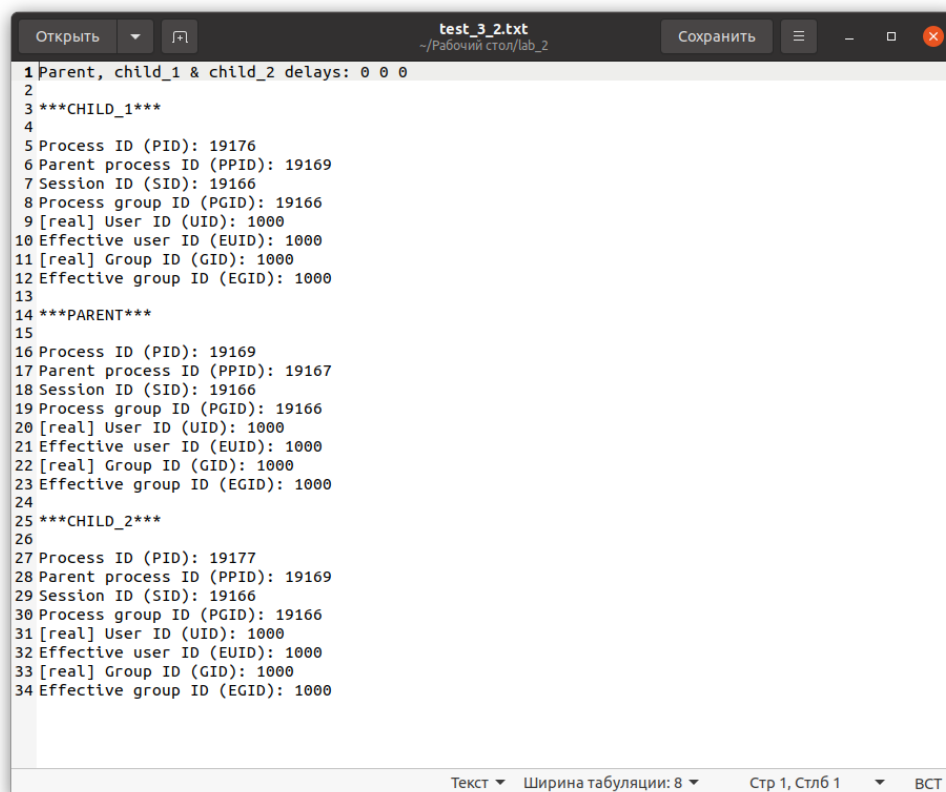
Рисунок 6. Тест 3.1: Порядок вывода процессов в файл – дочерний процесс 1, родительский процесс, дочерний процесс 2 (меняющий исполняемую программу)



```
Терминал
Filename: test_3.2.txt
Parent, child_1 & child_2 delays (separate w/ space): 0 0 0

-----
(program exited with code: 0)
Press return to continue
█
```

Рисунок 7. Тест 3.2: Задержки для родительского процесса, дочернего процесса 1 и дочернего процесса 2 (меняющим исполняемую программу) равны 0, 0 и 0 секунд соответственно



```
1 Parent, child_1 & child_2 delays: 0 0 0
2
3 ***CHILD_1***
4
5 Process ID (PID): 19176
6 Parent process ID (PPID): 19169
7 Session ID (SID): 19166
8 Process group ID (PGID): 19166
9 [real] User ID (UID): 1000
10 Effective user ID (EUID): 1000
11 [real] Group ID (GID): 1000
12 Effective group ID (EGID): 1000
13
14 ***PARENT***
15
16 Process ID (PID): 19169
17 Parent process ID (PPID): 19167
18 Session ID (SID): 19166
19 Process group ID (PGID): 19166
20 [real] User ID (UID): 1000
21 Effective user ID (EUID): 1000
22 [real] Group ID (GID): 1000
23 Effective group ID (EGID): 1000
24
25 ***CHILD_2***
26
27 Process ID (PID): 19177
28 Parent process ID (PPID): 19169
29 Session ID (SID): 19166
30 Process group ID (PGID): 19166
31 [real] User ID (UID): 1000
32 Effective user ID (EUID): 1000
33 [real] Group ID (GID): 1000
34 Effective group ID (EGID): 1000
```

Рисунок 8. Тест 3.2: Порядок вывода процессов в файл – дочерний процесс 1, родительский процесс, дочерний процесс 2 (меняющий исполняемую программу)

Как можно убедиться, при меньшей задержке процесс будет записывать свои данные в необходимый файл быстрее, чем тот процесс, у которого задержка больше, а при равной или отсутствующей задержке процессы будут записывать данные в том порядке, который был установлен в программе.

Например, задержки для родительского процесса, дочернего процесса 1 и дочернего процесса 2 (меняющим исполняемую программу) в 1 случае равны 3, 6 и 9 секунд соответственно, что соответствует выводу «родительский процесс, дочерний процесс 1, дочерний процесс 2 (меняющий исполняемую программу)», что отражает порядок задержек, упорядоченных по возрастанию от меньшей к большей.

Несмотря на то, что в программе содержатся блоки if-else, которые могут быть выполнены 1 раз, программа будет выводить данные для всех процессов. Это происходит потому, что при вызове функции «fork()» дочерний процесс полностью копирует адресное пространство родительского процесса, и, соответственно, продолжает выполнение программы с того же места, на котором остановился родитель. Из-за этого дочерний процесс может выполнить отличные от родителя блоки условий, так как ему присваивается при удачном создании «0», а родителю его ID. Аналогично при выполнении функции «exec1()» будет создаваться новое адресное пространство, даже если дочерний процесс был создан с помощью функции «vfork()», которая позволяет делить его с процессом-родителем.

При равных (например, у всех процессов задержка составляет 3 секунды) или отсутствующих задержках (у всех процессов задержка составляет 0 секунд) процессы будут записывать свои данные, как можно заметить, в порядке «дочерний процесс 1, родительский процесс, дочерний процесс 2 (меняющий исполняемую программу)», так как в коде программы पहले идёт блок, отвечающий за дочерний процесс 1, затем идёт блок, отвечающий за замену программы у дочернего процесса 2, а затем блок родительского процесса, но из-за того, что его адресное пространство отделяется, а не остаётся одним с родителем (и дополнительно тратится время на инициализацию переменных в новой программе), पहले будет записываться информация о родительском процессе, так как он фактически выполнится быстрее. Если бы процесс, созданный с помощью функции «vfork()», не создавал новое адресное пространство с помощью функции «exec1()», то он бы записывал свои данные в файл гораздо быстрее (при условии, что все задержки одинаковы или отсутствуют).

3. Вывод

В ходе выполнения лабораторной работы №3 «Создание и идентификация процессов» были изучены системные функции, позволяющие получить доступ к информации о процессах. Были написаны программы, порождающие 2 потомка от родительского процесса, одна из которых загружается в адресное пространство во время выполнения первой. Были использованы функции «fork()» и «vfork()», порождающие новые процессы с новым адресным пространством и без него соответственно, были использованы функции «getpid()», «getppid()», «getsid()» и другие, которые отвечают за чтение значений атрибутов процесса. Также были проведены тесты с различными временами задержек в программе, в результате которых запись данных об атрибутах процесса в файл производилась в разном порядке, что показало реальное различие в работе процессов. Таким образом и были изучены и использованы системные функции, обеспечивающие порождение и идентификацию процессов.

4. Список использованных источников

1. Онлайн-курс «Организация процессов и программирование в среде Linux» в LMS Moodle [сайт]. URL: <https://vec.etu.ru/moodle/course/view.php?id=9703>.
2. Разумовский Г.В. Организация процессов и программирование в среде Linux: учебно-методическое пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2018. 40с.
3. Проект OpenNet – всё, что связано с открытым ПО, открытыми технологиями, Linux, BSD и Unix [сайт]. URL: <https://www.opennet.ru/cgi-bin/opennet/man.cgi?topic=execl>.
4. Linux manual page [сайт]. URL: <https://man7.org/linux/man-pages/man2/fork.2.html>.
5. Ресурс для IT-специалистов «Хабр» [сайт]. URL: <https://habr.com/ru/company/embox/blog/232605/>.