**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»

|  |  |
| --- | --- |
| Студент гр. 1308 | Мельник Д. А. |
| Преподаватель | Разумовский Г. В. |

Тема: Управление потоками

***Цель работы***: знакомство с организацией потоков и способами синхронизации предков и потомков.

**Задание**:

1. Написать программу, которая открывает текстовый файл, порождает

поток, а затем ожидает его завершения. Потоку 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 должнавывести

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.

4. Откомпилировать программу и запустить ее.

**Листинги программ**

*main\_1.cpp:*

*#include <iostream>*

*#include <fstream>*

*#include <pthread.h>*

*#include <sched.h> // Для работы с приоритетами и классами планирования*

*#include <unistd.h> // Для проверки состояния файла*

*using namespace std;*

*// Функция потока*

*static void \*threadFunc(void \*arg) {*

*fstream \*file = static\_cast<fstream\*>(arg);*

*// Получение планировщика и приоритетов*

*int policy;*

*struct sched\_param param;*

*pthread\_getschedparam(pthread\_self(), &policy, &param);*

*// Вывод информации о планировании и приоритетах*

*cout << "Планировщик: " << (policy == SCHED\_OTHER ? "SCHED\_OTHER" : policy == SCHED\_RR ? "SCHED\_RR" : "SCHED\_FIFO") << endl;*

*cout << "Текущий приоритет: " << param.sched\_priority << endl;*

*cout << "Минимальный приоритет: " << sched\_get\_priority\_min(policy) << endl;*

*cout << "Максимальный приоритет: " << sched\_get\_priority\_max(policy) << endl;*

*// Чтение содержимого файла*

*string line;*

*cout << "Содержимое файла:" << endl;*

*if (file->is\_open()) {*

*while (getline(\*file, line)) {*

*cout << line << endl;*

*}*

*//file->close(); // удалить, чтобы поток не закрывал файл*

*} else {*

*cout << "Ошибка открытия файла." << endl;*

*}*

*pthread\_exit(0);*

*}*

*int main() {*

*pthread\_t thread;*

*pthread\_attr\_t attr;*

*fstream inFile;*

*// Открытие файла*

*inFile.open("test.txt");*

*if (!inFile.is\_open()) {*

*cerr << "Не удалось открыть файл test.txt" << endl;*

*return 1;*

*}*

*// Инициализация атрибутов потока*

*pthread\_attr\_init(&attr);*

*// Создание потока*

*pthread\_create(&thread, &attr, &threadFunc, &inFile);*

*// Ожидание завершения потока*

*pthread\_join(thread, NULL);*

*// Проверка состояния файла*

*if (!inFile.is\_open()) {*

*cout << "Файл закрыт." << endl;*

*} else {*

*cout << "Файл все еще открыт. Закрываю принудительно..." << endl;*

*inFile.close();*

*}*

*// Вывод текущего приоритета в главном потоке*

*int policy;*

*struct sched\_param param;*

*pthread\_getschedparam(pthread\_self(), &policy, &param);*

*cout << "Текущий приоритет главного потока: " << param.sched\_priority << endl;*

*return 0;*

*}*

*main\_2.cpp:*

*#include <iostream>*

*#include <fstream>*

*#include <pthread.h>*

*#include <unistd.h>*

*using namespace std;*

*ifstream input;*

*ofstream output1, output2;*

*static void \* threadFunc1(void \*arg){*

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

*output1 << line << endl;*

*cout << "Write line 1" << endl;*

*//usleep(1000000);*

*pthread\_exit(0);*

*}*

*static void \* threadFunc2(void \*arg){*

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

*output2 << line << endl;*

*cout << "Write line 2" << endl;*

*//usleep(1000000);*

*pthread\_exit(0);*

*}*

*int main(){*

*input.open("input.txt");*

*int lCnt = 0;*

*bool first = true, second = true;*

*if(!input.is\_open()){*

*perror("Не удалось открыть входной файл");*

*exit(1);*

*}*

*input.seekg(0);*

*output1.open("output1.txt");*

*output2.open("output2.txt");*

*pthread\_t thread\_1, thread\_2;*

*pthread\_attr\_t attr;*

*string line1, line2;*

*pthread\_attr\_init(&attr);*

*while(first && second){ //пока не закончится файл*

*first = true;*

*second = true;*

*if(getline(input, line1))*

*pthread\_create(&thread\_1, &attr, &threadFunc1, &line1);*

*else first = false;*

*if(getline(input, line2))*

*pthread\_create(&thread\_2, &attr, &threadFunc2, &line2);*

*else second = false;*

*pthread\_join(thread\_1, NULL);*

*pthread\_join(thread\_2, NULL);*

*}*

*pthread\_attr\_destroy(&attr);*

*input.close();*

*output1.close();*

*output2.close();*

*return 0;*

*}*

**Распечатки файлов программы main\_2.cpp**

***Входной файл input.txt:***

11111111

22222222

33333333

44444444

55555555

66666666

77777777

88888888

!1111111

!2222222

!3333333

!4444444

!5555555

!6666666

!7777777

!8888888

\*1111111

\*2222222

\*3333333

\*4444444

\*5555555

\*6666666

\*7777777

\*8888888

***Выходной файл output1.txt:***

11111111

33333333

55555555

77777777

!1111111

!3333333

!5555555

!7777777

\*1111111

\*3333333

\*5555555

\*7777777

***Выходной файл output2.txt:***

22222222

44444444

66666666

88888888

!2222222

!4444444

!6666666

!8888888

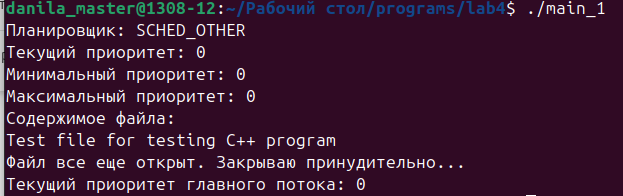
\*2222222

\*4444444

\*6666666

\*8888888

**Результат работы программы main\_1.cpp**



При работе программы main\_2.cpp видно корректное взаимодействие потоков. Из-за “ручного” управления потоками (строки распределяются заранее; новая итерация цикла не начнётся пока не завершится выпонение всех потоков) отсутсвует какая-либо конкуренция, благодаря чему достигается нужный результат.