МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Факультет автоматики и вычислительной техники

Кафедра автоматики



**ОТЧЁТ**

**по лабораторной работе №7**

***«****Многопоточное программирование****»***

по дисциплине: «Программирование»

Вариант № 2

Выполнили:Проверил:

студенты гр. АВТ-019 Нерлих Максим, с.п. Ядрышников О.Д.

Владислав Иманов, Владимир Рожнов

«\_\_\_» \_\_\_\_\_\_\_\_\_ 2021 г.«\_\_\_» \_\_\_\_\_\_\_\_\_ 2021 г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(оценка, подпись)

Новосибирск

2021

Оглавление

[Цель работы 3](#_Toc84443194)

[Методические указания 3](#_Toc84443195)

[Порядок выполнения работы 3](#_Toc84443196)

[Задание 3](#_Toc84443197)

[Исходный код 3](#_Toc84443198)

[Алгоритм работы 5](#_Toc84443199)

[Вывод 6](#_Toc84443200)

# Цель работы

Изучить способы и средства реализации параллельно выполняющихся потоков средствами стандарта POSIX.

# Методические указания

1. Проект может быть реализован на Visual C++ 6.0 или в среде Borland C++ 5.0 и выше. В первом случае выбирается консольное приложение Win32 без дополнительных библиотек. В любом случае в программу
2. Выбор функции для ожидания завершения порожденных процессов зависит от логики работы программы, определяемой вариантом задания. Уничтожение порожденных процессов применяется лишь в тех вариантах, где это действительно необходимо.
3. Для обмена информацией между процессами рекомендуется использовать аргументы командной строки и коды завершения процессов.
4. Примеры кода дочерней (file\_new.cpp) и родительской (spaces\_new.cpp) программ доступны по адресу: http://gun.cs.nstu.ru/ssw/API OC.

# Порядок выполнения работы

1. Переписать программу родительского процесса из лабораторной работы №3, запускающую потоки, реализующие функциональность программы из лабораторной работы №2 в количестве экземпляров, соответствующем количеству входных файлов, ждущую завершения всех потоков и получающую (выводящую в терминал) коды их завершения.

|  |  |  |
| --- | --- | --- |
| Вариант | Задание | Параметры командной строки |
| 2 | В конце каждой строки вставить заданный символ | 1. Имя входного файла 2. Заданный символ |

# Пункт 1.

# Исходный код

server/main.cpp

#include <iostream>

#include <vector>

#include <cstdlib>

#include <pthread.h>

#include "lib.h"

int main(int argc, char\* argv[]) {

const std::vector<std::string> filenames(argv + 1, argv + argc - 1);

const size\_t filesCount = filenames.size();

if (argc < 3) {

std::cerr << "too few arguments" << std::endl;

std::exit(-1);

}

const std::string replacer(argv[argc - 1]);

pthread\_t threads[filesCount];

for (uint8\_t i = 0; i < filesCount; i++) {

RoutineArguments\* args = new RoutineArguments;

args->filename = filenames.at(i);

args->replacer = replacer;

std::cout << "file: " << args->filename << std::endl;

if (pthread\_create(&threads[i], nullptr, appendFile, static\_cast<void\*>(args))) {

std::cerr << "failed to create thread" << std::endl;

std::exit(-1);

}

}

for (uint8\_t i = 0; i < filesCount; i++) {

int8\_t\* code;

pthread\_join(threads[i], reinterpret\_cast<void\*\*>(&code));

std::cout << "thread exited with code " << static\_cast<int>(\*code) << std::endl;

delete code;

}

pthread\_exit(nullptr);

}

server/lib.h

#include <string>

struct RoutineArguments {

std::string filename;

std::string replacer;

};

void\* appendFile(void\* arguments);

server/lib.cpp

#include <iostream>

#include <string>

#include <fstream>

#include <pthread.h>

#include "lib.h"

void\* appendFile(void\* arguments) {

RoutineArguments\* args = static\_cast<RoutineArguments\*>(arguments);

std::string filename(args->filename);

std::string replacer(args->replacer);

delete args;

std::cout << "from threads with file: " << filename << std::endl;

std::ifstream input(filename);

if (!input) {

std::cerr << "can't open " << filename << std::endl;

int8\_t\* code = new int8\_t(-1);

pthread\_exit(static\_cast<void\*>(code));

}

std::string outFilename = filename;

int dot = filename.find('.');

outFilename.replace(outFilename.begin() + dot, outFilename.end(), ".out");

std::ofstream output(outFilename, std::ios::app);

if (!output) {

std::cerr << "can't write to " << outFilename << std::endl;

int8\_t\* code = new int8\_t(-1);

pthread\_exit(static\_cast<void\*>(code));

}

while (input) {

std::string buff;

input >> buff;

if (!input) break;

output << buff << replacer << std::endl;

}

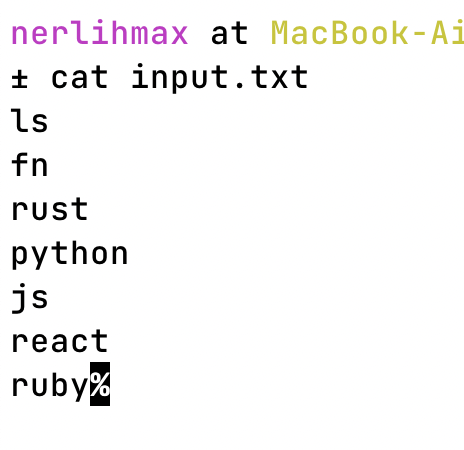
int8\_t\* code = new int8\_t(0);

pthread\_exit(static\_cast<void\*>(code));

}

# Алгоритм работы

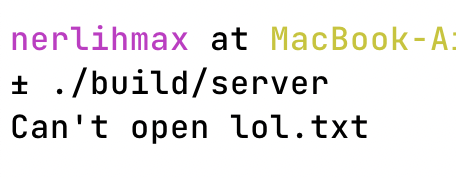
1. В папке с программой создаем текстовый документ input.txt с исходными данными:



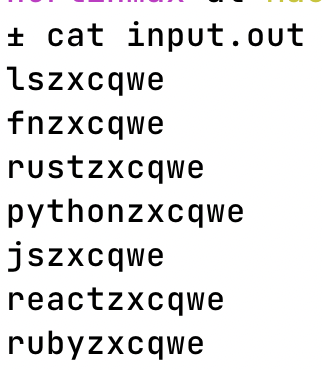
1. Запускаем программу-сервер следующим набором команд: 
2. Далее в другом шелле запускаем программу-клиент с нужными аргументами:
3. После отработки программы, вывод записывается в файл с таким же названием, как и входной файл, но с расширением out, а также сообщит в консоль о завершении дочернего процесса, который выполнял манипуляции.

В результате программа добавила к концу каждой строки заданный символ

1. Если программа не смогла открыть файл с исходными данными, выводится ошибка:



1. Если выходной файл уже существует, программа создаст и заменит прежний:



# Вывод

В результате работы над лабораторной работой изучили особенности написания многопоточных программ с использованием потоков POSIX.