

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

ОТЧЕТ
по лабораторной работе №4
по дисциплине «Организация процессов и программирование в среде
Linux»
Темы: Управление потоками

Студент гр. 8306

Пеунов В.В.

Преподаватель

Разумовский Г.В.

Санкт-Петербург,

2021

Цель работы

Знакомство с организацией потоков и способами синхронизации потоков и потомков

Задание на лабораторную работу

1. Написать программу, которая открывает текстовый файл, порождает поток, а затем ожидает его завершения. Поток в качестве параметра передается дескриптор файла. Поток выводит на экран класс планирования, текущий, минимальный и максимальный приоритеты, содержимое файла и закрывает файл. После завершения работы потока программа должна вывести текущий приоритет и проверить – закрыт ли файл, и если он не закрыт, то принудительно закрыть. Результат проверки должен быть выведен на экран.

2. Дважды окомпилировать программу при условии, когда поток закрывает и не закрывает файл. Затем последовательно запустить оба варианта.

3. Написать программу, которая открывает входной файл и 2 выходных файла. Затем она должна в цикле построчно читать входной файл и порождать 2 потока. Одному потоку передавать нечетную строку, а другому – четную. Оба потока должны работать параллельно. Каждый поток записывает в свой выходной файл полученную строку и завершает работу. Программа должна ожидать завершения работы каждого потока и повторять цикл порождения потоков и чтения строк входного файла, пока не прочтет последнюю строку, после чего закрыть все файлы.

Описание работы

Часть 1 (Запуск 1)

- Лабораторная 4. Задание 1
- (Главный поток) - файл успешно открыт. Дескриптор: 3

- (Дочерний поток) - класс планирования: SCHED_OTHER
- (Дочерний поток) - текущий приоритет: 0
- (Дочерний поток) - минимальный приоритет: 0
- (Дочерний поток) - максимальный приоритет: 0
- (Дочерний поток) - содержимое файла: Smells Like Teen Spirit
- (Главный поток) - текущий приоритет: 0
- (Главный поток) - закрываем файл с дескриптором: 3
- (Главный поток) - файл не был закрыт дочерним потоком

Часть 1 (Запуск 2)

- Лабораторная 4. Задание 1
- (Главный поток) - файл успешно открыт. Дескриптор: 3
- (Дочерний поток) - класс планирования: SCHED_OTHER
- (Дочерний поток) - текущий приоритет: 0
- (Дочерний поток) - минимальный приоритет: 0
- (Дочерний поток) - максимальный приоритет: 0
- (Дочерний поток) - содержимое файла: Smells Like Teen Spirit
- (Дочерний поток) - закрываем файл с дескриптором: 3
- (Главный поток) - текущий приоритет: 0
- (Главный поток) - файл был закрыт дочерним потоком

После написания программы было проведено 3 эксперимента. С разными значениями задержек.

Часть 2

Текст в файле output.txt:

- Раз
- Два
- Три
- Четыре
- Пять
- Шесть

- Семь
- Восемь
- Девять
- Десять
- Одиннадцать

Файл output1.txt

- Раз
- Три
- Пять
- Семь
- Девять
- Одиннадцать

Файл output2.txt

- Два
- Четыре
- Шесть
- Восемь
- Десять

Вывод

В ходе лабораторной работы мы познакомились с механизмом управления потоками и способами организации предков и потомков.

Приложение А. Задание 1

```
#include <iostream>
```

```
#include <string>
```

```
#include <fcntl.h>
```

```
#include <pthread.h>
```

```
#include <unistd.h>
```

```
using namespace std;
```

```
/*
```

```
 * Написать программу, которая открывает текстовый файл, порождает поток, а затем  
 * ожидает его завершения.
```

```
 * Поток в качестве параметра передается дескриптор файла. Поток выводит на экран  
 * класс планирования,
```

```
 * текущий, минимальный и максимальный приоритеты, содержимое файла и закрывает  
 * файл.
```

```
 * После завершения работы потока программа должна вывести
```

```
 * текущий приоритет и проверить – закрыт ли файл, и если он не закрыт, то
```

```
 * принудительно закрыть. Результат проверки должен быть выведен на экран
```

```
*/
```

```
int openFile(const string& filePath, const string& threadName);
```

```
void closeFile(int handle, const string& threadName);
```

```
int error(const string& errorMessage);
```

```
void printThreadInfo(const string& threadName);
```

```

void* threadFunction(void* arg);

bool checkFileIsClosed(int handle);

void printFileContain(int handle, const string& threadName);

void print_current_priority(const string& threadName);


int main() {

    cout << "Лабораторная 4. Задание 1" << endl;


    //инициализация переменных

    pthread_t thread;

    string filePath = "/home/peunov/highschool/opp-linux/lr4/files/text.txt";


    //открываем файл и получаем дескриптор (handle)

    int handle = openFile(filePath, "Главный поток");

    if(handle == -1){ return error("Не удалось открыть файл");}


    //порождаем поток и передаем дескриптор файла

    pthread_create(&thread, nullptr, threadFunction, &handle);


    //ожидает завершение потока

    pthread_join(thread, nullptr);


    //вывести текущий приоритет

```

```

print_current_priority("Главный поток");

//проверить закрыт ли файл

bool fileIsClosed = checkFileIsClosed(handle);

//если файл открыт, то закрыть

if(!fileIsClosed){

    closeFile(handle, "Главный поток");

}


//вывести результат проверки

if(fileIsClosed) {

    cout << "(Главный поток) - файл был закрыт дочерним потоком";

} else {

    cout << "(Главный поток) - файл не был закрыт дочерним потоком";

}


return 0;

}


void* threadFunction(void* arg){

    //получение данных из главного потока

    int handle = *((int *) arg);

```

```

// Вывод информации о потоке

printThreadInfo("Дочерний поток");


// Поток выводит содержимое файла

printFileContain(handle, "Дочерний поток");


// Поток закрывает файл (опционально)

//closeFile(handle, "Дочерний поток");


pthread_exit(nullptr);
}


bool checkFileIsClosed(int handle){

    return fcntl(handle, F_GETFD) == -1;

}


void closeFile(int handle, const string& threadName){

    cout << "(" << threadName << ") - закрываем файл с дескриптором: " << handle << endl;

    close(handle);

}


int openFile(const string& filePath, const string& threadName){

    int handle = open(filePath.c_str(), O_RDONLY);

    if(handle == -1){

        cout << "Ошибка открытия файла" << endl;
    }
}

```



```

        return -1;
    }

    cout << "(" << threadName << ") - файл успешно открыт. " << "Дескриптор: " << handle
    << endl;

    return handle;
}

int error(const string& errorMessage){

    cout << "Ошибка:" << errorMessage;

    return -1;
}

void printThreadInfo(const string& threadName){

    //получение данных о потоке

    int policy;

    struct sched_param param{};

    pthread_getschedparam(pthread_self(), &policy, &param);

    // Поток выводит класс планирования

    switch(policy){

        case SCHED_FIFO:

            cout << "(" << threadName << ") - класс планирования: SCHED_FIFO" << endl;

            break;

        case SCHED_RR:

```

```

        cout << "(" << threadName << ") - класс планирования: SCHED_RR" << endl;

        break;

    case SCHED_OTHER:

        cout << "(" << threadName << ") - класс планирования: SCHED_OTHER" << endl;

        break;

    default:

        break;

}

// Поток выводит текущий, минимальный и максимальный приоритеты

cout << "(" << threadName << ") - текущий приоритет: " << param.sched_priority << endl;

        cout << "(" << threadName << ") - минимальный приоритет: " <<
sched_get_priority_min(policy) << endl;

        cout << "(" << threadName << ") - максимальный приоритет: " <<
sched_get_priority_max(policy) << endl;

}

void printFileContain(int handle, const string& threadName){

    char buffer[1024];

    size_t buffer_size = sizeof(buffer);

    ssize_t bytes_read = read(handle, buffer, buffer_size);

    buffer[bytes_read] = '\0';

    cout << "(" << threadName << ") - содержимое файла: " << buffer;

}

void print_current_priority(const string& threadName){

```

```
int policy;

struct sched_param param{};

pthread_getschedparam(pthread_self(), &policy, &param);

// Поток выводит текущий, минимальный и максимальный приоритеты

cout << "(" << threadName << ") - текущий приоритет: " << param.sched_priority << endl;

}
```

Приложение Б. Задание 2

```
#include <iostream>

#include <fstream>

#include <pthread.h>

using namespace std;

ofstream outputFirst, outputSecond;

ifstream input;

string inputPath = "/home/peunov/highschool/opp-linux/lr4/files/input.txt";

string outputFirstPath = "/home/peunov/highschool/opp-linux/lr4/files/output1.txt";

string outputSecondPath = "/home/peunov/highschool/opp-linux/lr4/files/output2.txt";

void* threadFunctionFirst(void *arg){

    string line = *((string *) arg);

    outputFirst << line << endl;

    pthread_exit(NULL);

}

void* threadFunctionSecond(void *arg){

    string line = *((string *) arg);

    outputSecond << line << endl;

    pthread_exit(NULL);

}
```

```

int main(){

    pthread_t thread1, thread2;

    string str1, str2;

    bool flag1, flag2;


    input.open(inputPath);

    outputFirst.open(outputFirstPath);

    outputSecond.open(outputSecondPath);


    while(true){

        flag1 = (bool)getline(input, str1);

        flag2 = (bool)getline(input, str2);


        if(flag1) pthread_create(&thread1, NULL, threadFunctionFirst, &str1);

        if(flag2) pthread_create(&thread2, NULL, threadFunctionSecond, &str2);


        if(flag1) pthread_join(thread1, NULL);

        if(flag1) pthread_join(thread2, NULL);


        if(not flag1 or not flag2) break;

    }


    input.close();

    outputFirst.close();

    outputSecond.close();

```

```
    return 0;  
}
```