Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект по курсу**

**«Операционные системы»**

**Тема работы**

**“Клиент-серверная система для передачи мгновенных сообщений”**

Студент: Клитная Анастасия Викторовна

Группа: М8О-208Б-20

Вариант: 28

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

https://github.com/silverfatt

**Постановка задачи**

Необходимо спроектировать и реализовать программный прототип в соответствии с выбранным вариантом. Произвести анализ и сделать вывод на основании данных, полученных при работе программного прототипа.

Базовый функционал должен быть следующим:

• Клиент может присоединиться к серверу, введя логин

• Клиент может отправить сообщение другому клиенту по его логину

• Клиент в реальном времени принимает сообщения от других клиентов

28. Необходимо предусмотреть возможность отправки отложенного сообщения или себе, или другому пользователю. При выходе с сервера отправка всё равно должна осуществиться. Связь между сервером и клиентом должна быть реализована при помощи pipe'ов

**Общие сведения о программе**

Программа состоит из трёх файлов – server.cpp, client.cpp, funcs.cpp, в которых расположены код сервера, код клиента, реализация вспомогательный функций соответственно. Для удобства также был создан Makefile.

**Общий метод и алгоритм решения**

Общение между клиентом и сервером осуществляется как на схеме, изображённой ниже:

Для начала необходимо запустить сервер и «зарегистрировать» пользователей (ввести список допустимых логинов, которые сервер сохранит для дальнейшего использования). После окончания ввода логинов сервер создаст именованный pipe “input” – для приёма сигналов от клиентов (все клиенты будут писать в один pipe), а также по одному pipe для каждого логина (сервер будет отвечать каждому клиенту в его персональный pipe, откуда тот и будет читать информацию).

Когда подготовка и настройка сервера завершена, можно запустить клиент и ввести логин. Если логин зарегистрирован на сервере – можно будет начинать работу, иначе программа выдаст ошибку и предложит ещё раз ввести логин.

В клиентской программе предусмотрен дополнительный поток. В основном потоке осуществляется вод и отправление сообщений серверу, в дополнительном – получение сообщений от сервера и вывод их на экран. Для отправки сообщения необходимо ввести в терминал команду вида «логин сообщение». Если логин существует – другой пользователь получит сообщение. Иначе сервер сообщит об ошибке.

Черновые сообщения отправляться не будут, пока клиент не захочет их отправить. Для этого у него будет отдельная команда. При отправке сообщения клиента попросят ввести логин получателя и тогда все сообщения черновика будут пересылаться по данному адресу.

**Исходный код**

**Server.cpp**

#include <iostream>

#include <stdlib.h>

#include <unistd.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <errno.h>

#include <vector>

#include <fcntl.h>

#include "funcs.hpp"

#include <map>

#include <vector>

int in(std::vector<std::string> logins, std::string str)

{

for (int i = 0; i < logins.size(); ++i)

{

if (logins[i] == str)

return i;

}

return -1;

}

int main()

{

// std::vector<std::vector<std::string>> draft;

std::vector<std::string> logins;

std::string command;

std::string login;

//ввод логинов

std::cout << "Enter all user's logins. Insert 'end' to stop:\n";

while (login != "end")

{

std::cin >> login;

//std::vector<std::string> vec;

//vec.push\_back(login);

//draft.push\_back(vec);

if (in(logins, login) == -1){

logins.push\_back(login);

}

else

std::cout << "already exists!";

}

//создание выходных FIFO для всех логинов

for (int i = 0; i < logins.size(); ++i)

{

if (mkfifo(logins[i].c\_str(), 0777) == -1)

{

if (errno != EEXIST)

{

std::cout << "FIFO WAS NOT CREATED";

exit(1);

}

}

}

//создание входного FIFO

if (mkfifo("input", 0777) == -1)

{

std::cout << "MAIN INPUT FIFO WAS NOT CREATED";

exit(1);

}

int fd\_recv = open("input", O\_RDWR);

if (fd\_recv == -1)

{

std::cout << "INPUT FIFO WAS NOT OPENED";

exit(1);

}

//открытие всех FIFO на запись

int fd[logins.size()];

for (int i = 0; i < logins.size(); ++i)

{

fd[i] = open(logins[i].c\_str(), O\_RDWR);

}

while (1)

{

std::string message;

message = recieve\_message\_server(fd\_recv);

std::cout << message;

std::string my\_log = extract\_login(message); //от кого

std::string rec\_log = extract\_addressee(message); //кому

std::string rcvd\_message = extract\_message(message); //что

int repl\_fd = in(logins, rec\_log); //id получателя

int fd\_usr = in(logins, my\_log); //id отправителя

int pos = -1;

if (in(logins, rec\_log) == -1)

{

send\_message\_to\_client(fd[fd\_usr], "Login does not exists!\n");

}

else

{

send\_message\_to\_client(fd[repl\_fd], rcvd\_message);

}

}

}

**Funcs.hpp**

#include <string>

//отправить сообщение серверу в удобной форме - логин$получатель$сообщение

void send\_message\_to\_server(int fd, std::string curlogin, std::string user, std::string message)

{

std::string text = curlogin + "$" + user + "$" + message;

int k = text.size();

write(fd, &k, sizeof(k));

char messagec[k];

for (int i = 0; i < k; ++i)

{

messagec[i] = text[i];

}

write(fd, messagec, k);

}

//отправить сообщение клиенту

void send\_message\_to\_client(int fd, std::string message)

{

std::string text = message;

int k = text.size();

write(fd, &k, sizeof(k));

char messagec[k];

for (int i = 0; i < k; ++i)

{

messagec[i] = text[i];

}

write(fd, messagec, k);

}

//получить сообщение в удобной для клиента форме

std::string recieve\_message\_client(int fd)

{

int size;

read(fd, &size, sizeof(size));

char messagec[size];

read(fd, messagec, size);

std::string recv;

for (int i = 0; i < size; ++i)

{

if (messagec[i] != '$')

{

recv.push\_back(messagec[i]);

}

else

{

recv = recv + ": ";

}

}

return recv;

}

//получить сообщение в удобной для сервера форме

std::string recieve\_message\_server(int fd)

{

int size;

read(fd, &size, sizeof(size));

char messagec[size];

read(fd, messagec, size);

std::string recv;

for (int i = 0; i < size; ++i)

{

recv.push\_back(messagec[i]);

}

return recv;

}

//получить логин из сообщения для сервера

std::string extract\_login(std::string message)

{

std::string login;

int i = 0;

while (message[i] != '$')

{

login.push\_back(message[i]);

++i;

}

return login;

}

//получить сообщение для клиента

std::string extract\_message(std::string message)

{

std::string text, text1, text2;

int i = 0;

while (message[i] != '$')

{

text1.push\_back(message[i]);

++i;

}

++i;

while (message[i] != '$')

{

++i;

}

while (i < message.size())

{

text2.push\_back(message[i]);

++i;

//std::cout << "TESTSSSS";

}

text = text1 + text2;

return text;

}

//получить получателя сообщения

std::string extract\_addressee(std::string message)

{

std::string text;

int i = 0;

while (message[i] != '$')

{

//login.push\_back(message[i]);

++i;

}

++i;

while (message[i] != '$')

{

text.push\_back(message[i]);

++i;

//std::cout << "TESTSSSS";

}

return text;

}

//получить текст сообщения

std::string extract\_text(std::string message)

{

std::string text;

int i = 0;

while (message[i] != '$')

{

//login.push\_back(message[i]);

++i;

}

++i;

while (message[i] != '$')

{

//login.push\_back(message[i]);

++i;

}

++i;

++i;

while (i < message.size())

{

text.push\_back(message[i]);

++i;

//std::cout << "TESTSSSS";

}

return text;

}

//обычный поиск подстроки

bool search(std::string text, std::string pattern)

{

if (pattern.size() <= text.size())

{

//cout << text << " " << pattern << "\n";

for (int i = 0; i <= text.size() - pattern.size(); ++i)

{

std::string pat;

for (int z = 0; z < pattern.size(); ++z)

{

if (text[i + z] == pattern[z])

pat.push\_back(text[i + z]);

}

if (pat == pattern)

{

return true;

}

pat.clear();

}

}

return false;

}

**Client.cpp**

#include <iostream>

#include <stdlib.h>

#include <unistd.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <errno.h>

#include <vector>

#include <fcntl.h>

#include "funcs.hpp"

#include <thread>

//функция приёма сообщений (для потока)

void func(int fd\_recv, std::string login)

{

while (1)

{

std::string reply = recieve\_message\_client(fd\_recv);

std::cout << reply << "\n";

std::cout.flush();

std::cout << login << ">";

std::cout.flush();

}

}

int main()

{

//подключение к входному FIFO сервера

int fd\_send = open("input", O\_RDWR);

if (fd\_send == -1)

{

std::cout << "ERROR: MAIN FIFO WAS NOT OPENED\n";

exit(1);

}

//подготовка - инструкции, ввод логина

std::cout << "Wellcome to VMAI.\nTo create accounts launch ./server and insert logins.\n Than relaunch this application and enter your login.\n";

std::cout << "Input: login message. Example: anton\n hey, how are you?\n";

std::cout << "Insert your login: ";

std::string login;

std::vector<std::string> drafts;

//подключение к персональному именованному пайпу

int fd\_recv = -1;

while (fd\_recv == -1)

{

std::cin >> login;

fd\_recv = open(login.c\_str(), O\_RDWR);

if (fd\_recv == -1){

std::cout << "Wrong login!\nInsert your login: ";

}

};

//вход успешен, запуск потока принятия сообщений от сервера

std::string adressee, message;

std::cout << "Congrats! You have signed. Now you can send messages!\n";

std::thread thr\_recieve(func, fd\_recv, login);

//запуск цикла отправки сообщений на сервер

while (1){

std::cout << login << "> ";

std::cin >> adressee;

if (adressee == "draft"){

std::string mesage;

std::getline(std::cin, mesage);

drafts.push\_back(mesage);

}

else{

if (adressee == "send\_draft"){

std::string recipient;

std::getline(std::cin,recipient);

for(int i = 0; i < drafts.size(); i++){

send\_message\_to\_server(fd\_send, login, recipient, drafts[i]);

}

}

else{

if (adressee == "quit")

break;

std::getline(std::cin, message);

send\_message\_to\_server(fd\_send, login, adressee, message);

}

}}

thr\_recieve.detach();

}

**Выводы**Данный курсовой проект оказался довольно интересным. Я закрепила навыки использования pipe, в целом узнал больше о межпроцессовом взаимодействии, закрепил навыки работы со строками в С++. А также разобралась с логикой отправки и обработки сообщений серверами.