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

ОТЧЁТ

по лабораторной работе №3

по дисциплине «Организация процессов и программирования в среде Linux»

1. Тема: СОЗДАНИЕ И ИДЕНТИФИКАЦИЯ ПРОЦЕССОВ

|  |  |  |
| --- | --- | --- |
| Студент гр. 9308 |  | Соболев М.С. |
| Преподаватель |  | Разумовский Г.В. |

Оглавление

[1. Введение 3](#__RefHeading___Toc336_311249038)

[1.1. Введение 3](#__RefHeading___Toc15338_3795591338)

[1.2. Порядок выполнения работы 3](#__RefHeading___Toc15340_3795591338)

[1.3. Содержание отчёта 3](#__RefHeading___Toc15342_3795591338)

[2. Тексты программ с комментариями 4](#__RefHeading___Toc5901_764176985)

[2.1. main.cpp 4](#__RefHeading___Toc5903_764176985)

[2.2. executable.cpp 7](#__RefHeading___Toc5905_764176985)

[3. Распечатки файлов, содержащих параметры вызова программы и атрибуты процессов 8](#__RefHeading___Toc5907_764176985)

[3. Вывод 17](#__RefHeading___Toc358_311249038)

[4. Список использованных источников 18](#__RefHeading___Toc360_311249038)

# 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 successed

{

// 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.

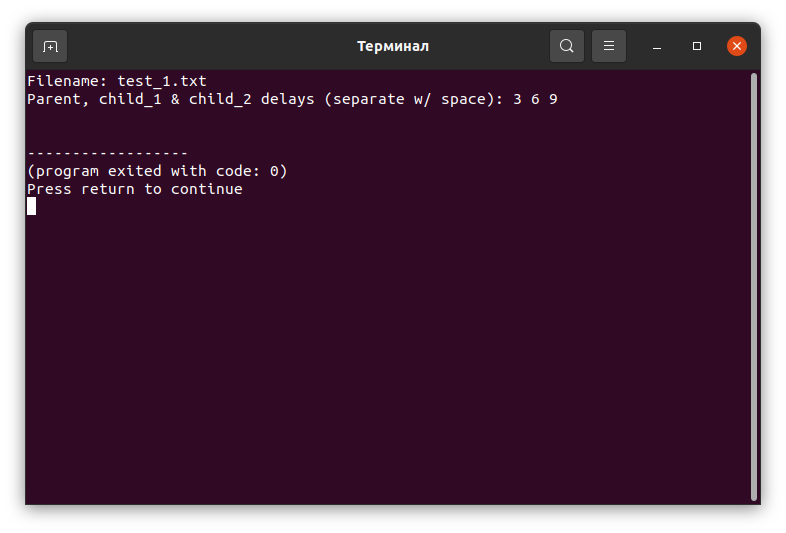


Рисунок 1. Тест 1: Задержки для родительского процесса, дочернего процесса 1 и дочернего процесса 2 (меняющим исполняемую программу) равны 3, 6 и 9 секунд соответственно

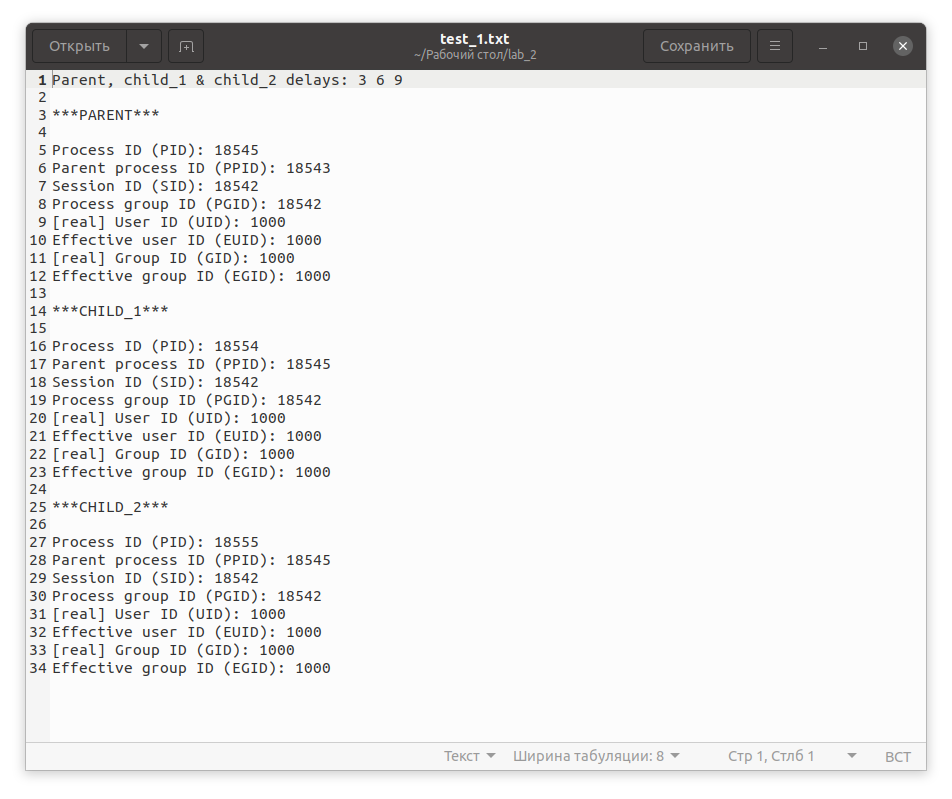


Рисунок 2. Тест 1: Порядок вывода процессов в файл – родительский процесс, дочерний процесс 1, дочерний процесс 2 (меняющий исполняемую программу)

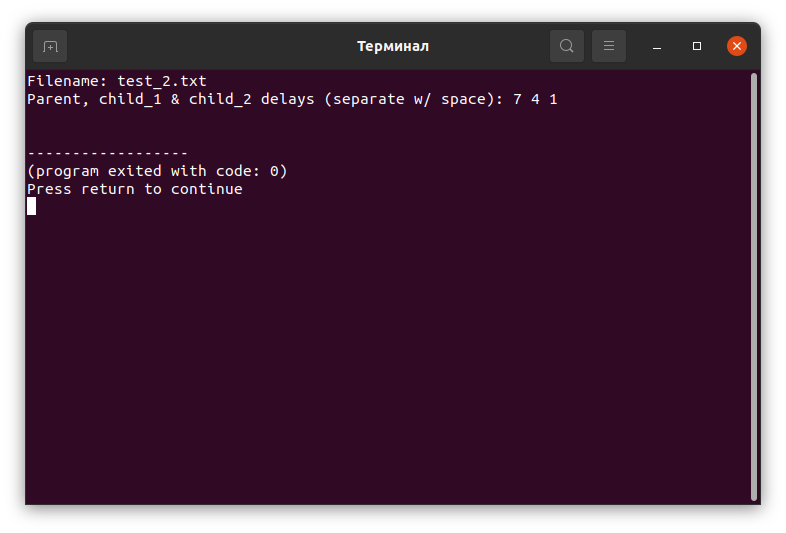


Рисунок 3. Тест 2: Задержки для родительского процесса, дочернего процесса 1 и дочернего процесса 2 (меняющим исполняемую программу) равны 7, 4 и 1 секунд соответственно

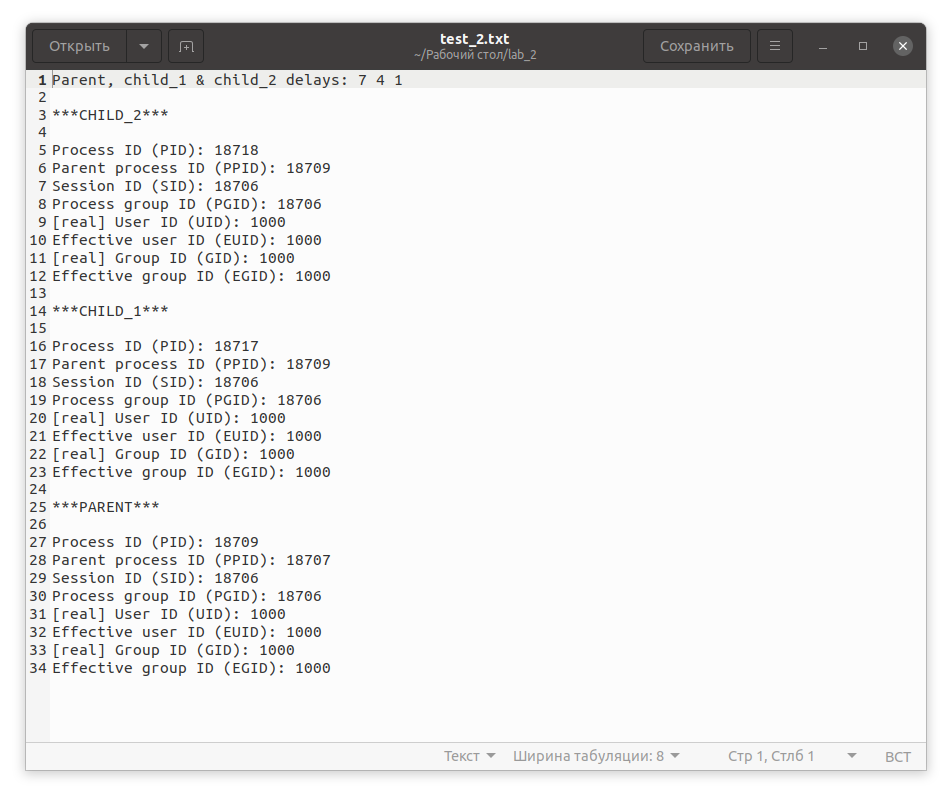


Рисунок 4. Тест 2: Порядок вывода процессов в файл – дочерний процесс 2 (меняющий исполняемую программу), дочерний процесс 1, родительский процесс

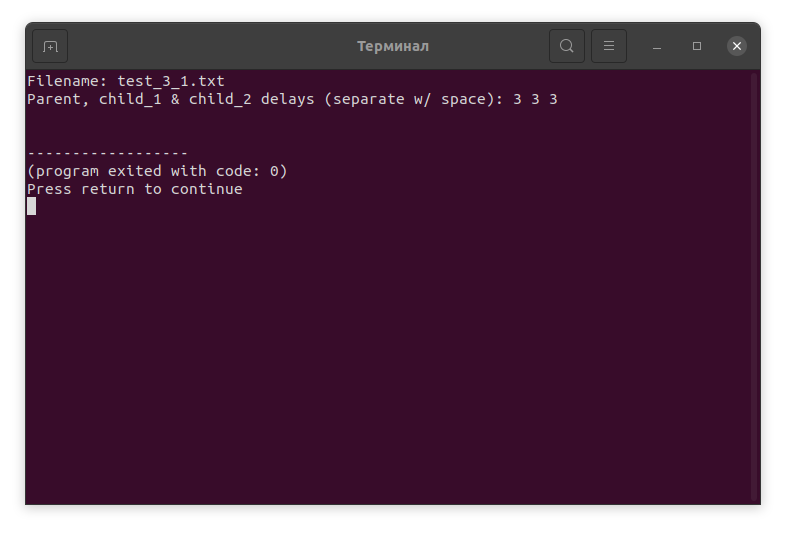


Рисунок 5. Тест 3.1: Задержки для родительского процесса, дочернего процесса 1 и дочернего процесса 2 (меняющим исполняемую программу) равны 3, 3 и 3 секунд соответственно

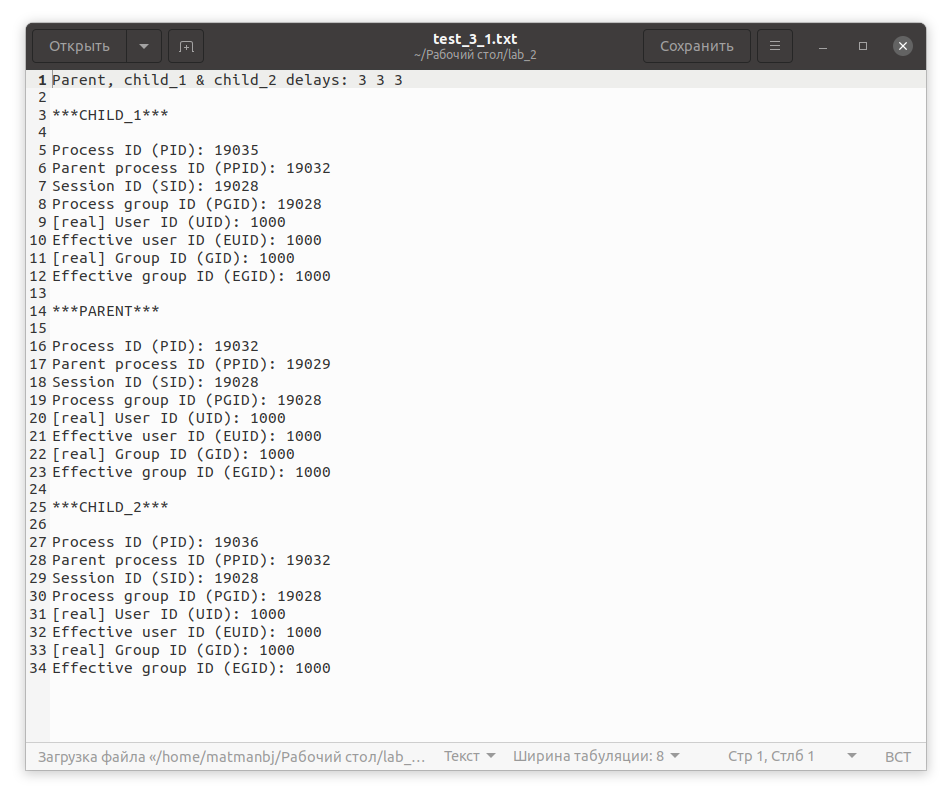


Рисунок 6. Тест 3.1: Порядок вывода процессов в файл – дочерний процесс 1, родительский процесс, дочерний процесс 2 (меняющий исполняемую программу)

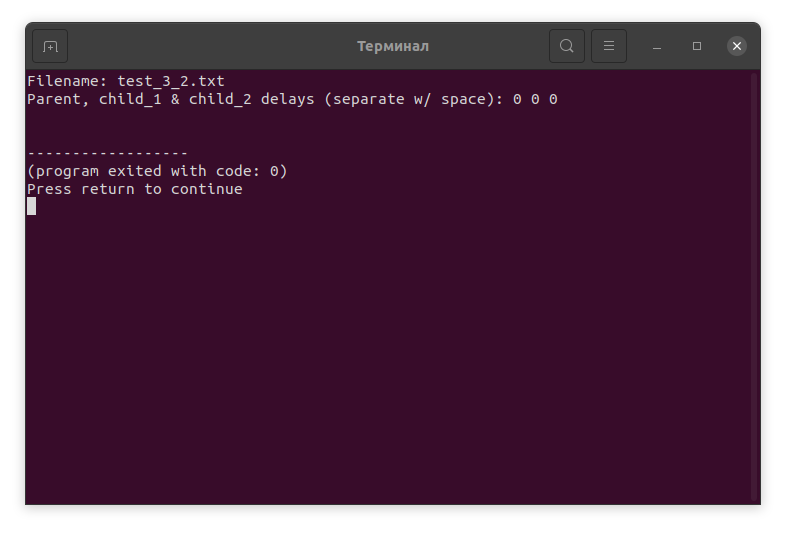


Рисунок 7. Тест 3.2: Задержки для родительского процесса, дочернего процесса 1 и дочернего процесса 2 (меняющим исполняемую программу) равны 0, 0 и 0 секунд соответственно

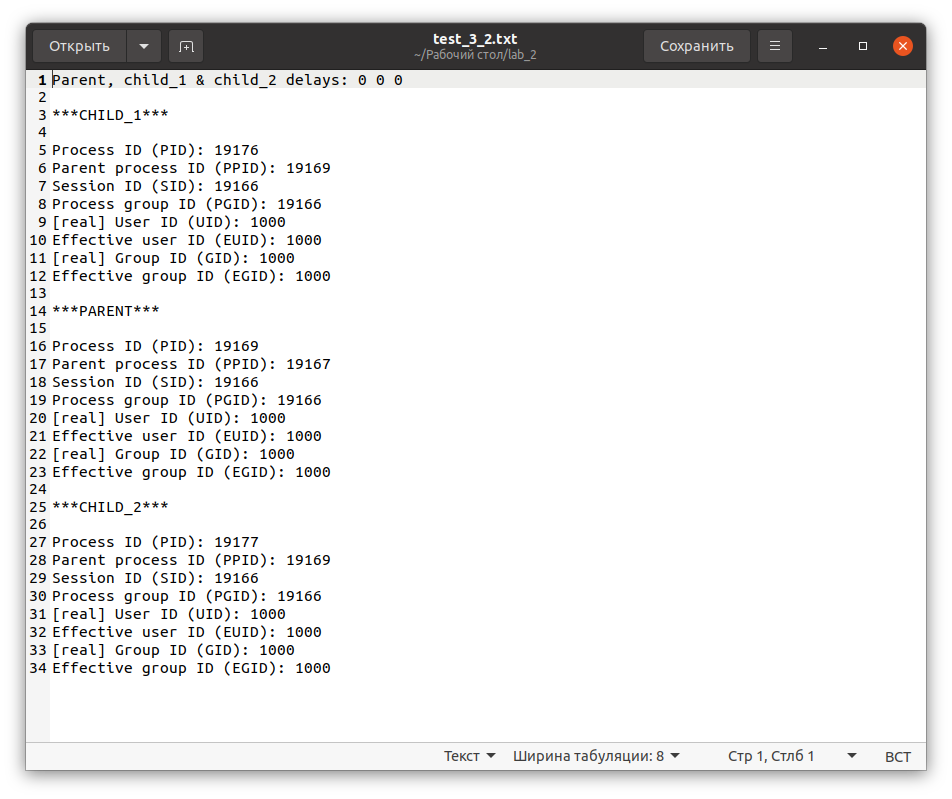


Рисунок 8. Тест 3.2: Порядок вывода процессов в файл – дочерний процесс 1, родительский процесс, дочерний процесс 2 (меняющий исполняемую программу)

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 «execl()» 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 первее 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), первее 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 «execl()», 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 пространство 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/>.