**МИНОБРНАУКИ РОССИИ**

**САНКТ–ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра ВТ**

**ОТЧЕТ**

**по лабораторной работе №4**

**по дисциплине *«Организация процессов и программирование в среде Linux»***

**Тема: «Управление потоками»**

Cтудентка гр. 7306 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Мищенко А.В.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Разумовский Г.В.

Санкт–Петербург

2020

**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 переда-ется дескриптор файла. Поток выводит на экран класс планирования, теку-щий, минимальный и максимальный приоритеты, содержимое файла и за-крывает 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. Дважды окомпилировать программу при условии, когда поток закрывает и не закрывает файл. Затем последовательно запустить оба варианта.
2. 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. Откомпилировать программу и запустить ее.

**Ход работы:**

1. Для выполнения лабораторной работы было написано приложение, состоящее из одного исходного файла main.cpp. Его код приведен в приложении 1 с соответствующими комментариями.

Были скомпилированы две программы: normMain - с закрытием файла в коде потока и main - без закрытия файла в коде потока и с последующей проверкой закрытия файла в коде основного потока. Обе программы выводят информацию о приоритетах системы и читают содержимое файла file.txt на экран. На рисунке 1 и 2 показаны запуски приложений normMain и main соотвественно.

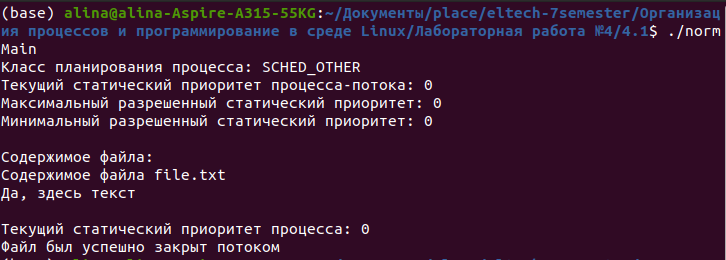


Рисунок 1. Запуск приложения с закрытием файла в коде потока

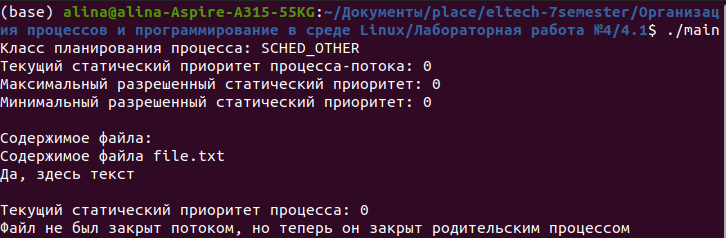


Рисунок 2. Содержимое файла file.txt при задержках 4, 3, 2

1. Для выполнения лабораторной работы была написана программа main.cpp, исходный код которого представлен в приложении 2. Приложение читает файл mainFile.txt с помощью двух потоков, которые читают построчно нечетные и четные строки и записывают их каждый в свои файлы oddFile.txt и eventFile.txt соответственно. Содержимое файлов mainFile.txt, oddFile.txt и eventFile.txt после запуска приложения представлено на рисунках 3-5.

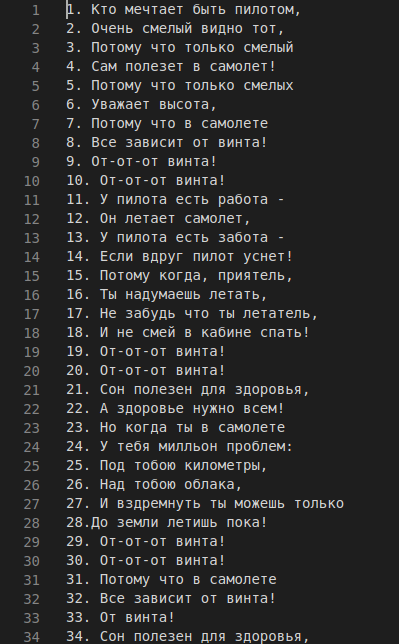


Рисунок 3. Содержимое файла mainFile.txt

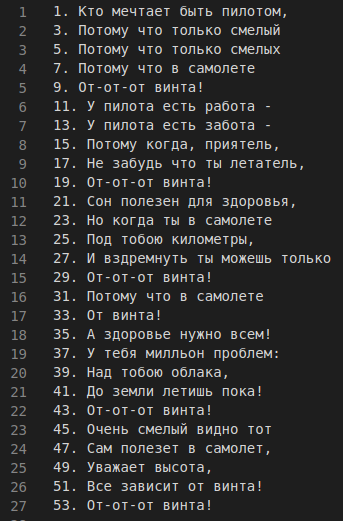


Рисунок 4. Содержимое файла oddFile.txt

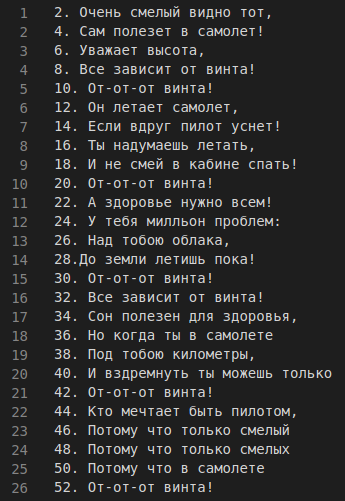


Рисунок 5. Содержимое файла evenFile.txt

**Вывод:**

В ходе лабораторной работы я научилась создавать потоки с помощью инструментов pthread.

**Приложения**

Приложение 1. Файл main.cpp

#include <fcntl.h> /\* open() and O\_XXX flags \*/

#include <sys/types.h>

#include <sys/resource.h>

#include <sys/time.h>

#include <unistd.h> /\* close() \*/

#include <iostream>

#include <pthread.h>

#include <errno.h>

#include "sched.h"

//структура для передачи аргументов потоку

struct argStruct{

int fd; //дескриптор файла

};

//функция, которую будет выполнять поток

void\* threadFunc(void\* args){

//приведение аргументов к необходимому типу argStruct

argStruct\* arg = (argStruct\*) args;

//вспомогательная переменная для чтения файла, по которой определяем закончилось ли чтение файла

ssize\_t ret = 0;

//вспомогательная переменная, в которую будет записан текущий прочитанный символ

char ch;

//вспомогательная переменная для чтения текущего приоритета процесса

//получаем текущий pid процесса

pid\_t pid = getpid();

//получаем номер класса планирования

int policy = sched\_getscheduler(pid);

//и выводим его значение на экран

printf("Класс планирования процесса: ");

switch(policy){

case SCHED\_OTHER:

printf("SCHED\_OTHER\n");

break;

case SCHED\_BATCH:

printf("SCHED\_BATCH\n");

break;

case SCHED\_IDLE:

printf("SCHED\_OTHER\n");

break;

case SCHED\_FIFO:

printf("SCHED\_FIFO\n");

break;

case SCHED\_RR:

printf("SCHED\_RR\n");

}

//получаем значение приоритета по pid процессу

printf("Текущий статический приоритет процесса-потока: %d\n",getpriority(PRIO\_PROCESS,pid ));

printf("Максимальный разрешенный статический приоритет: %d\n",sched\_get\_priority\_max(policy));

printf("Минимальный разрешенный статический приоритет: %d\n", sched\_get\_priority\_min(policy));

printf("\nСодержимое файла: \n");

//Выводим содержимое файла

while ((ret = read (arg->fd, &ch, 1)) > 0)

{

putchar (ch);

}

//закрываем файл

// close(arg->fd);

return 0;

}

//главная функция

int main(){

//переменная потока

pthread\_t thread;

//переменная с передаваемыми параметрами потоку

argStruct arg;

int status; //вспомогательная переменная для определения ошибок

//вспомогательная переменная для чтения текущего приоритета процесса

//имя файла

char nameFile[] = "file.txt";

//открываем файл с именем nameFile только для чтения и возвращаем его дескриптор в переменную fd

int fd = open(nameFile, O\_RDONLY);

//заполняем структуру для передачи данных в поток

arg.fd = fd;

//создаем поток с выполнением функции threadFunc с аргументами arg

status = pthread\_create(&thread, NULL, threadFunc, (void\*) &arg);

//Если поток создать не удалось выводим ошибку

if(status != 0){

printf("Невозможно создать поток, код ошибки: %d\n", status);

exit(-11);

}

//приостанавливаем основной поток, пока не завершится поток thread

status = pthread\_join(thread, NULL);

if(status != 0){

printf("Невозможно приостановить основной поток, код ошибки: %d\n", status);

exit(-12);

}

printf("\nТекущий статический приоритет процесса: %d\n",getpriority(PRIO\_PROCESS,getpid()));

//с помощью fcntl пытаемся получить флаги дескриптора fd, если произошла ошибка функция вернет -1 и

//сгенерируется ошибка, что fd не является открытым дескриптором

if(fcntl(fd, F\_GETFD) != -1 || errno != EBADF){

//если поток не закрыл файл - главный процесс должен его закрыть

if(close(fd) != -1){

printf("Файл не был закрыт потоком, но теперь он закрыт родительским процессом\n");

}

}

else {

printf("Файл был успешно закрыт потоком\n");

}

}

Приложение 2. Файл main.cpp

#include <fcntl.h>

#include <string> // подключаем строки

#include <fstream> // подключаем файлы

#include <pthread.h>

#include <unistd.h> /\* close() \*/

#include <sys/types.h>

#include <sys/stat.h>

#include <iostream>

using namespace std;

//структура для передачи аргументов потоку

struct argStruct{

string str; //строка

int fd; //дескриптор файла

//конструктор с аргументами

argStruct(string str, int fd): fd(fd), str(str) {};

//конструктор по умолчанию

argStruct() {};

};

//функция, которую будет выполнять поток по нечетным строкам

static void\* threadFuncOdd(void\* args){

//приведение аргументов к необходимому типу argStruct

argStruct arg = \*((argStruct\*) args);

//записываем в файл с дескриптором fd строку str с длиной str.length()

write(arg.fd,(arg.str).c\_str(),(arg.str).length());

//завершаем работу нити

pthread\_exit(0);

}

//функция, которую будет выполнять поток по четным строкам

static void\* threadFuncEven(void\* args){

//приведение аргументов к необходимому типо argStruct

argStruct arg = \*((argStruct\*) args);

//Записываем в файл с дескриптором fd строку str с длиной str.length()

write(arg.fd, (arg.str).c\_str(), (arg.str).length());

//завершаем работу нити

pthread\_exit(0);

}

int main(){

pthread\_t threadOdd, //1 поток

threadEven; //2 поток

pthread\_attr\_t attr; //атрибуты потоков

pthread\_attr\_init(&attr); //инициализируем атрибуты

//открываем файл mainFile.txt, из которого будем считывать строчки

ifstream mainFile("mainFile.txt");

//создаем файлы для потоков, возвращается дескрипторы открытых файлов

Int fdOdd = open("oddFile.txt", O\_WRONLY|O\_CREAT, S\_IRUSR|S\_IWUSR|S\_IRGRP|S\_IWGRP|S\_IROTH );

int fdEven = open("evenFile.txt", O\_WRONLY|O\_CREAT, S\_IRUSR|S\_IWUSR|S\_IRGRP|S\_IWGRP|S\_IROTH);

//объявляем статические переменные аргументов функций потоков

static argStruct arg1, arg2;

//флаги завершения чтения файла, если файл содержит четное количество строк или нечетное

bool isNotEndOdd = true;

bool isNotEndEven = true;

//пока не достигнут конец файла

while((isNotEndEven && isNotEndOdd)){

//переменные, куда будут записываться строки из файла

string str1, str2;

//считываем строку по нечетному порядку

if(getline(mainFile,str1)){

//присваиваем аргументы функции - строка и дескриптор файла

arg1 = argStruct(str1+"\n",fdOdd);

//создаем нить, которая выполнит функцию threadFuncOdd с аргументами arg1

pthread\_create(&threadOdd, &attr, &threadFuncOdd, &arg1);

}

else{ //если чтение не удалось - значит достигли конца файла

isNotEndOdd = false;

} //считываем четную по порядку строку

if(getline(mainFile,str2)){

//присваиваем аргументы функции - строка и дескриптор файла

arg2 = argStruct(str2+"\n",fdEven);

//создаем нить, которая выполнит функцию threadFuncEven с аргументами arg2

pthread\_create(&threadEven, &attr, &threadFuncEven, &arg2);

}

else{ //если чтение не удалось - достигл конца файла

isNotEndEven = false;

}

//приостанавливаем основный процесс, пока потоки не завершатся

pthread\_join(threadOdd, NULL);

pthread\_join(threadEven, NULL);

}

//удаляем аттрибуты потоков

pthread\_attr\_destroy(&attr);

//закрываем все файла

mainFile.close();

close(fdOdd);

close(fdEven);

}