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 №5**

***«****Межпроцессорные коммуникации. Сокеты****»***

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 UNIX-сокетов 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 подключение клиента, для приема параметров не через аргументы командной строки, а через сокет, а также реализовать клиентскую программу, которая будет подключаться по сокету к серверу и передавать ему аргументы командой строки

|  |  |  |
| --- | --- | --- |
| 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 <fstream>

#include <string>

#include <vector>

#include <sstream>

#include <unistd.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/un.h>

#include "lib.h"

const std::string socketPath("/var/run/gunlab");

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

int sock = socket(AF\_UNIX, SOCK\_STREAM, 0);

if (sock == -1) {

std::cerr << "could not create socket" << std::endl;

std::exit(-1);

}

sockaddr\_un addr;

{

addr.sun\_family = AF\_UNIX;

char pathBuff[104];

std::size\_t copied = socketPath.copy(addr.sun\_path, socketPath.size());

pathBuff[copied] = '\0';

addr.sun\_len = copied;

}

if (bind(sock, (struct sockaddr \*) &addr, sizeof(addr)) == -1) {

std::cerr << "could not bind socket" << std::endl;

std::exit(-1);

}

if (listen(sock, 1) == -1) {

std::cerr << "could not open socket to listening" << std::endl;

std::exit(-1);

}

int connection = accept(sock, nullptr, nullptr);

if (connection == -1) {

std::cerr << "could not accept incoming connection" << std::endl;

std::exit(-1);

}

char buff[128];

if (read(connection, buff, 128) == -1) {

std::cerr << "failed to read message" << std::endl;

close(connection);

close(sock);

unlink(socketPath.c\_str());

std::exit(-1);

}

const std::string message(buff);

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) {

close(connection);

close(sock);

unlink(socketPath.c\_str());

std::exit(-1);

}

close(connection);

close(sock);

unlink(socketPath.c\_str());

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 <cstdio>

#include <unistd.h>

#include <sys/socket.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <sys/un.h>

const std::string socketPath("/var/run/gunlab");

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::exit(-1);

}

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

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

int sock = socket(AF\_UNIX, SOCK\_STREAM, 0);

if (sock == -1) {

std::cerr << "could not create socket" << std::endl;

std::exit(-1);

}

sockaddr\_un addr;

{

addr.sun\_family = AF\_UNIX;

char pathBuff[104];

std::size\_t copied = socketPath.copy(addr.sun\_path, socketPath.size());

pathBuff[copied] = '\0';

addr.sun\_len = copied;

}

if (connect(sock, (struct sockaddr \*) &addr, sizeof(addr)) == -1) {

std::cerr << "could not connect" << std::endl;

std::exit(-1);

}

std::stringstream ss;

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

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

char buff[128];

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

if (write(sock, buff, 128) == -1) {

std::cerr << "failed to write message" << std::endl;

close(sock);

std::exit(-1);

}

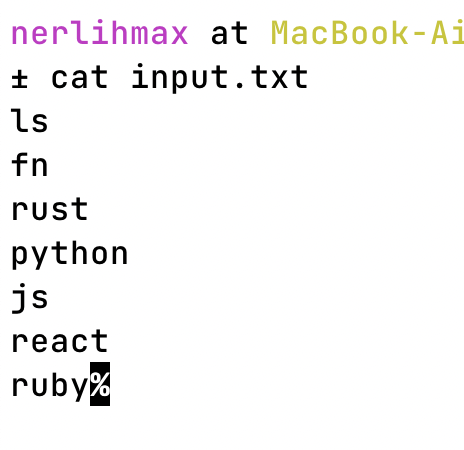
close(sock);

return 0;

}

# Алгоритм работы

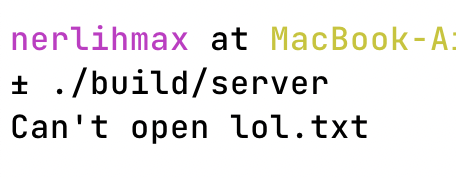
1. В папке с программой создаем текстовый документ input.txt с исходными данными:



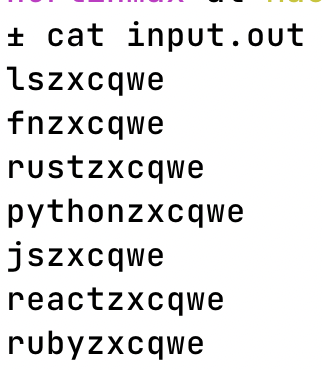
1. Запускаем программу-сервер следующим набором команд: 
2. Далее в другом шелле запускаем программу-клиент с нужными аргументами:
3. После отработки программы, вывод записывается в файл с таким же названием, как и входной файл, но с расширением out, а также сообщит в консоль о завершении дочернего процесса,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