

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

ОТЧЁТ
по лабораторной работе №4
по дисциплине «Организация процессов и программирования в среде Linux»
Тема: УПРАВЛЕНИЕ ПОТОКАМИ

Студент гр. 9308

Преподаватель

Соболев М.С.

Разумовский Г.В.

Санкт-Петербург,

2022

Оглавление

| | |
|---|----|
| 1. Введение..... | 3 |
| 1.1. Введение..... | 3 |
| 1.2. Порядок выполнения работы..... | 3 |
| 1.3. Содержание отчёта..... | 4 |
| 2. Тексты программ, распечатка входных и выходных файлов..... | 5 |
| 2.1. main.cpp (часть 1)..... | 5 |
| 2.2. main.cpp (часть 2)..... | 10 |
| 2.3. Входной файл (часть 1)..... | 14 |
| 2.4. Входной файл (часть 2)..... | 16 |
| 2.5. Выходные файлы (часть 2)..... | 18 |
| 3. Скриншот экрана вывода файла для первой программы..... | 21 |
| 3.1. Компиляция..... | 21 |
| 3.2. Часть 1..... | 23 |
| 3.3. Часть 2..... | 26 |
| 3. Вывод..... | 27 |
| 4. Список использованных источников..... | 28 |

1. Введение

1.1. Введение

Тема работы: Управление потоками.

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

1.2. Порядок выполнения работы

1. Написать программу, которая открывает текстовый файл, порождает поток, а затем ожидает его завершения. Поток в качестве параметра передаётся 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 откомпилировать 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 нечётную строку, а другому – чётную. 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 прочтёт последнюю строку, после чего закрыть все файлы.

4. Откомпилировать программу и запустить её.

1.3. Содержание отчёта

Отчёт по лабораторной работе должен содержать:

1. Цель и задания.
2. Тексты программ, распечатку входных и выходных файлов.
3. Скриншот экрана вывода файла для первой программы.

2. Тексты программ, распечатка входных и выходных файлов

2.1. main.cpp (часть 1)

```
// !!! compile w/ flag "-pthread" !!!
// i.e. "g++ -Wall -pthread -o "%e" "%f""

#include <iostream>
#include <fstream>
#include <string>
#include <pthread.h>
#include <unistd.h>
#include <fcntl.h>

using namespace std;

void *ChildThread(void *arg);

int main()
{
    int handle_check; // handle check
    int handle; // handle
    int scheduling_policy; // scheduling policy field MAIN THREAD
    struct sched_param scheduling_parameters; // struct for schedual priority MAIN THREAD
    pthread_t thread;
    pthread_attr_t thread_attributes;

    cout << "----- MAIN THREAD BEGIN -----\\n";

    // ----- OPENING TEXT FILE -----

    handle = open("/home/matmanbj/lorem_ipsum.txt", O_RDONLY);

    if (handle == -1)
    {
        cout << "File's handle HAS NOT BEEN opened by PARENT thread!\\n";
    }
}
```

```

    return -1; // end program with error
}
else
{
    cout << "File's handle HAS BEEN opened by PARENT thread! Handle: " << handle << "\n";
}

// ----- CREATING A THREAD (SENDING HANDLE AS A PARAMETER) -----

pthread_attr_init(&thread_attributes);
pthread_create(&thread, &thread_attributes, ChildThread, &handle);
pthread_join(thread, nullptr);

// ----- CURRENT PRIORITY OUTPUT -----

cout << "----- MAIN THREAD INFO ----- \n";
pthread_getschedparam(pthread_self(), &scheduling_policy, &scheduling_parameters); // getting parameters about
MAIN thread
cout << "Current priority: " << scheduling_parameters.sched_priority << "\n"; // current priority

// ----- CHECKING ON OPEN/CLOSED FILE (IF OPENED -- FORCIBLY CLOSE) -----

handle_check = fcntl(handle, F_GETFD); // checking handle

if (handle_check != -1)
{
    cout << "File's handle HAS NOT BEEN closed by CHILD thread!\n";
    close (handle); // forced closing the file (handle)
    handle_check = fcntl(handle, F_GETFD); // checking handle again

    if (handle_check != -1)
    {
        cout << "File's handle HAS NOT BEEN closed by PARENT thread!\n";
        return -2; // end program with error
    }
    else

```

```

        {
            cout << "File's handle HAS BEEN closed by PARENT thread!\n";
        }
    }
    else
    {
        cout << "File's handle HAS BEEN closed by CHILD thread!\n";
    }

    cout << "----- MAIN THREAD END ----- \n";
    pthread_attr_destroy(&thread_attributes); // cleaning memory w/ destroying thread attributes no longer required

    return 0;
}

// ----- ChildThread function -----

void *ChildThread(void *arg)
{
    int local_handle_check; // handle check local
    int local_scheduling_policy; // scheduling policy field
    int local_close = 0; // 0 -- child thread DOES NOT CLOSES the handle, 1 -- child thread CLOSES the handle, other
-- DOES NOT CLOSE by default
    int local_buffer_counter = -1; // number of bytes read (-1 is for begin, 0 is for ending loop)
    int local_handle = *((int*)arg); // handle of opened file
    struct sched_param local_scheduling_parameters; // struct for schedul priority

    cout << "----- CHILD THREAD BEGIN ----- \n";

    cout << "Close the handle by child process (0 -- don't close, 1 -- close, other -- don't close)?\n"; // close handle by
child thread or not
    cin >> local_close; // user's choice

    pthread_getschedparam(pthread_self(), &local_scheduling_policy, &local_scheduling_parameters); // getting
parameters about CHILD thread

    // ----- SCHEDULING POLICY, CURRENT, MIN & MAX PRIORITY, FILE OUTPUT -----

```

```

cout << "----- CHILD THREAD INFO -----\\n"
<< "Scheduling policy: " // scheduling policy
<< (local_scheduling_policy == SCHED_FIFO ? to_string(local_scheduling_policy) + " -- SCHED_FIFO" : "")
<< (local_scheduling_policy == SCHED_RR ? to_string(local_scheduling_policy) + " -- SCHED_RR" : "")
<< (local_scheduling_policy == SCHED_OTHER ? to_string(local_scheduling_policy) + " -- SCHED_OTHER" :
"")) << "\\n"
<< "Current priority: " << local_scheduling_parameters.sched_priority << "\\n" // current priority
<< "Minimal priority: " << sched_get_priority_min(local_scheduling_policy) << "\\n" // minimal priority
<< "Maximal priority: " << sched_get_priority_max(local_scheduling_policy) << "\\n" // maximal priority
<< "File:\\n" // file output
<< "----- BEGIN OF FILE -----\\n";
    while(local_buffer_counter != 0) // 0 means no bytes to read
    {
        char local_buffer[80];
        size_t n = sizeof(local_buffer);
        local_buffer_counter = read(local_handle, &local_buffer, n);
        local_buffer[local_buffer_counter] = '\\0';
        cout << local_buffer;
    }
    cout << "\\n";
cout << "----- END OF FILE -----\\n";

if (local_close == 0) // handle close chosen actions
{
    cout << "File's handle SHOULD NOT BE closed by CHILD thread!\\n";
}
else if (local_close == 1)
{
    cout << "File's handle SHOULD BE closed by CHILD thread!\\n";
    close(local_handle);
}
else
{
    cout << "File's handle SHOULD NOT BE closed by CHILD thread by default!\\n";
}

local_handle_check = fcntl(local_handle, F_GETFD);

```



```
if (local_handle_check == -1) // handle close try check
{
    cout << "File's handle HAS BEEN closed by CHILD thread!\n";
}
else
{
    cout << "File's handle HAS NOT BEEN closed by CHILD thread!\n";
}

cout << "----- CHILD THREAD END ----- \n";
pthread_exit(NULL); // terminating calling thread, i.e. this CHILD process will be terminated
}
```

2.2. main.cpp (часть 2)

```
// !!! compile w/ flag "-pthread" !!!
// i.e. "g++ -Wall -pthread -o "%e" "%f""

#include <iostream>
#include <fstream>
#include <string>
#include <pthread.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>

using namespace std;

typedef struct // struct to give arguments (string & write file adress) to function
{
    string local_string;
    ofstream *write_file;
} funcArg;

void isFileOpenMyFunc (ifstream *local_file, const string local_file_name);
void isFileOpenMyFunc (ofstream *local_file, const string local_file_name);
void* threadFunction (void* main_argument);

// ----- MAIN -----

int main(int argc, char *argv[])
{
    pthread_t thread_1; // 1st thread to write odd strings (1, 3, 5...)
    pthread_t thread_2; // 2nd thread to write even strings (2, 4, 6...)
    bool is_file_end = false; // file read end indicator
    funcArg main_arguments_1;
    funcArg main_arguments_2;

    cout << "----- MAIN THREAD BEGIN -----\\n";

    cout << "----- FILE OPEN & CHECK -----\\n";
```

```

ifstream input_file("/home/matmanbj/lorem_ipsum_2.txt"); // open file to read
ofstream output_file_1("output_file_1.txt"); // open file to write 1
ofstream output_file_2("output_file_2.txt"); // open file to write 2


main_arguments_1.write_file = &output_file_1; // for struct
main_arguments_2.write_file = &output_file_2; // for struct


isFileOpenMyFunc (&input_file, "/home/matmanbj/lorem_ipsum_2.txt"); // check opening file to read
isFileOpenMyFunc (&output_file_1, "output_file_1.txt"); // check opening file to write 1
isFileOpenMyFunc (&output_file_2, "output_file_2.txt"); // check opening file to write 2


cout << "----- READ & WRITE BEGINS -----\\n";


while (is_file_end == false) // while file's end didn't reached, do R/W
{
    if(getline(input_file, main_arguments_1.local_string)) // if we can read 2n+1_th string, then put
characters into string
    {
        pthread_create(&thread_1, NULL, threadFunction, (void*)&main_arguments_1); // create
thread to write this string to file
    }
    else
    {
        is_file_end = true; // else indicate the loop to stop
    }


    if(getline(input_file, main_arguments_2.local_string)) // if we can read 2n_th string, then put
characters into string
    {
        pthread_create(&thread_2, NULL, threadFunction, (void*)&main_arguments_2); // create
thread to write this string to file
    }
    else
    {
        is_file_end = true; // else indicate the loop to stop
    }
}

```

```

        pthread_join(thread_1,NULL); // waiting for parallel threads termination
        pthread_join(thread_2,NULL); // waiting for parallel threads termination
    }

    cout << "----- READ & WRITE ENDS -----\\n";

    // close all opened files
    output_file_1.close();
    output_file_2.close();
    input_file.close();

    cout << "----- MAIN THREAD END -----\\n";

    return 0;
}

// ----- isFileOpen function -----

void isFileOpenMyFunc (ifstream *local_file, const string local_file_name) // check if the file open
{
    if ((*local_file).is_open()) //if (*local_file)
    {
        cout << "File to READ \\\n" << local_file_name << "\\n" HAS BEEN opened by PARENT thread!\\n";
    }
    else // if the file isn't opened, throw an error
    {
        cout << "File to READ \\\n" << local_file_name << "\\n" HAS NOT BEEN opened by PARENT thread!\\n"
        << "The program terminates w/ error!\\n";
        puts(("File to READ \\\n" + local_file_name + "\\n" HAS NOT BEEN opened by PARENT thread.
Continuation is impossible!").c_str());
    }
}

void isFileOpenMyFunc (ofstream *local_file, const string local_file_name) // check if the file open
{

```

```

if ((*local_file).is_open()) //if (*local_file)
{
    cout << "File to WRITE \"" << local_file_name << "\" HAS BEEN opened by PARENT thread!\n";
}
else // if the file isn't opened, throw an error
{
    cout << "File to WRITE \"" << local_file_name << "\" HAS NOT BEEN opened by PARENT thread!\n"
n"
    << "The program terminates w/ error!\n";
    puts(("File to WRITE \"" + local_file_name + "\" HAS NOT BEEN opened by PARENT thread.
Continuation is impossible!").c_str());
}
}

// ----- threadFunction function -----

void* threadFunction(void* main_argument) // write to file
{
    funcArg* local_argument = (funcArg*) main_argument;
    *(local_argument->write_file) << local_argument->local_string << "\n";
    pthread_exit(NULL);
}

```

2.3. Входной файл (часть 1)

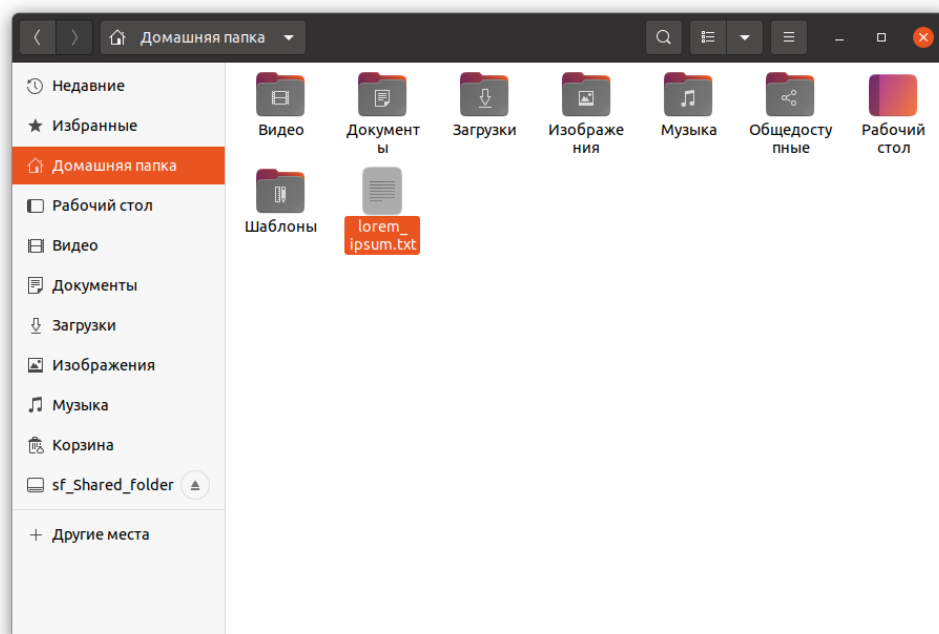


Рисунок 1. Входной файл для первой программы



Рисунок 2. Входной файл для первой программы

2.4. Входной файл (часть 2)

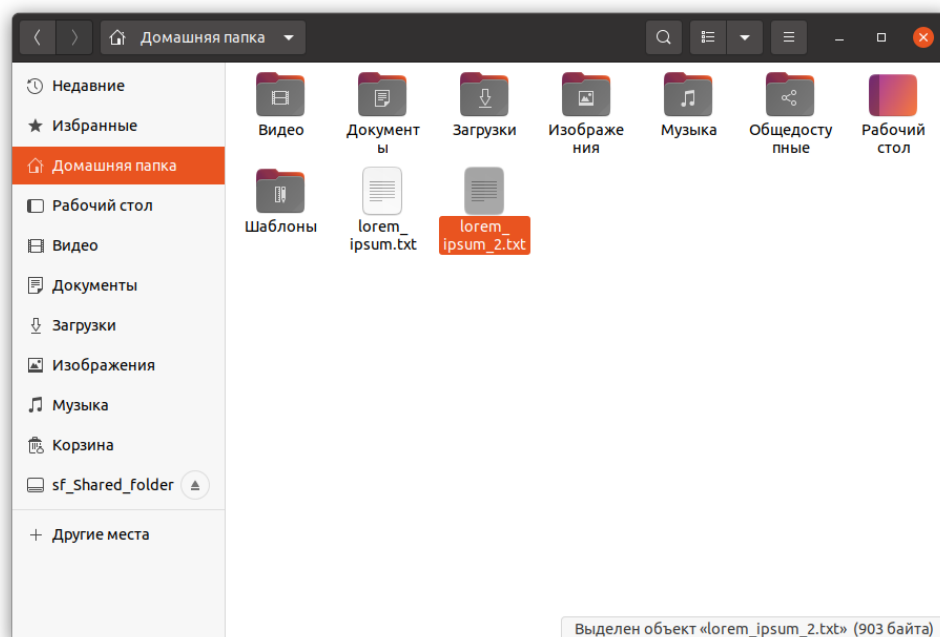


Рисунок 3. Входной файл для второй программы

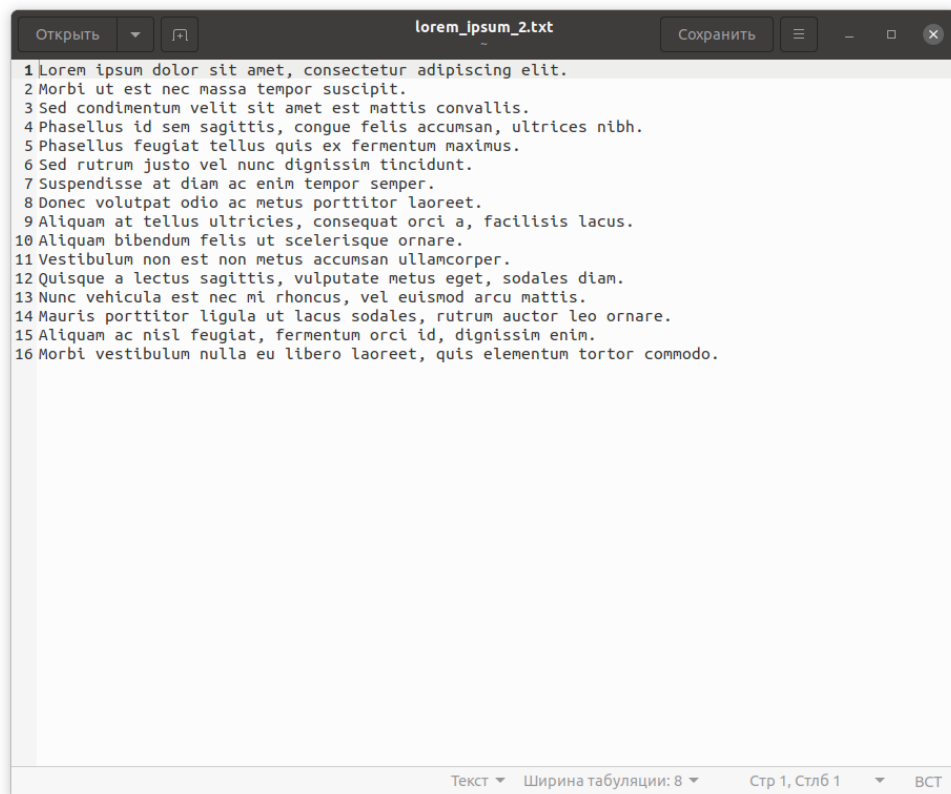


Рисунок 4. Входной файл для второй программы

2.5. Выходные файлы (часть 2)

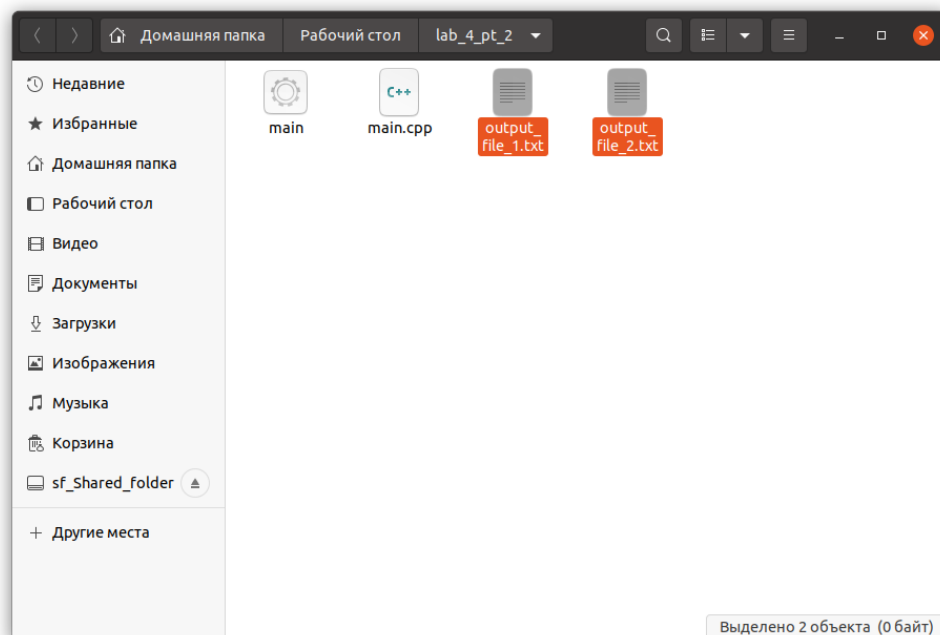


Рисунок 5. Выходные файлы для второй программы



Рисунок 6. Первый выходной файл для второй программы

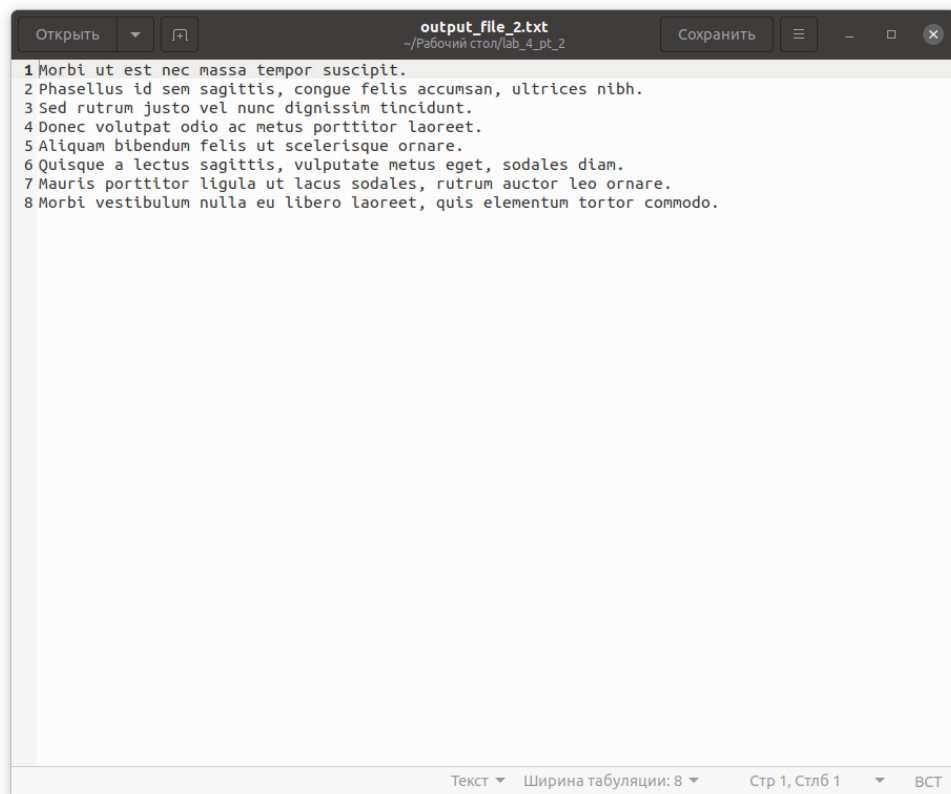
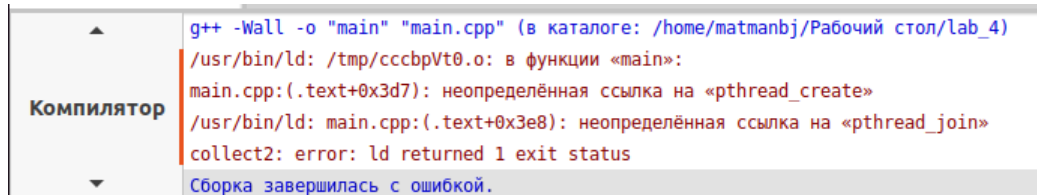


Рисунок 7. Второй выходной файл для второй программы

3. Скриншот экрана вывода файла для первой программы

3.1. Компиляция

Для запуска программы необходимо выставить специальный флаг «-pthread», который отвечает за сборку программы с функциями процессов. Иначе – будет ошибка сборки.



```
g++ -Wall -o "main" "main.cpp" (в каталоге: /home/matmanbj/Рабочий стол/lab_4)
/usr/bin/ld: /tmp/cccbpVt0.o: в функции «main»:
main.cpp:(.text+0x3d7): неопределённая ссылка на «pthread_create»
/usr/bin/ld: main.cpp:(.text+0x3e8): неопределённая ссылка на «pthread_join»
collect2: error: ld returned 1 exit status
Сборка завершилась с ошибкой.
```

Рисунок 8. Сборка с ошибкой при отсутствии флага «-pthread»

В IDE Geany были выставлены следующие флаги, для компиляции: «g++ -Wall -o "%e" "%f"»; для сборки: «g++ -Wall -pthread -o "%e" "%f"».

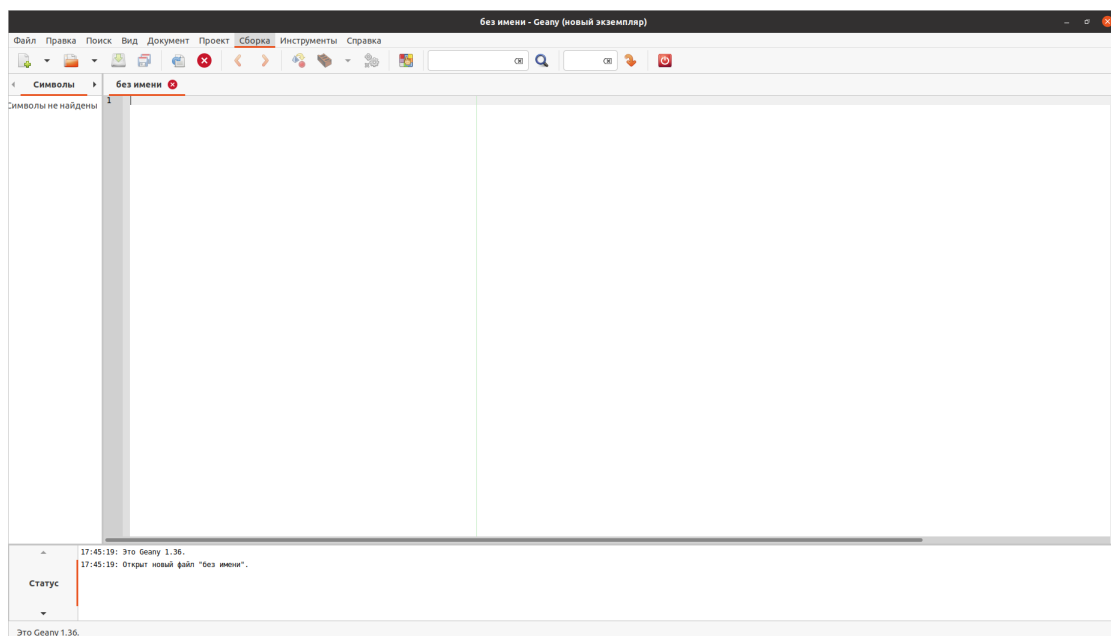


Рисунок 9. Установка флага «-pthread» в IDE «Geany»

Установить пользовательские команды

| # | Метка | Команда | Рабочий каталог | Сбросить |
|---|--------------------------|--|-----------------|----------------|
| Команды для языка: C++ | | | | |
| 1. | Compile | <code>g++ -Wall -c "%f"</code> | | |
| 2. | Build | <code>g++ -Wall -o "%e" "%f"</code> | | |
| 3. | Lint | <code>cppcheck --language=c++ --enable=warning,style --template=gcc "</code> | | |
| | | Регулярное выражение, являющееся признаком ошибки: | | |
| Независимые команды | | | | |
| 1. | Собрать | <code>make</code> | | |
| 2. | Собрать заданную цель... | <code>make</code> | | |
| 3. | Собрать объектный файл | <code>make %e.o</code> | | |
| 4. | | | | |
| | | Регулярное выражение, являющееся признаком ошибки: | | |
| <small>Примечание: для команды 2 будет открыт диалог, позволяющий ввести дополнительные параметры.</small> | | | | |
| Выполнить команды | | | | |
| 1. | Execute | <code>./%e</code> | | |
| 2. | | | | |
| <small>Шаблоны для подстановки в команду и рабочий каталог: %d, %e, %f, %r и %l, смотрите руководство для более детальной информации.</small> | | | | |
| | | | | Отменить ОК |

Рисунок 10. Установка флага «-pthread» в IDE «Geany»

Установить пользовательские команды

| # | Метка | Команда | Рабочий каталог | Сбросить |
|---|--------------------------|--|-----------------|----------------|
| Команды для языка: C++ | | | | |
| 1. | Compile | <code>g++ -Wall -c "%f"</code> | | |
| 2. | Build | <code>g++ -Wall -pthread -o "%e" "%f"</code> | | |
| 3. | Lint | <code>cppcheck --language=c++ --enable=warning,style --template=gcc "</code> | | |
| | | Регулярное выражение, являющееся признаком ошибки: | | |
| Независимые команды | | | | |
| 1. | Собрать | <code>make</code> | | |
| 2. | Собрать заданную цель... | <code>make</code> | | |
| 3. | Собрать объектный файл | <code>make %e.o</code> | | |
| 4. | | | | |
| | | Регулярное выражение, являющееся признаком ошибки: | | |
| <small>Примечание: для команды 2 будет открыт диалог, позволяющий ввести дополнительные параметры.</small> | | | | |
| Выполнить команды | | | | |
| 1. | Execute | <code>./%e</code> | | |
| 2. | | | | |
| <small>Шаблоны для подстановки в команду и рабочий каталог: %d, %e, %f, %r и %l, смотрите руководство для более детальной информации.</small> | | | | |
| | | | | Отменить ОК |

Рисунок 11. Установка флага «-pthread» в IDE «Geany»

▲

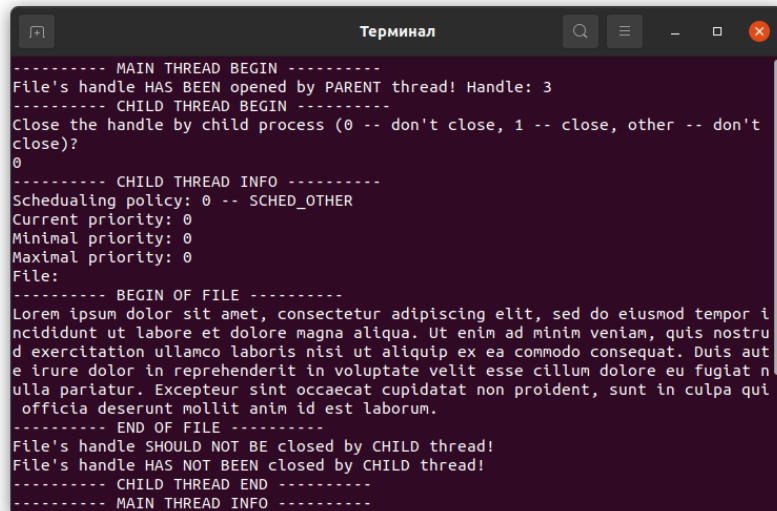
▼

Компилятор

```
g++ -Wall -pthread -o "main" "main.cpp" (в каталоге: /home/matmanbj/Рабочий стол/lab_4)
Сборка прошла успешно.
```

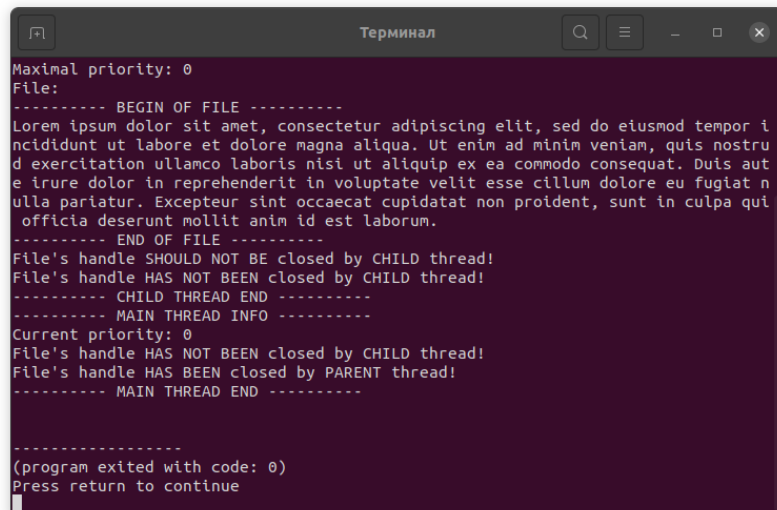
Рисунок 12. Сборка без ошибки при наличии флага «-pthread»

3.2. Часть 1



```
----- MAIN THREAD BEGIN -----
File's handle HAS BEEN opened by PARENT thread! Handle: 3
----- CHILD THREAD BEGIN -----
Close the handle by child process (0 -- don't close, 1 -- close, other -- don't
close)?
0
----- CHILD THREAD INFO -----
Scheduling policy: 0 -- SCHED_OTHER
Current priority: 0
Minimal priority: 0
Maximal priority: 0
File:
----- BEGIN OF FILE -----
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor i
ncidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostru
d exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aut
e irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat n
ulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui
officia deserunt mollit anim id est laborum.
----- END OF FILE -----
File's handle SHOULD NOT BE closed by CHILD thread!
File's handle HAS NOT BEEN closed by CHILD thread!
----- CHILD THREAD END -----
----- MAIN THREAD INFO -----
```

Рисунок 13. Запуск первой программы без закрытия дескриптора файла



```
Maximal priority: 0
File:
----- BEGIN OF FILE -----
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor i
ncidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostru
d exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aut
e irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat n
ulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui
officia deserunt mollit anim id est laborum.
----- END OF FILE -----
File's handle SHOULD NOT BE closed by CHILD thread!
File's handle HAS NOT BEEN closed by CHILD thread!
----- CHILD THREAD END -----
----- MAIN THREAD INFO -----
Current priority: 0
File's handle HAS NOT BEEN closed by CHILD thread!
File's handle HAS BEEN closed by PARENT thread!
----- MAIN THREAD END -----

(program exited with code: 0)
Press return to continue
```

Рисунок 14. Запуск первой программы без закрытия дескриптора файла

```
Терминал
----- MAIN THREAD BEGIN -----
File's handle HAS BEEN opened by PARENT thread! Handle: 3
----- CHILD THREAD BEGIN -----
Close the handle by child process (0 -- don't close, 1 -- close, other -- don't
close)?
1
----- CHILD THREAD INFO -----
Scheduling policy: 0 -- SCHED_OTHER
Current priority: 0
Minimal priority: 0
Maximal priority: 0
File:
----- BEGIN OF FILE -----
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor i
ncididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostru
d exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aut
e irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat n
ulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui
officia deserunt mollit anim id est laborum.
----- END OF FILE -----
File's handle SHOULD BE closed by CHILD thread!
File's handle HAS BEEN closed by CHILD thread!
----- CHILD THREAD END -----
----- MAIN THREAD INFO -----
```

Рисунок 15. Запуск первой программы с закрытием дескриптора файла

```
Терминал
Minimal priority: 0
Maximal priority: 0
File:
----- BEGIN OF FILE -----
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor i
ncididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostru
d exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aut
e irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat n
ulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui
officia deserunt mollit anim id est laborum.
----- END OF FILE -----
File's handle SHOULD BE closed by CHILD thread!
File's handle HAS BEEN closed by CHILD thread!
----- CHILD THREAD END -----
----- MAIN THREAD INFO -----
Current priority: 0
File's handle HAS BEEN closed by CHILD thread!
----- MAIN THREAD END -----

(program exited with code: 0)
Press return to continue
```

Рисунок 16. Запуск первой программы с закрытием дескриптора файла


```
Терминал
----- MAIN THREAD BEGIN -----
File's handle HAS BEEN opened by PARENT thread! Handle: 3
----- CHILD THREAD BEGIN -----
Close the handle by child process (0 -- don't close, 1 -- close, other -- don't
close)?
w
----- CHILD THREAD INFO -----
Scheduling policy: 0 -- SCHED_OTHER
Current priority: 0
Minimal priority: 0
Maximal priority: 0
File:
----- BEGIN OF FILE -----
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor i
ncididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostru
d exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aut
e irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat n
ulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui
officia deserunt mollit anim id est laborum.
----- END OF FILE -----
File's handle SHOULD NOT BE closed by CHILD thread!
File's handle HAS NOT BEEN closed by CHILD thread!
----- CHILD THREAD END -----
----- MAIN THREAD INFO -----
```

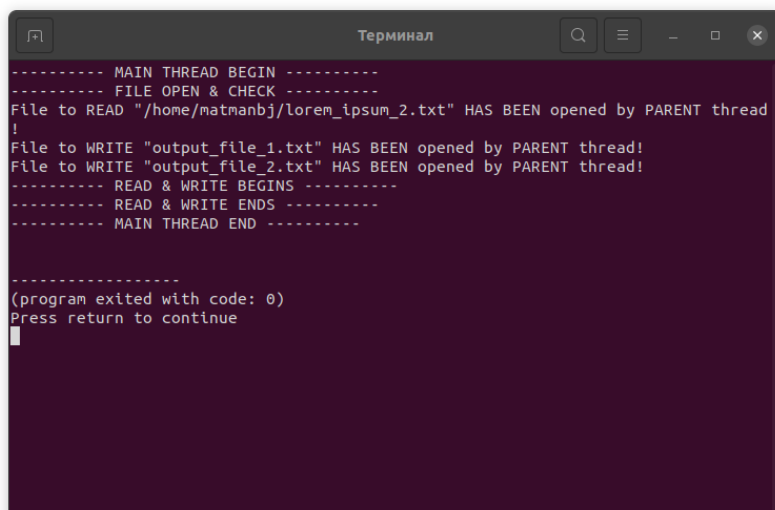
Рисунок 17. Запуск первой программы по умолчанию (без закрытия дескриптора файла)

```
Терминал
Maximal priority: 0
File:
----- BEGIN OF FILE -----
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor i
ncididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostru
d exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aut
e irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat n
ulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui
officia deserunt mollit anim id est laborum.
----- END OF FILE -----
File's handle SHOULD NOT BE closed by CHILD thread!
File's handle HAS NOT BEEN closed by CHILD thread!
----- CHILD THREAD END -----
----- MAIN THREAD INFO -----
Current priority: 0
File's handle HAS NOT BEEN closed by CHILD thread!
File's handle HAS BEEN closed by PARENT thread!
----- MAIN THREAD END -----

(program exited with code: 0)
Press return to continue
```

Рисунок 18. Запуск первой программы по умолчанию (без закрытия дескриптора файла)

3.3. Часть 2



```
Терминал
----- MAIN THREAD BEGIN -----
----- FILE OPEN & CHECK -----
File to READ "/home/matmanbj/lorem_ipsum_2.txt" HAS BEEN opened by PARENT thread
!
File to WRITE "output_file_1.txt" HAS BEEN opened by PARENT thread!
File to WRITE "output_file_2.txt" HAS BEEN opened by PARENT thread!
----- READ & WRITE BEGINS -----
----- READ & WRITE ENDS -----
----- MAIN THREAD END -----

-----
(program exited with code: 0)
Press return to continue
█
```

Рисунок 19. Запуск второй программы

3. Вывод

В ходе выполнения лабораторной работы №4 «Управление потоками» были изучены системные функции, позволяющие создавать потоки и управлять ими. Были написаны 2 программы: в первой программе дочерний поток осуществлял чтение файла по дескриптору, а затем по выбору пользователя закрывал его или оставлял открытым; во второй программе два дочерних потока осуществляли чтение файла, а затем записывали в новый, где один поток читал и записывал чётные строки, а другой – нечётные. Были использованы функции «pthread_create()» и «pthread_join()», которые создавали и ожидали поток соответственно. Таким образом и было произведено знакомство с организацией потоков и со способами синхронизации предков и потомков.

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.