

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

ОТЧЕТ
по лабораторной работе №3
по дисциплине «Организация процессов и программирование в среде
Linux»
ТЕМА: «СОЗДАНИЕ И ИДЕНТИФИКАЦИЯ ПРОЦЕССОВ»

Студент гр. 8308

Тайсумов И.И.

Преподаватель

Разумовский Г.В.

Санкт-Петербург

2021

Цель работы.

Изучение и использование системных функций, обеспечивающих порождение и идентификацию процессов.

Задание.

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 с один файл свои атрибуты с предварительным указанием имени процесса (например: Предок, Потомок1, Потомок2). Имя выходного файла задается 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 ее 3 раза с различными сочетаниями задержек.

Основные теоретические положения.

Единицей управления и потребления ресурсов в ОС служит процесс. Информация о процессе хранится в дескрипторе процесса. Чтобы отличать процессы друг от друга, ОС присваивает каждому процессу уникальный номер, называемый идентификатором процесса. Все процессы в системе связаны отношением предок-потомок и образуют дерево процессов, т. е. у одного процесса может быть только один предок.

Для чтения значений атрибутов процесса можно воспользоваться следующими функциями, определенными в файлах `sys/types.h` и `unistd.h`:

pid_t getpid(void); идентификатор процесса

pid_t getppid(void); идентификатор предка

pid_t getsid(pid_t pid); идентификатор сессии процесса

pid_t getpgid(pid_t pid); идентификатор группы процессов

uid_t getuid(void); реальный идентификатор пользователя

uid_t geteuid(void); эффективный идентификатор пользователя

gid_t getgid(void); реальный групповой идентификатор **gid_t**
getegid(void); эффективный групповой идентификатор

В ОС Ubuntu процесс может быть создан при помощи функций **int fork()** и **int vfork()**. Обе функции возвращают предку идентификатор потомка, а потомку – 0. Отличие в их работе состоит в том, что дочерний процесс, порожденный функцией **vfork()**, разделяет всю память с родительским процессом, включая стек, и родительский процесс блокируется до тех пор, пока дочерний процесс не будет заблокирован или не вызовет функцию **exec** или **_exit**.

Порожденный функцией **fork** или **vfork** процесс всегда выполняет программу предка. Однако любой процесс может перейти к выполнению другой программы, хранящейся в файле на диске. Для этого можно воспользоваться одной из функций семейства **exec**:

```
int execl ( char* name, char* arg0, char* arg1, . . . , char* argn, (char*)0);  
int execv ( char* name, char* argv[ ] );
```

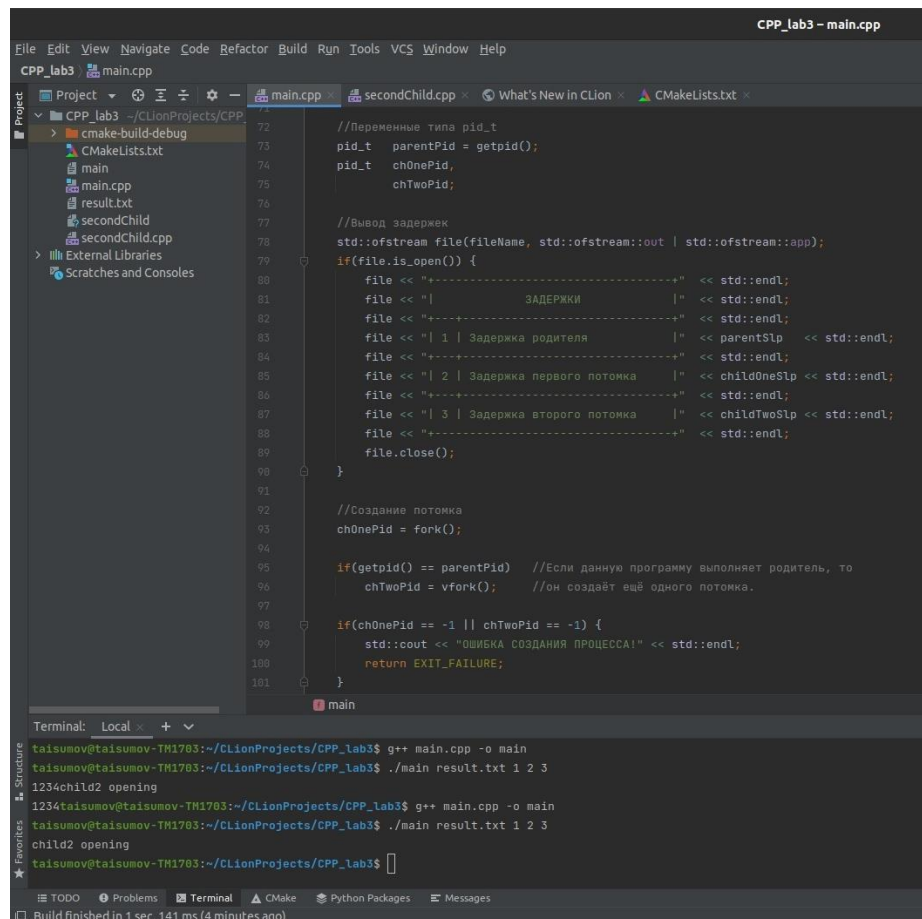
Для реализации пассивного ожидания, переводящего процесс в заблокированное состояние, используется следующий набор функций:

```
int usleep(useconds_t usec); int sleep(unsigned sec);  
int nanosleep(const struct timespec *req, struct timespec *rem); struct timespec {  
time_t tv_sec;  
long tv_nsec};
```

Обработка результатов эксперимента.

В ходе выполнения лабораторной работы были использованы функции, описанные выше в основных теоретических положениях (текст программы см. в Приложении).

Ниже приведён скриншот программы в среде разработки:



The screenshot shows a C++ IDE with the following components:

- Project Explorer:** Shows the project structure for 'CPP_lab3' with files like 'main.cpp', 'secondChild.cpp', 'CMakeLists.txt', and 'result.txt'.
- Source Editor:** Displays the code in 'main.cpp'. The code includes headers for `unistd.h` and `fstream`, defines `pid_t` variables, and uses `fork()` and `vfork()` to create child processes. It also writes to a file named 'result.txt'.
- Terminal:** Shows the execution of the program. The commands and output are:

```
taismov@taismov-TM1703:~/CLionProjects/CPP_lab3$ g++ main.cpp -o main
taismov@taismov-TM1703:~/CLionProjects/CPP_lab3$ ./main result.txt 1 2 3
1234child2 opening
1234taismov@taismov-TM1703:~/CLionProjects/CPP_lab3$ g++ main.cpp -o main
taismov@taismov-TM1703:~/CLionProjects/CPP_lab3$ ./main result.txt 1 2 3
child2 opening
taismov@taismov-TM1703:~/CLionProjects/CPP_lab3$
```

Рисунок 1. Работа над программой

Были использованы следующие команды для компиляции проекта и, собственно, запуска кода:

```
g++ main.cpp -o main
```

```
g++ secondChild.cpp -o secondChild
```

```
./main *файл для вывода* *задержки*
```

Первые две команды компилируют проект, третья – отправляет в основную программу имя файла, в который будут выводиться атрибуты и задержки для процесса-родителя и для дочерних процессов соответственно.

В Приложении 2 указана распечатка текстового документа, содержащего в себе информацию о процессах, ниже – скриншот данного файла:

-----+		
	ЗАДЕРЖКИ	
-----+		
1	Задержка родителя	1
-----+		
2	Задержка первого потомка	2
-----+		
3	Задержка второго потомка	3
-----+		
-----+		
	РОДИТЕЛЬСКИЙ ПРОЦЕСС	
-----+		
1	ID процесса	6522
-----+		
2	ID предка	3961
-----+		
3	ID сессии процесса	3961
-----+		
4	ID группы процессов	6522
-----+		
5	Реальный ID пользователя	1000
-----+		
6	Эффективный ID пользователя	1000
-----+		
7	Реальный групповой ID	1000
-----+		
8	Эффективный групповой ID	1000
-----+		
-----+		
	ДОЧЕРНИЙ ПРОЦЕСС №1	
-----+		
1	ID процесса	6523
-----+		
2	ID предка	902
-----+		
3	ID сессии процесса	3961
-----+		
4	ID группы процессов	6522
-----+		
5	Реальный ID пользователя	1000
-----+		
6	Эффективный ID пользователя	1000

Рисунок 2. Пример вывода программы

Заметим, что первое в файле отобразится тот процесс, чья задержка меньше, и наоборот.

Выводы.

Были изучены и использованы системные функции, обеспечивающие порождение и идентификацию процессов.

Приложение 1.

MAIN.CPP

```
#include <iostream>
#include <fstream>
#include <unistd.h>
#include <string>

void printPidInfo(pid_t pid, std::string fileName, int num) {
    //Функция вывода информации
    std::ofstream file(fileName, std::ofstream::out |
std::ofstream::app);
    std::string output = "";
    //Добавление заголовка к таблице с данными
    switch(num){
        case 1:
            output = "|          РОДИТЕЛЬСКИЙ ПРОЦЕСС          |";
            break;
        case 2:
            output = "|          ДОЧЕРНИЙ ПРОЦЕСС №1          |";
            break;
        case 3:
            output = "|          ДОЧЕРНИЙ ПРОЦЕСС №2          |";
            break;
        default:
            break;
    };
    //Открываем файл как поток и печатаем данные
    if(file.is_open()) {
        file << "+-----+" << std::endl;
        file <<          output          << std::endl;
        file << "+---+-----+" << std::endl;
        file << "| 1 | ID процесса          | " << pid <<
std::endl;
    }
```

```

        file << "+---+-----+" << std::endl;
        file << "| 2 | ID предка          | " << getppid()
<< std::endl;
        file << "+---+-----+" << std::endl;
        file << "| 3 | ID сессии процесса      | " << getsid(pid)
<< std::endl;
        file << "+---+-----+" << std::endl;
        file << "| 4 | ID группы процессов      | " <<
getpgid(pid) << std::endl;
        file << "+---+-----+" << std::endl;
        file << "| 5 | Реальный ID пользователя    | " << getuid() <<
std::endl;
        file << "+---+-----+" << std::endl;
        file << "| 6 | Эффективный ID пользователя | " << geteuid()
<< std::endl;
        file << "+---+-----+" << std::endl;
        file << "| 7 | Реальный групповой ID        | " << getgid() <<
std::endl;
        file << "+---+-----+" << std::endl;
        file << "| 8 | Эффективный групповой ID      | " << getegid()
<< std::endl;
        file << "+---+-----+" << std::endl;
        file.close();
    }
    else std::cout << "ПРОЦЕСС НЕ МОЖЕТ ОТКРЫТЬ ФАЙЛ!" << std::endl;
}

int main(int arc, char* argv[]) {
    //Переменные типа char*
    char* endPointer = nullptr;
    char* fileNameInput = argv[1];
    //Переменные типа long
    long delayParent = 0,
        delayFirstChild = 0,
        delaySecondChild = 0;
    //Переменные типа string

```



```

std::string fileName = "./" + std::string(fileNameInput);
//Переменные задержек
delayParent      = std::strtol(argv[2], &endPointer, 10);
delayFirstChild  = std::strtol(argv[3], &endPointer, 10);
delaySecondChild = std::strtol(argv[4], &endPointer, 10);
//Переменные типа pid_t
pid_t  pidParent = getpid();
pid_t  pidFirstChild,
        pidSecondChild;
//Вывод задержек
std::ofstream file(fileName, std::ofstream::out |
std::ofstream::app);
if(file.is_open()) {
    file << "+-----+" << std::endl;
    file << "|          ЗАДЕРЖКИ          |" << std::endl;
    file << "+---+-----+" << std::endl;
    file << "| 1 | Задержка родителя          |" << delayParent
<< std::endl;
    file << "+---+-----+" << std::endl;
    file << "| 2 | Задержка первого потомка      |" <<
delayFirstChild << std::endl;
    file << "+---+-----+" << std::endl;
    file << "| 3 | Задержка второго потомка      |" <<
delaySecondChild << std::endl;
    file << "+-----+" << std::endl;
    file.close();
}
//Создание потомка
pidFirstChild = fork();
if(getpid() == pidParent)          //Если данную программу
выполняет родитель, то
    pidSecondChild = vfork();      //он создаёт ещё одного
потомка.
if(pidFirstChild == -1 || pidSecondChild == -1) {
    std::cout << "ОШИБКА СОЗДАНИЯ ПРОЦЕССА!" << std::endl;
}

```

```

        return EXIT_FAILURE;
    }
    else if(pidFirstChild > 0 && pidSecondChild > 0) {
        //Ветвь родителя
        sleep(delayParent);
        //Выводим информацию
        printPidInfo(pidParent, fileName, 1);
    }
    else if(pidFirstChild == 0 && pidSecondChild > 0) {
        //Ветвь первого потомка
        sleep(delayFirstChild);

        //Выводим информацию
        pid_t cpid = getpid();
        printPidInfo(cpid, fileName, 2);
        return EXIT_SUCCESS;
    }
    else if (pidFirstChild > 0 && pidSecondChild == 0) {
        //Ветвь второго потомка
        execl("/home/taisumov/CLionProjects/CPP_lab3/secondChild",
//Перенаправляем второй дочерний процесс
            "/home/taisumov/CLionProjects/CPP_lab3/secondChild",
//на выполнение другой программы.
            argv[4], fileNameInput, nullptr);
        return EXIT_SUCCESS;
    }
    return 0;
}

```

SECONDCHILD.CPP

```
#include <iostream>
#include <fstream>
#include <unistd.h>
#include <string>

void printPidInfo(pid_t pid, std::string fileName, int num) {
    std::ofstream file(fileName, std::ofstream::out |
std::ofstream::app);
    std::string output = "";
    switch(num){
        case 1:
            output = "|          РОДИТЕЛЬСКИЙ ПРОЦЕСС          |";
            break;
        case 2:
            output = "|          ДОЧЕРНИЙ ПРОЦЕСС №1          |";
            break;
        case 3:
            output = "|          ДОЧЕРНИЙ ПРОЦЕСС №2          |";
            break;
        default:
            break;
    };
    if(file.is_open()) {
        file << "+-----+" << std::endl;
        file <<          output          << std::endl;
        file << "+---+-----+" << std::endl;
        file << "| 1 | ID процесса          | " << pid <<
std::endl;
        file << "+---+-----+" << std::endl;
        file << "| 2 | ID предка          | " << getppid()
<< std::endl;
        file << "+---+-----+" << std::endl;
```

```

        file << "| 3 | ID сессии процесса          | " << getsid(pid)
<< std::endl;
        file << "+---+-----+" << std::endl;
        file << "| 4 | ID группы процессов          | " <<
getpgid(pid) << std::endl;
        file << "+---+-----+" << std::endl;
        file << "| 5 | Реальный ID пользователя      | " << getuid() <<
std::endl;
        file << "+---+-----+" << std::endl;
        file << "| 6 | Эффективный ID пользователя  | " << geteuid()
<< std::endl;
        file << "+---+-----+" << std::endl;
        file << "| 7 | Реальный групповой ID          | " << getgid() <<
std::endl;
        file << "+---+-----+" << std::endl;
        file << "| 8 | Эффективный групповой ID      | " << getegid()
<< std::endl;
        file << "+---+-----+" << std::endl;
        file.close();
    }
    else std::cout << "ПРОЦЕСС НЕ МОЖЕТ ОТКРЫТЬ ФАЙЛ!" << std::endl;
}

```

```

int main(int arc, char* argv[]) {
    char* endPointer    = nullptr;
    long delay          = std::strtol(argv[1], &endPointer, 10);
    char* fileNameInput = argv[2];
    std::string fileName(fileNameInput);
    sleep(delay);

    pid_t childPid = getpid();
    printPidInfo(childPid, fileName, 3);
    return EXIT_SUCCESS;
}

```

Приложение 2.

Пример 1:

+-----+		
	ЗАДЕРЖКИ	
+-----+		
1	Задержка родителя	1
+-----+		
2	Задержка первого потомка	2
+-----+		
3	Задержка второго потомка	3
+-----+		
	РОДИТЕЛЬСКИЙ ПРОЦЕСС	
+-----+		
1	ID процесса	6522
+-----+		
2	ID предка	3961
+-----+		
3	ID сессии процесса	3961
+-----+		
4	ID группы процессов	6522
+-----+		
5	Реальный ID пользователя	1000
+-----+		
6	Эффективный ID пользователя	1000
+-----+		
7	Реальный групповой ID	1000
+-----+		
8	Эффективный групповой ID	1000
+-----+		
	ДОЧЕРНИЙ ПРОЦЕСС №1	
+-----+		
1	ID процесса	6523
+-----+		
2	ID предка	902
+-----+		
3	ID сессии процесса	3961
+-----+		
4	ID группы процессов	6522
+-----+		
5	Реальный ID пользователя	1000
+-----+		
6	Эффективный ID пользователя	1000
+-----+		
7	Реальный групповой ID	1000
+-----+		
8	Эффективный групповой ID	1000
+-----+		

+-----+		
	ДОЧЕРНИЙ ПРОЦЕСС №2	
+-----+		
1	ID процесса	6524
+-----+		
2	ID предка	902
+-----+		
3	ID сессии процесса	3961
+-----+		
4	ID группы процессов	6522
+-----+		
5	Реальный ID пользователя	1000
+-----+		
6	Эффективный ID пользователя	1000
+-----+		
7	Реальный групповой ID	1000
+-----+		
8	Эффективный групповой ID	1000
+-----+		

Пример 2:

+-----+		
	ЗАДЕРЖКИ	
+-----+		
1	Задержка родителя	5
+-----+		
2	Задержка первого потомка	6
+-----+		
3	Задержка второго потомка	4
+-----+		
+-----+		
	ДОЧЕРНИЙ ПРОЦЕСС №2	
+-----+		
1	ID процесса	7415
+-----+		
2	ID предка	7412
+-----+		
3	ID сессии процесса	3961
+-----+		
4	ID группы процессов	7412
+-----+		
5	Реальный ID пользователя	1000
+-----+		
6	Эффективный ID пользователя	1000
+-----+		
7	Реальный групповой ID	1000
+-----+		
8	Эффективный групповой ID	1000
+-----+		
+-----+		
	РОДИТЕЛЬСКИЙ ПРОЦЕСС	
+-----+		
1	ID процесса	7412
+-----+		
2	ID предка	3961
+-----+		
3	ID сессии процесса	3961
+-----+		
4	ID группы процессов	7412
+-----+		
5	Реальный ID пользователя	1000
+-----+		
6	Эффективный ID пользователя	1000
+-----+		
7	Реальный групповой ID	1000
+-----+		
8	Эффективный групповой ID	1000
+-----+		

+-----+		
	ДОЧЕРНИЙ ПРОЦЕСС №1	
+-----+		
1	ID процесса	7414
+-----+		
2	ID предка	902
+-----+		
3	ID сессии процесса	3961
+-----+		
4	ID группы процессов	7412
+-----+		
5	Реальный ID пользователя	1000
+-----+		
6	Эффективный ID пользователя	1000
+-----+		
7	Реальный групповой ID	1000
+-----+		
8	Эффективный групповой ID	1000
+-----+		

Пример 3:

ЗАДЕРЖКИ		
1	Задержка родителя	4
2	Задержка первого потомка	3
3	Задержка второго потомка	2
ДОЧЕРНИЙ ПРОЦЕСС №2		
1	ID процесса	7482
2	ID предка	7480
3	ID сессии процесса	3961
4	ID группы процессов	7480
5	Реальный ID пользователя	1000
6	Эффективный ID пользователя	1000
7	Реальный групповой ID	1000
8	Эффективный групповой ID	1000
ДОЧЕРНИЙ ПРОЦЕСС №1		
1	ID процесса	7481
2	ID предка	7480
3	ID сессии процесса	3961
4	ID группы процессов	7480
5	Реальный ID пользователя	1000
6	Эффективный ID пользователя	1000
7	Реальный групповой ID	1000
8	Эффективный групповой ID	1000

+-----+		
	РОДИТЕЛЬСКИЙ ПРОЦЕСС	
+-----+		
1	ID процесса	7480
+-----+		
2	ID предка	3961
+-----+		
3	ID сессии процесса	3961
+-----+		
4	ID группы процессов	7480
+-----+		
5	Реальный ID пользователя	1000
+-----+		
6	Эффективный ID пользователя	1000
+-----+		
7	Реальный групповой ID	1000
+-----+		
8	Эффективный групповой ID	1000
+-----+		