НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ Р.Е.АЛЕКССЕВА

ОТЧЕТ
по курсовой работе
по дисциплине
Технологии программирования

Выполнили:
Любимцев Павел
Антипин Роман
Бровкин Глеб
Горохов Дмитрий
Бушковский Даниил
Куликов Максим

Группа: 23-ПМ-1

Нижний Новгород
2024 г.

Содержание

1. Введение

1.1. Идея проекта

Перед выполнением лабораторной работы наша команда, состоящая из шести человек - Паши, Ромы, Димы, Максима, Глеба и Данила, столкнулась с интересной задачей - разработать базу данных. Идея пришла неожиданно и была настолько оригинальной, что сразу завоевала интерес всей группы. Мы решили создать базу данных для «дурдома» - вымышленной системы, в которой хранится информация о жителях необычного учреждения, их особенностях и взаимодействиях.

Вдохновленные нестандартной идеей, мы начали активно обсуждать дальнейший план действий и решили распределить роли между собой.

1.2. Распределение ролей

Долгое время мы думали над тем, кто за какую часть проекта будет отвечать. Тем не менее, ответственность в нашей команде распределилась следующим образом:

Паша - лидер команды. Он постоянно руководил процессом и подталкивал нас на работу. без него бы ничего не получилось. Помимо лидерской деятельности, Паша написал немалую часть кода.

Дима - разработчик. Они с Пашей - мозги нашей команды. Разработка легла на их плечи.

Данил и Глеб занялись разработкой внешнего вида нашего проекта. Они делали интерфейс.

Максим и Рома взяли на себя написание отчета и документации к проекту.

После распределения ролей мы оценили фронт работы и поняли, что нам предстоит нелегко. Не теряя лишнего времени, мы приступили к работе.

2. Рабочий процесс

2.1. Написание кода

Итак, приступим к самому интересному. Код уже написан, осталось объяснить читателю, что там происходит. Весь проект был разбит на 2 части: **Клиент** и **Сервер**. **Сервер** - это база данных, которая принимает к себе различные запросы с **Клиента** (К примеру, с клиента отправляется строчка информации, которую нужно изменить/добавить. Эта информация изменяется на сервере).

2.2. Сервер

В папке Server содержится 4 файла с расширением cpp. Рассмотрим каждый из них и разберемся в написанном коде.

2.2.1 main.cpp

Начнем с файла **main.cpp**

```cpp
#include <iostream>
#include <string>

#include "menu.h"
#include "network.h"

using namespace std;

int main(void)
{

        PsychiatryServer dbShizov;

        struct sockaddr_storage their_addr;
        socklen_t sin_size;
        int valread;
        int sockfd, new_fd;
        string log;
        char s[INET6_ADDRSTRLEN];

        connect_with_client(sockfd);

        printf("server: waiting for connections...\n");

        while(1) {
                sin_size = sizeof their_addr;
                new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size);
                if (new_fd == -1) {
                        perror("accept");
                        continue;
                }

                inet_ntop(their_addr.ss_family, get_in_addr((struct sockaddr *)&their_addr), s, sizeof s);

                cout << "server: got connection from " << s << endl;

                char buf[1024] = {0};
                valread = read(new_fd, buf, 1024);
                string str_buffer = buf;

                log = server_menu(str_buffer, dbShizov);

                send(new_fd, log.c_str() , strlen(log.c_str()), 0);
        }

        return 0;
}
```

Рис. 1: Код из файла main.cpp из папки Server

На Рис. (1) изображен код, который мы щас раскидаем по полочкам
Это главный серверный код, который инициирует подключение клиента и обрабатывает запросы от клиента.

Сетевое подключение:

- **Создаётся сервер** (с помощью connect_with_client), который слушает подключения на порту PORT.

- **Сервер** принимает входящие соединения (с помощью accept).

- После установления соединения сервер читает запрос от клиента (через сокет) и обрабатывает его, используя server_menu.

Основная логика:

- После того как клиент подключается, сервер читает строку запроса, передающую информацию (например, о добавлении, удалении пациента или врача) и вызывает соответствующие методы сервера для изменения данных.

- Результат обрабатывается и отправляется обратно клиенту.

2.2.2 asylum.cpp

Итак, второй файл - **asylum.cpp**.

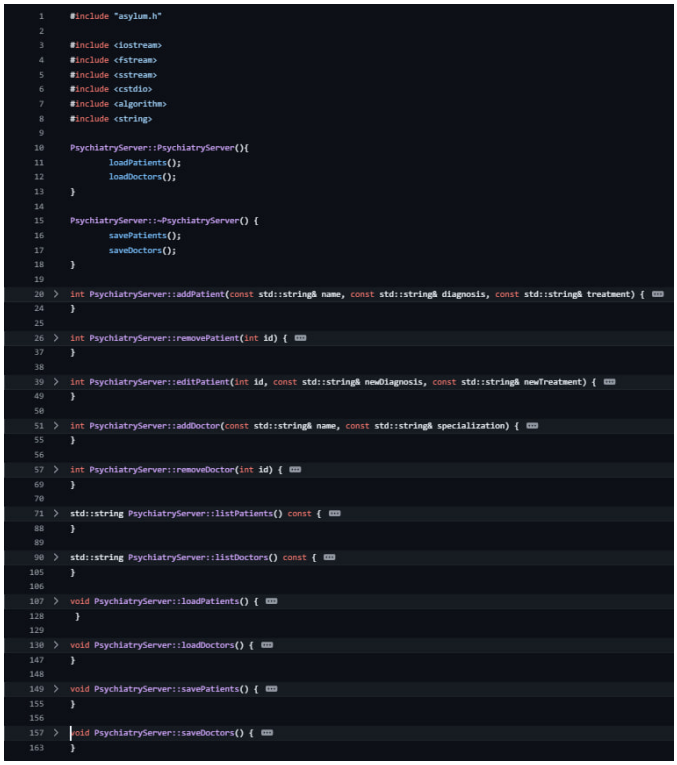

Рис. 2: Код из файла asylum.cpp из папки Server

Это класс PsychiatryServer, который управляет основной базой данных пациентов и врачей. Код изображен на Рис. (2). База данных:

- Класс хранит список пациентов (patients) и врачей (doctors) в виде векторов.
- С помощью методов можно добавлять, удалять или редактировать данные о пациентах и врачах.

Методы:

- **addPatient**: Добавление нового пациента.
- **removePatient**: Удаление пациента по ID.
- **editPatient**: Изменение диагноза и лечения пациента.
- **addDoctor**: Добавление нового врача.
- **removeDoctor**: Удаление врача по ID.
- **listPatients** и **listDoctors**: Возвращают список всех пациентов и врачей.

Сохранение и загрузка данных:

- Данные сохраняются в файлы (savePatients и saveDoctors) и загружаются при запуске программы (loadPatients и loadDoctors), используя стандартные потоки ввода/вывода.

2.2.3 menu.cpp

Переходим к третьему файлу. Им является **menu.cpp**. Рассмотрим код на Рис.(3)

```cpp
#include "menu.h"
#include "asylum.h"

#include <iostream>
#include <sstream>
#include <vector>

using namespace std;

string server_menu(string info, PsychiatryServer &db)
{
        istringstream iss(info);
        string s;
        vector<string> svec;

        while (std::getline(iss, s, ' ')) svec.push_back(s);

        string result;

        switch(stoi(svec[0])){
                case 1: {
            if(db.addPatient(svec[1], svec[2], svec[3]))
            result = "Patient added successfully.";
            break;
        }
        case 2: {
            if(db.removePatient(stoi(svec[1])))
            break;
        }
        case 3: {
            if (db.editPatient(stoi(svec[1]), svec[2], svec[3])) result = "Patient updated successfully.";
            else result = "Patient not found.";
            break;
        }
        case 4: {
            if(db.addDoctor(svec[1], svec[2])) result = "Doctor added successfully.";
            break;
        }
        case 5: {
            if(db.removeDoctor(stoi(svec[1]))) result = "Doctor removed successfully.";
            else result = "Doctor not found.";
            break;
        }
        case 6:
            result = db.listPatients();
            break;
        case 7:
            result = db.listDoctors();
            break;
        }

        db.savePatients();
        db.saveDoctors();
```

Рис. 3: Код из файла menu.cpp из папки Server

Это меню обработки запросов, получаемых от клиента, и взаимодействие с базой данных.

Обработка команд клиента:

- Когда **сервер** получает команду от **клиента**, она разбивается на части и передаётся в функции для добавления, удаления или редактирования данных.

- В зависимости от команды (например, добавление пациента или врача) вызываются соответствующие методы класса **PsychiatryServer**.

Ответ сервером:

- После выполнения операции **сервер** формирует ответ, который отправляется обратно **клиенту**.

2.2.4 network.cpp

Последний файл из папки **Server**, который мы рассмотрим - network.cpp. Это код для настройки и запуска сетевого **сервера**

```cpp
#include "menu.h"
#include "asylum.h"

#include <iostream>
#include <sstream>
#include <vector>

using namespace std;

string server_menu(string info, PsychiatryServer &db)
{
        istringstream iss(info);
        string s;
        vector<string> svec;

        while (std::getline(iss, s, ' ')) svec.push_back(s);

        string result;

        switch(stoi(svec[0])){
                case 1: {
            if(db.addPatient(svec[1], svec[2], svec[3]))
            result = "Patient added successfully.";
            break;
        }
        case 2: {
            if(db.removePatient(stoi(svec[1])))
            break;
        }
        case 3: {
            if (db.editPatient(stoi(svec[1]), svec[2], svec[3])) result = "Patient updated successfully.";
            else result = "Patient not found.";
            break;
        }
        case 4: {
            if(db.addDoctor(svec[1], svec[2])) result = "Doctor added successfully.";
            break;
        }
        case 5: {
            if(db.removeDoctor(stoi(svec[1]))) result = "Doctor removed successfully.";
            else result = "Doctor not found.";
            break;
        }
        case 6:
            result = db.listPatients();
            break;
        case 7:
            result = db.listDoctors();
            break;
        }

        db.savePatients();
        db.saveDoctors();
```

Рис. 4: Код из файла network.cpp из папки Server ч.1

```cpp
#include "menu.h"
#include "asylum.h"

#include <iostream>
#include <sstream>
#include <vector>

using namespace std;

string server_menu(string info, PsychiatryServer &db)
{
        istringstream iss(info);
        string s;
        vector<string> svec;

        while (std::getline(iss, s, ' ')) svec.push_back(s);

        string result;

        switch(stoi(svec[0])){
                case 1: {
            if(db.addPatient(svec[1], svec[2], svec[3]))
            result = "Patient added successfully.";
            break;
        }
        case 2: {
            if(db.removePatient(stoi(svec[1])))
            break;
        }
        case 3: {
            if (db.editPatient(stoi(svec[1]), svec[2], svec[3])) result = "Patient updated successfully.";
            else result = "Patient not found.";
            break;
        }
        case 4: {
            if(db.addDoctor(svec[1], svec[2])) result = "Doctor added successfully.";
            break;
        }
        case 5: {
            if(db.removeDoctor(stoi(svec[1]))) result = "Doctor removed successfully.";
            else result = "Doctor not found.";
            break;
        }
        case 6:
            result = db.listPatients();
            break;
        case 7:
            result = db.listDoctors();
            break;
        }

        db.savePatients();
        db.saveDoctors();
```

Рис. 5: Код из файла network.cpp из папки Server ч.2

На Рис. (4) и Рис. (5) код выполняет следующие функции:

Подключение:

- Настройка **серверного сокета** с помощью **getaddrinfo** и **bind**.

- **Сервер** принимает входящие соединения через **сокет** и обрабатывает их, используя обработчик сигналов **sigchld_handler**, который помогает избежать "зависших"процессов после завершения работы клиентов.

Обработка ошибок:

- Код включает обработку ошибок на каждом этапе (создание **сокета**, **соединение**, **привязка к порту** и т.д.)

Основные моменты с папкой Server Рассмотрены. Самое время посмотреть, что же интересного есть в папке Client.

2.3. Клиент

2.3.1 main.cpp

Как и у **Server**, у папки **Client**, естественно, присутствует файл main.cpp. Рассмотрим, что он выполняет на Рис. (6) и Рис. (7):

```cpp
#include <iostream>
#include <fstream>
#include <string>

#include "network.h"

using namespace std;

int main()
{
	int choice;


	do {
		string result = "";
		cout << "\n--- Psychiatry Server ---\n";
		cout << "1. Add Patient\n2. Remove Patient\n3. Edit Patient\n4. Add Doctor\n5. Remove Doctor\n6. List Patients\n7. List Doctors\n0. Exit\n";
		cout << "Choose: ";

		cin >> choice;
		cin.ignore();

		result += to_string(choice);
		result += ' ';


		switch (choice) {
		case 1: {
			string name, diagnosis, treatment;
			cout << "Enter patient name: ";
			getline(cin, name);
			cout << "Enter diagnosis: ";
			getline(cin, diagnosis);
			cout << "Enter treatment: ";
			getline(cin, treatment);

			result += name;
			result += ' ';
			result += diagnosis;
			result += ' ';
			result += treatment;

			break;
			}
		case 2: {
	    int id;
	    cout << "Enter patient ID to remove: ";
	    cin >> id;

	    result += to_string(id);
```

Рис. 6: Код из файла main.cpp из папки Client ч.1

Рис. 7: Код из файла main.cpp из папки Client ч.2

Это клиентская программа, которая предоставляет пользовательский интерфейс для взаимодействия с сервером.

Меню выбора:

- Пользователь может выбрать одно из действий: **добавить пациента**, **удалить пациента**, **изменить данные**, **добавить врача** и т.д.

Формирование запроса:

- В зависимости от выбранного действия **клиент** собирает информацию (например, **имя пациента**, **диагноз**) и отправляет её **серверу** через **сеть**.

Отправка запроса:

- После формирования строки запроса, программа вызывает **network::speak_to_server**, чтобы передать информацию на **сервер**.

2.3.2 network.cpp

Последний файл нашего проекта, который мы детально рассмотрим.

```cpp
#include "network.h"

#include <cstring>
#include <vector>

using namespace std;

int network::speak_to_server(string info)
{
        int sockfd, numbytes;
        char buf[MAXDATASIZE];
        struct addrinfo hints, *servinfo, *p;
        int rv;
        int valread;
        char s[INET6_ADDRSTRLEN];

        memset(&hints, 0, sizeof hints);
        hints.ai_family = AF_UNSPEC;
        hints.ai_socktype = SOCK_STREAM;

        if ((rv = getaddrinfo(HOSTNAME, PORT, &hints, &servinfo)) != 0) {
          fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
          return 1;
        }

        for(p = servinfo; p != NULL; p = p->ai_next) {
          if ((sockfd = socket(p->ai_family, p->ai_socktype,
                    p->ai_protocol)) == -1) {
                perror("client: socket");
                continue;
          }
          if (connect(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
                close(sockfd);
                perror("client: connect");
                continue;
          }
          break;
        }

        if (p == NULL) {
          fprintf(stderr, "client: failed to connect\n");
          return 2;
        }

        send(sockfd, info.c_str() , strlen(info.c_str()), 0);

        char buffer[1024] = {0};
        valread = read(sockfd, buffer, 1024);
        string str_buffer = buffer;

        cout << '\n' << str_buffer << endl;

        close(sockfd);

        return 0;
}
```

Рис. 8: Код из файла network.cpp из папки Client

На Рис. (8) показана функция для отправки данных на **сервер**

Сетевое соединение:

- Создаёт соединение с сервером через **сокет**, используя **getaddrinfo** и **connect**.

Отправка и получение данных:

- После установления соединения **клиент** отправляет строку запроса **серверу**, а затем получает ответ от **сервера**.
- Ответ отображается пользователю в **консоли**.

2.4. Небольшой подытог кодинга

Взаимодействие осуществляется с помощью **сервера** и **клиента**.

Сервер:

- Обрабатывает **запросы** от **клиента**.
- Использует класс **PsychiatryServer** для работы с **данными** (пациентами и врачами)
- Отвечает на команды **клиента**.

Клиент:

- Клиентская программа позволяет пользователю вводить команды через текстовое меню.
- Формирует запросы и отправляет их **серверу**.

- Выводит ответы от **сервера** на экран.

В работе были задействованы следующие библиотеки:

- **<stdlib.h>**
- **<cstdio>**
- **<unistd.h>**
- **<errno.h>**
- **<string.h>**
- **<sys/types.h>**
- **<sys/socket.h>**
- **<netinet/in.h>**
- **<netdb.h>**
- **<arpa/inet.h>**
- **<sys/wait.h>**
- **<signal.h>**

3. Выводы и заключения

В данном проекте наша команда реализовала серверные и клиентские части базы данных для психиатрической больницы. Серверная часть включает в себя обработку запросов, управление данными пациентов и врачей, а также взаимодействие с клиентами через сеть. Программы из папки Server обеспечивают создание, удаление и редактирование записей пациентов и врачей, а также их сохранение в файлы. Программа main.cpp реализует основную серверную логику, программы asylum.cpp и menu.cpp отвечают за хранение и обработку данных, а программа network.cpp - за сетевое соединение с клиентами.

Клиентская часть (программы main.cpp и network.cpp из папки Client) позволяет пользователям взаимодействовать с сервером через простое текстовое меню, отправляя запросы на сервер и получая ответы в виде результатов операций с данными. Программа main.cpp предоставляет интерфейс для ввода команд и данных, а программа network.cpp осуществляет отправку запросов на сервер и вывод полученных результатов.

В целом, проект демонстрирует работу распределенной системы с клиент-серверной архитектурой, в которой сервер управляет данными о пациентах и врачах, а клиент взаимодействует с этим сервером через сеть для выполнения необходимых операций.