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 №4**

***«****Межпроцессорные коммуникации. Именованные каналы****»***

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 именованных каналов 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 порожденный процесс – клиентское приложение, для приема параметров не через аргументы командной строки, а через канал.
2. Модифицировать и отладить программу из лабораторной работы 3, реализующую родительский процесс, вызывающий и отслеживающий состояние порожденных процессов – клиентов, передающий им параметры и получающий результаты выполнения порожденных процессов через канал.

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

# Пункт 1.

# Исходный код

client/main.cpp

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

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

const std::string pipePath("/tmp/gunlab");

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

if (params.size() < 3) {

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

std::terminate();

}

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

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

std::ofstream pipe(pipePath);

if (!pipe) {

std::cerr << "failed to open pipe" << std::endl;

std::terminate();

}

pipe << filename << " " << symbol;

pipe.close();

return 0;

}

server/lib.h

#include <string>

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

server/lib.cpp

#include <iostream>

#include <string>

#include <fstream>

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

std::ifstream input(filename);

if (!input) {

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

std::terminate();

}

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;

}

while (input) {

std::string buff;

input >> buff;

if (!input) break;

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

}

}

server/main.cpp

#include <iostream>

#include <fstream>

#include <filesystem>

#include <exception>

#include <sys/stat.h>

#include "lib.h"

bool checkFileExisting(std::string path) {

return std::filesystem::exists(path);

}

void deleteFile(std::string path) {

try {

std::filesystem::remove(path);

} catch (std::filesystem::filesystem\_error err) {

std::cerr << "file deleting failed" << std::endl;

std::terminate();

}

}

bool makeFifo(std::string path) {

return mkfifo(path.c\_str(), 0777);

}

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

const std::string pipePath("/tmp/gunlab");

if (checkFileExisting(pipePath)) {

deleteFile(pipePath);

}

if (makeFifo(pipePath) != 0) {

std::cerr << "creating pipe failed" << std::endl;

std::terminate();

}

std::ifstream pipe(pipePath);

if (!pipe) {

std::cerr << "opening pipe failed" << std::endl;

std::terminate();

}

std::string filename;

std::string replacer;

pipe >> filename >> replacer;

appendFile(filename, replacer);

return 0;

}

Makefile

CXC = clang++

CARGS = -g -std=c++17

all: mkdir lib server build/client

mkdir:

mkdir -pv build

lib: server/lib.cpp

$(CXC) $(CARGS) -dynamiclib -o build/liblab.dylib server/lib.cpp

server: lib server/main.cpp

$(CXC) $(CARGS) -o build/server -L./build -llab server/main.cpp

build/client: client/main.cpp

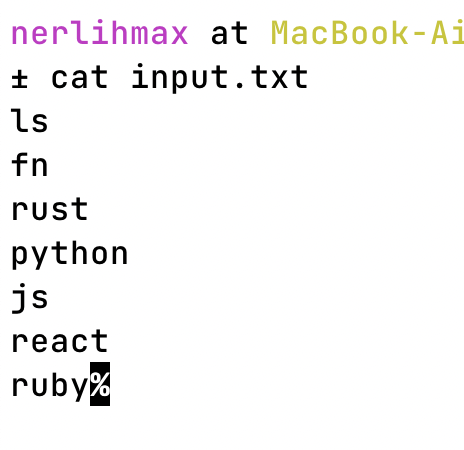
$(CXC) $(CARGS) -o build/client client/main.cpp

clean:

rm -rf build

# Алгоритм работы

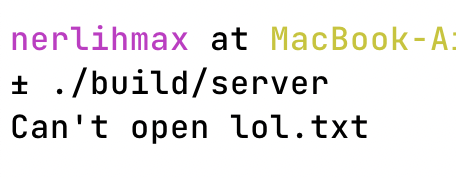
1. В папке с программой создаем текстовый документ input.txt с исходными данными:



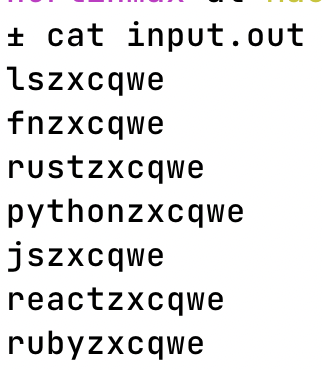
1. Запускаем программу-сервер следующим набором команд: 
2. Далее в другом шелле запускаем программу-клиент с нужными аргументами:
3. После отработки программы, вывод записывается в файл с таким же названием, как и входной файл, но с расширением out, а также сообщит в консоль о завершении дочернего процесса,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



# Пункт 2.

# Исходный код

client/main.cpp

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <filesystem>

#include <thread>

#include <chrono>

#include <unistd.h>

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

const std::string pipePath("/tmp/gunlab");

const std::string serverPath("./build/server");

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

if (params.size() < 3) {

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

std::terminate();

}

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

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

pid\_t pid = fork();

if (pid == -1) {

std::cerr << "failed to create child process" << std::endl;

std::terminate();

}

if (pid > 0) {

// parent process

while (!std::filesystem::exists(pipePath)) {

std::this\_thread::sleep\_for(std::chrono::milliseconds(100));

}

std::ofstream pipe(pipePath);

if (!pipe) {

std::cerr << "failed to open pipe" << std::endl;

std::terminate();

}

pipe << filename << " " << symbol;

pipe.close();

waitpid(pid, nullptr, 0);

} else {

// child process

if (execlp(serverPath.c\_str(), serverPath.c\_str(), NULL) == -1) {

std::cerr << "failed to exec server" << std::endl;

std::terminate();

}

}

return 0;

}

server/lib.h

#include <string>

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

server/lib.cpp

#include <iostream>

#include <string>

#include <fstream>

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

std::ifstream input(filename);

if (!input) {

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

std::terminate();

}

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;

}

while (input) {

std::string buff;

input >> buff;

if (!input) break;

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

}

}

server/main.cpp

#include <iostream>

#include <fstream>

#include <filesystem>

#include <exception>

#include <sys/stat.h>

#include "lib.h"

bool checkFileExisting(std::string path) {

return std::filesystem::exists(path);

}

void deleteFile(std::string path) {

try {

std::filesystem::remove(path);

} catch (std::filesystem::filesystem\_error err) {

std::cerr << "file deleting failed" << std::endl;

std::terminate();

}

}

bool makeFifo(std::string path) {

return mkfifo(path.c\_str(), 0777);

}

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

const std::string pipePath("/tmp/gunlab");

if (checkFileExisting(pipePath)) {

deleteFile(pipePath);

}

if (makeFifo(pipePath) != 0) {

std::cerr << "creating pipe failed" << std::endl;

std::terminate();

}

std::ifstream pipe(pipePath);

if (!pipe) {

std::cerr << "opening pipe failed" << std::endl;

std::terminate();

}

std::string filename;

std::string replacer;

pipe >> filename >> replacer;

pipe.close();

appendFile(filename, replacer);

if (!std::filesystem::remove(pipePath)) {

std::cerr << "failed to delete pipe" << std::endl;

std::terminate();

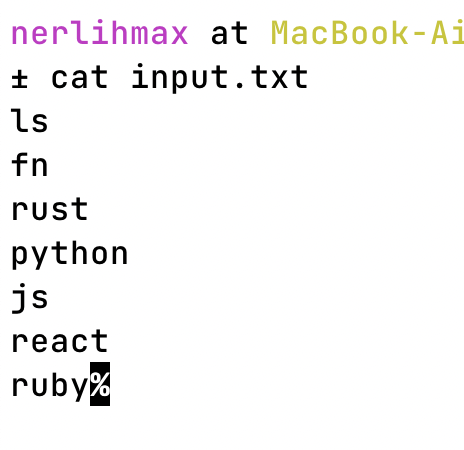
}

return 0;

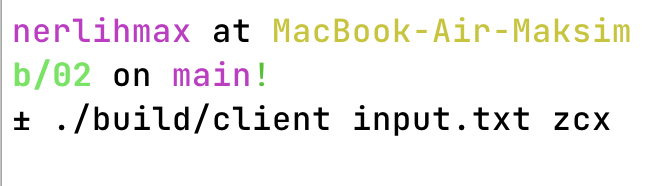
}

# Алгоритм работы

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

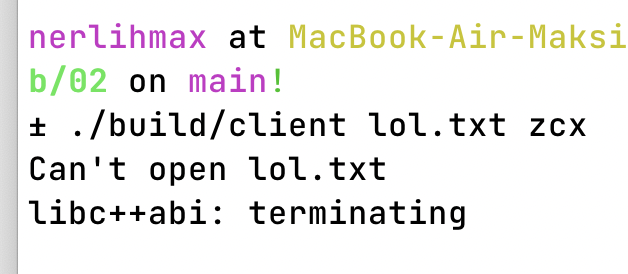


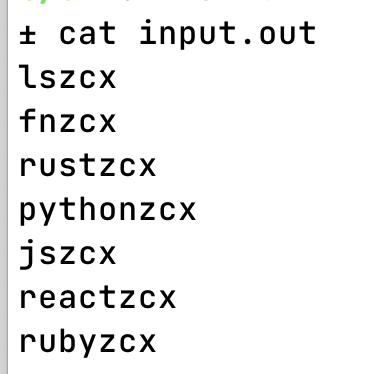
1. Запускаем программу следующим набором команд:



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



# Вывод

В результате работы над лабораторной работой изучили особенности написания программ с многозадачностью путем использования дополнительных дочерних процессов, а также передачи информации между процессами с использованием именованных каналов.