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: Меджидли Махмуд Ибрагим оглы

Группа: М8О-208Б-20

Вариант: 25

Преподаватель: Миронов Евгений Сергеевич

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/loshadkaigogo/OS

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

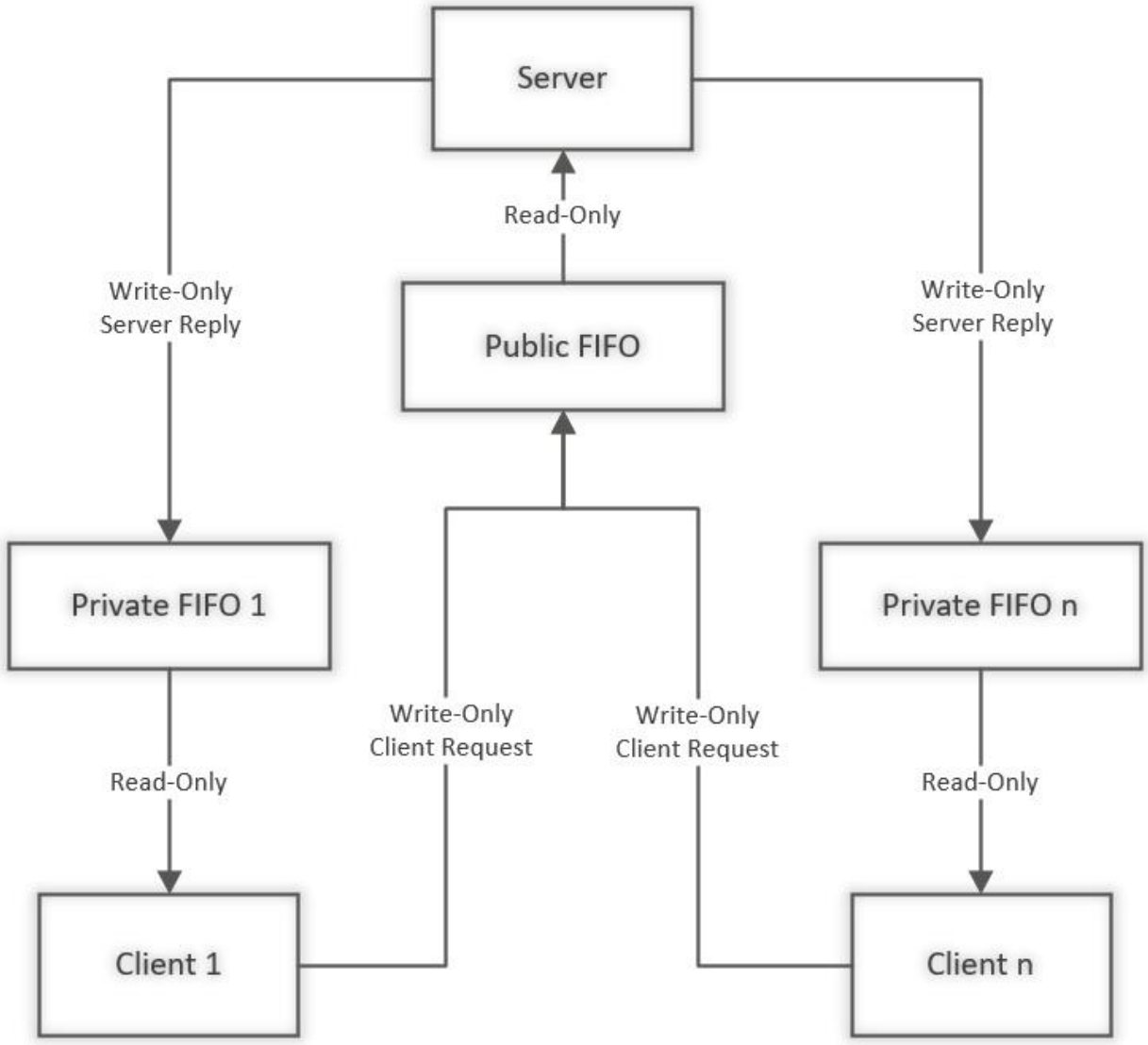
25. Необходимо предусмотреть возможность хранения истории переписок (на сервере) и поиска по ним. Связь между сервером и клиентом должна быть реализована при помощи pipe'ов

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.

История переписок хранится в векторе векторов строк. Для каждого пользователя создаётся отдельный вектор, в котором хранятся отправленные им и полученные им же сообщения. Доступ к истории сообщений осуществляется командой вида «history текст». Если в истории найдётся строка, подстрокой которой является текст, то история будет выведена на экран.

Комментарии к работе программы можно найти в самом коде, который представлен ние.

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>> history;

    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);
        history.push_back(vec);
        if (in(logins, login) == -1)
            logins.push_back(login);
        else
            std::cout << "already exists!";
    }
```

```
//std::cout << "TEST3\n";
 //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 rcvd_usr = extract_login(message);          //от кого
        std::string rcvd_adressee = extract_addressee(message); //кому
        std::string rcvd_message = extract_message(message);    //что
        int fd_repl = in(logins, rcvd_adressee);                //id получателя
        int fd_usr = in(logins, rcvd_usr);                      //id отправителя
        //std::cout << rcvd_adressee;
        int pos = -1;
        if (rcvd_adressee == "history")
        {
            std::string reply = "No matches found\n";
            for (int i = 0; i < history.size(); ++i)
            {
                if (i == fd_usr)
                {
                    for (int j = 0; j < history[i].size(); ++j)
                    {
                        if (search(history[i][j], extract_text(message)))
                        {
                            reply = history[i][j];
```

```
                        pos = i;
                    }
                }
            }
        }
        if (reply != "No matches found\n")
        {
            for (int i = 0; i < history.size(); ++i)
            {
                if (i != pos)
                {
                    for (int j = 0; j < history[i].size(); ++j)
                    {
                        if (search(history[i][j], extract_text(message)))
                        {
                            reply = "[" + logins[i] + "] " + history[i][j];
                        }
                    }
                }
            }
        }
        send_message_to_client(fd[fd_usr], reply);
    }
    else
    {
        for (int i = 0; i < history.size(); ++i)
        {
            if (logins[i] == rcvd_usr)
                history[i].push_back(extract_text(message));
```

```
                if (logins[i] == rcvd_adressee && rcvd_usr != rcvd_adressee)
                    history[i].push_back(extract_text(message));
            }
            if (in(logins, rcvd_adressee) == -1)
            {
                send_message_to_client(fd[fd_usr], "Login does not exists!\n");
            }
            else
            {
                send_message_to_client(fd[fd_repl], rcvd_message);
            }
        }
    }
}
```

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;

    //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 in VMAI. 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 == "history")
        {
            std::string pattern;
            std::getline(std::cin, pattern);
            send_message_to_server(fd_send, login, adressee, pattern);
        }
        else
        {
            if (adressee == "quit")
```

```
                break;
            std::getline(std::cin, message);
            send_message_to_server(fd_send, login, adressee, message);
        }
    }
    //return 0;
    thr_recieve.detach();
}
```

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)
        {
            //cout << "TEST";
            std::string pat;
            for (int z = 0; z < pattern.size(); ++z)
            {
                if (text[i + z] == pattern[z])
                    pat.push_back(text[i + z]);
            }
```

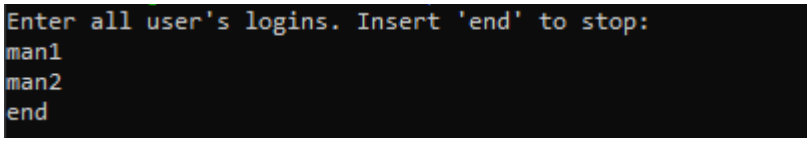
```
            if (pat == pattern)
            {
                return true;
            }

            pat.clear();
        }
    }
    return false;
}
```

Демонстрация работы программы

Регистрация пользователей «man1» и «man2» на сервере:

./server



Запуск клиентов с двух новых терминалов и ввод логинов, тестовых сообщений, поиск по истории сообщений и завершение работы (при этом сервер должен быть запущен).

Выводы

Данный курсовой проект оказался довольно интересным. Я закрепил навыки использования pipe, в целом узнал больше о межпроцессовом взаимодействии, закрепил навыки работы со строками в C++.