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 №6**

***«****Межпроцессорные коммуникации. Семафоры. Отображаемые на память файлы****»***

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 обмена информацией между процессами с использованием отображаемых на память файлов и семафоров в качестве средства синхронизации в 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. Модифицировать и отладить программу из лабораторной работы 2, реализующую серверный процесс, ожидающий открытие семафор, для приема параметров не через аргументы командной строки, а через файл, отображенный на память, а также реализовать клиентскую программу, которая будет писать в файл параметры командной и открывать семафор.

|  |  |  |
| --- | --- | --- |
| 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 <cstdlib>

#include <thread>

#include <chrono>

#include <unistd.h>

#include <semaphore.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <sys/types.h>

#include <sys/stat.h>

#include "lib.h"

const std::string semaphoreName("/gunlab-sem");

const std::string sharedFile("/tmp/gunlab-sf");

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

int file = open(sharedFile.c\_str(), O\_RDWR | O\_CREAT, S\_IRUSR | S\_IWUSR);

if (file == -1) {

std::cerr << "could not open shared file" << std::endl;

std::exit(-1);

}

if (ftruncate(file, 128) == -1) {

std::cerr << "could not truncate file" << std::endl;

close(file);

std::exit(-1);

}

void\* shared = mmap(0, 128, PROT\_WRITE | PROT\_READ, MAP\_SHARED, file, 0);

if (shared == MAP\_FAILED) {

std::cerr << "could not map file on memory" << std::endl;

close(file);

std::exit(-1);

}

if (close(file) == -1) {

std::cerr << "could not close shared file" << std::endl;

std::exit(-1);

munmap(shared, 128);

}

sem\_t\* sem = sem\_open(semaphoreName.c\_str(), O\_CREAT, 0777, 0);

if (sem == SEM\_FAILED) {

std::cerr << "could not open semaphore" << std::endl;

munmap(shared, 128);

std::exit(-1);

}

sem\_wait(sem);

std::string message((char\*) shared);

std::cout << message << std::endl;

std::size\_t delimer = message.find(' ');

std::string filename = message.substr(0, delimer);

std::string replacer = message.substr(delimer + 1);

if (appendFile(filename, replacer) == -1) {

std::exit(-1);

}

sem\_unlink(semaphoreName.c\_str());

munmap(shared, 128);

return 0;

}

server/lib.h

#include <string>

int appendFile(std::string filename, std::string symbol);

server/lib.cpp

#include <iostream>

#include <string>

#include <fstream>

int appendFile(std::string filename, std::string symbol) {

std::ifstream input(filename);

if (!input) {

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

return -1;

}

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;

return -1;

}

while (input) {

std::string buff;

input >> buff;

if (!input) break;

output << buff << symbol << std::endl;

}

return 0;

}

client/main.cpp

#include <iostream>

#include <string>

#include <vector>

#include <sstream>

#include <unistd.h>

#include <semaphore.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <sys/types.h>

#include <sys/stat.h>

const std::string semaphoreName("/gunlab-sem");

const std::string sharedFile("/tmp/gunlab-sf");

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

const std::vector<std::string> params(argv, argv + argc);

if (params.size() < 3) {

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

std::exit(-1);

}

const std::string filename = params.at(1);

const std::string replacer = params.at(2);

int file = open(sharedFile.c\_str(), O\_RDWR, S\_IRUSR | S\_IWUSR);

if (file == -1) {

std::cerr << "could not open shared file" << std::endl;

std::exit(-1);

}

void\* shared = mmap(0, 128, PROT\_WRITE | PROT\_READ, MAP\_SHARED, file, 0);

if (shared == MAP\_FAILED) {

std::cerr << "could not map file on memory" << std::endl;

std::exit(-1);

}

if (close(file) == -1) {

std::cerr << "could not close shared file" << std::endl;

std::exit(-1);

}

sem\_t\* sem = sem\_open(semaphoreName.c\_str(), O\_RDWR);

if (sem == SEM\_FAILED) {

std::cerr << "could not open semaphore" << std::endl;

std::exit(-1);

}

std::stringstream ss;

ss << filename << " " << replacer;

std::string message = ss.str();

char buff[128];

buff[message.copy(buff, message.length())] = '\0';

message.copy((char\*) shared, message.size());

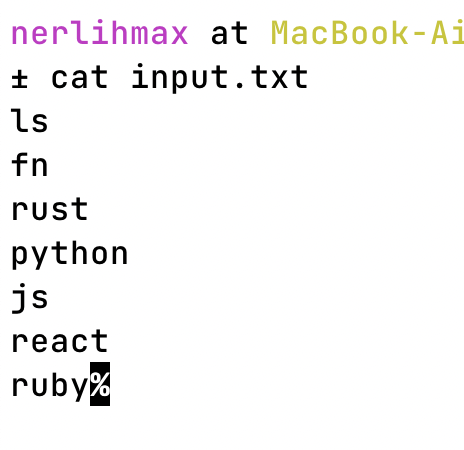
sem\_post(sem);

return 0;

}

# Алгоритм работы

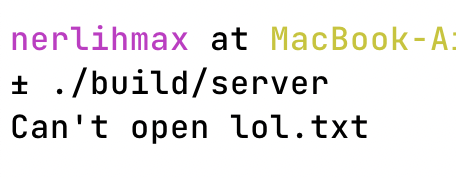
1. В папке с программой создаем текстовый документ input.txt с исходными данными:



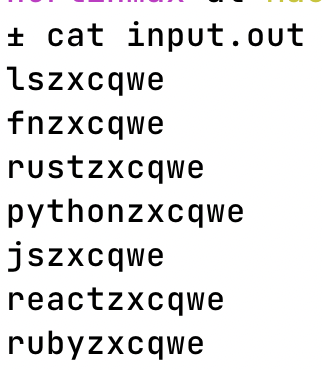
1. Запускаем программу-сервер следующим набором команд: 
2. Далее в другом шелле запускаем программу-клиент с нужными аргументами:
3. После отработки программы, вывод записывается в файл с таким же названием, как и входной файл, но с расширением out, а также сообщит в консоль о завершении дочернего процесса, 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 программ с использованием файлов отображенных на память, а также использование семафоров в качестве средства синхронизации процессов.