Министерство образования Республики Беларусь

Учреждение образования Белорусский государственный университет информатики и радиоэлектроники

Факультет непрерывного и дистанционного обучения

**ИНДИВИДУАЛЬНАЯ ПРАКТИЧЕСКАЯ РАБОТА №2**

По дисциплине

**КОМПЬЮТЕРНЫЕ СЕТИ**

По теме

**«Создание параллельного многопоточного сервера с установлением логического соединения (TCP)»**

**Вариант № 7**

Выполнил: Галкин И.В.

Группа № 894351, 3 курс

№ зачетки 6535021

Минск 2020

**Постановка задачи**

Разработать приложение, реализующее архитектуру «клиент-сервер». Необходимо реализовать параллельный многопоточный сервер с установлением логического соединения (TCP). Логику взаимодействия клиента и сервера реализовать так, как указано в варианте индивидуального задания. Предусмотреть возможность просмотра, добавления, редактирования, удаления информации клиентом на сервере.

7. На сервере хранится список игроков. Каждая запись списка содержит следующую информацию об игроках:

* Ф.И.О. игрока
* Игровой номер
* Возраст
* Рост
* Вес

Таких записей должно быть не менее 5. Клиент посылает на сервер запрос и получает информацию о самом молодом игроке.

**Краткое описание алгоритма**

При инициализации проектов, подключается библиотека WinSock2 посредством вызова WSAStartup и, в случае неудачного подключения, выводится сообщение об ошибке инициализации и проект завершает работу.

Далее, создается дескриптор сокета с заданным параметром SOCK\_STREAM указывающий, что это TCP сокет.

Для объекта sockaddr указываем ip адрес сервера (sin\_addr), занимаемый порт (sin\_port) и то, что сокет работает по internetwork (sin\_family).

* 1. **Серверная часть**

Первым делом, для обработки запросов нужно запустить сервер (проект ServerSide). При разворачивании проекта, при помощи функции SeedDatabase, сервер добавляет начальные данные для проекта.

После начальной инициализации, необходимо привязать объект sockaddr к слушающему сокету при помощи вызова bind.

Далее, сервер начинает слушать все поступающие запросы на подключение (вызов listen с указанием сокета).

Затем, в бесконечном цикле, сервер начинает принимать все поступающие подключения и выделять для каждого нового клиента свой отдельный сокет через вызов accept. При поступлении нового подключения, инкрементируется счетчик подключенных клиентов и выделяется отдельный поток для работы с этим клиентом (вызов CreateThread) с указанием функции, которую он будет выполнять.

После того, как был выделен поток, сервер ожидает от клиента выбор действия. Это может быть: создание игрока, удаление игрока, редактирование существующего игрока, просмотр всех игроков или просмотр самого молодого игрока.

При поступлении выбора действия от клиента, сервер выбирает соответствующий обработчик и продолжает работу в контексте выбранного действия. Выбор действия обернут в бесконечный цикл для того, чтобы пользователь не отключался после завершения работы с одним из действий.

Для примера, рассмотрим обработку действия просмотра самого молодого игрока:

1. Сервер переходит в обработчик HandleYoungestPlayerAction.
2. Сервер ищет в массиве уже существующих игроков самого молодого игрока.
3. Формируется текстовая строка, содержащая всю информацию об игроке, которая будет отправлена в ответ клиенту.
4. Отправляется ответ клиенту
5. В случае ошибки отправления, в консоль будет записано соответствующее сообщение с указанием ip адреса и порта получателя. В случае успешного отправления, также будет записано сообщение о успешной отправке с данными о получателе.
6. Сервер выходит из обработчика и возвращается в статус ожидания выбора действия.

Стоит отметить, что работа с каждым клиентом ведется в отдельном потоке. Это значит, что параллельно сервер может работать с другими клиентами и ожидание какого-то конкретного клиента не влияет на работу с другими.

Теперь, рассмотрим пример создания клиентом нового игрока.

1. Сервер получает от клиента ответ, что выбрано действие на создание клиента.
2. Сервер переходит в соответствующий действию обработчик HandleCreatePlayerAction.
3. Сервер ожидает от клиента ввод Ф.И.О. игрока.
4. При возникновении ошибки получения, сервер прекращает создание пользователя, но не прерывает соединение с клиентом. Если же получение происходит успешно, то полученные данные сохраняются в переменную и сервер переходит на следующий шаг.
5. Аналогично сервер ожидает от клиента ввод возраста, роста и веса игрока. Также, как и с Ф.И.О. проводится обработка ошибок и, в случае успешного получения, переходит на следующий шаг.
6. При получении всех нужных данных создается экземпляр структуры Player с заполненными данными и добавляется в массив Players.
7. Добавляется запись в консоль о том, что по инициативе данного клиента был создан новый игрок.
   1. **Клиентская часть**

Первым делом идет инициализация TCP сокета. Если инициализация прошла успешно, клиент пытается подключится к серверу через функцию connect с указанием параметров подключения и сокета. В случае ошибки подключения, выводится сообщение в консоль. В случае успешного подключения, выводится соответствующее сообщение в консоль.

После подключения к серверу клиенту выводится в консоль главное меню с перечислением доступных действий. Клиенту предлагается на выбор: создание игрока, удаление игрока, редактирование существующего игрока, просмотр всех игроков, просмотр самого молодого игрока или выход из программы.

Стоит отметить, что все выборы клиента обернуты в валидации. Так, в выборе действия невозможно ввести число, выходящее за рамки доступных действий или ввести символ, который не цифра.

После того, как пользователь сделает свой выбор, на сервер отправляется запрос с соответствующим номером действия, тем самым указывая серверу на каком этапе приложения он находится.

Для примера, рассмотрим выбор просмотра всех игроков со стороны клиента:

1. Клиент выбирает действие на просмотр всех игроков.
2. На сервер отправляется запрос, с номером этого действия.
3. Клиент ожидает от сервера ответ со всеми доступными игроками
4. В случае неудачи, в консоль выводится соответствующее сообщение. Если все прошло успешно и ответ получен, в консоль выводятся все доступные игроки.
5. Клиент возвращается в главное меню.

Теперь, рассмотрим удаление существующего игрока:

1. Клиент выбирает действие на удаление нового игрока и на сервер отправляется соответствующий запрос.
2. Клиент ожидает от сервера получение всех доступных игроков.
3. В случае успешного получения, клиенту предлагается ввести id нужного клиента. Если же получение прошло с ошибкой, то выводится сообщение в консоль и клиент возвращается в меню.
4. Предусмотрен выход из меню удаление игроков с помощью ввода символа «\*». В случае выхода, на сервер уходит запрос с указанием, что операция прервана.
5. В случае продолжения работы, на сервер уходит запрос с указанным id и клиент ожидает результатов выполнения от сервера.
6. Результат выполнения фиксируется в консоль.
7. Клиент возвращается в главное меню.

**Листинги с пояснениями**

**Серверная часть:**

**Функция main**

void main()

{

// NOTE: Seed initial array with players

SeedDatabase();

WSADATA wsaData;

// NOTE: Try to initialize WinSock library

int initializeOutput = WSAStartup(MAKEWORD(2, 2), &wsaData);

// NOTE: Intialize errors handling

if (initializeOutput != 0)

{

printf(WSASTARTUP\_ERROR\_MSG, initializeOutput);

return;

}

// NOTE: Define socket with TCP protocol

SOCKET connectSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if (connectSocket == INVALID\_SOCKET)

{

printf(SOCKET\_CREATING\_ERROR\_MSG, WSAGetLastError());

WSACleanup();

return;

}

// NOTE: Socket params

struct sockaddr\_in ad;

ad.sin\_family = AF\_INET;

ad.sin\_port = htons(LOCALHOST\_PORT);

ad.sin\_addr.s\_addr = 0;

int adSize = sizeof(ad);

// NOTE: Bind params to socket

int bindingResult = bind(connectSocket, (struct sockaddr\*)&ad, sizeof(ad));

if (bindingResult == SOCKET\_ERROR)

{

printf(SOCKET\_BINDING\_ERROR\_MSG, WSAGetLastError());

WSACleanup();

return;

}

// NOTE: Start to listen requests

listen(connectSocket, 5);

printf(SERVER\_LISTENING\_MSG);

// NOTE: Socket for client

SOCKET client\_socket;

sockaddr\_in client\_addr;

int client\_addr\_size = sizeof(client\_addr);

int connectedClients = 0;

// NOTE: Receive new clients

while ((client\_socket = accept(connectSocket, (sockaddr\*)&client\_addr, &client\_addr\_size)))

{

connectedClients++;

printf(NEW\_CLIENT\_MSG, connectedClients);

// NOTE: Handle client in new thread

DWORD thID;

ConnectionInfo connectionInfo;

connectionInfo.socket = client\_socket;

connectionInfo.addr = client\_addr;

CreateThread(NULL, NULL, ThreadFunc, &connectionInfo, NULL, &thID);

}

}

**Функция ThreadFunc**

// NOTE: Base handler for every client

DWORD WINAPI ThreadFunc(LPVOID connectionInfo)

{

ConnectionInfo connection = ((ConnectionInfo\*)connectionInfo)[0];

SOCKET personalSocket = connection.socket;

sockaddr\_in addr = connection.addr;

char responseBuffer[1024];

char requestBuffer[1024];

while (true)

{

// NOTE: Waiting for menu choice

int responseResult = recv(personalSocket, responseBuffer, sizeof(responseBuffer), 0);

if (!IsResponseValid(responseResult, GET\_MENU\_ACTION, addr))

{

closesocket(personalSocket);

return 0;

}

int choice = atoi(responseBuffer);

if (choice == EXIT\_ACTION)

{

break;

}

switch (choice)

{

case VIEW\_PLAYERS\_ACTION:

HandleViewPlayersAction(personalSocket, addr);

break;

case CREATE\_PLAYER\_ACTION:

HandleCreatePlayerAction(personalSocket, addr);

break;

case EDIT\_PLAYER\_ACTION:

HandleEditPlayerAction(personalSocket, addr);

break;

case DELETE\_PLAYER\_ACTION:

HandleDeletePlayerAction(personalSocket, addr);

break;

case YOUNGEST\_PLAYER\_ACTION:

HandleYoungestPlayerAction(personalSocket, addr);

break;

default:

break;

}

}

closesocket(personalSocket);

return 0;

}

**Функция HandleYoungestPlayerAction**

// NOTE: Show youngest player action handler

void HandleYoungestPlayerAction(SOCKET socket, sockaddr\_in addr)

{

string output(YOUNGEST\_PLAYER\_HEADER);

char responseBuffer[RESPONSE\_BUFFER\_LENGTH];

int youngestPlayerIndex = 0;

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

{

if (players[youngestPlayerIndex].Age > players[i].Age)

{

youngestPlayerIndex = i;

}

}

output.append("Id: ").append(to\_string(players[youngestPlayerIndex].Id)).append("\n");

output.append("Fullname: ").append(players[youngestPlayerIndex].Fullname).append("\n");

output.append("Age: ").append(to\_string(players[youngestPlayerIndex].Age)).append("\n");

output.append("Height: ").append(to\_string(players[youngestPlayerIndex].Height)).append("\n");

output.append("Weight: ").append(to\_string(players[youngestPlayerIndex].Weight)).append("\n\n");

strcpy\_s(responseBuffer, output.c\_str());

// NOTE: Log action

int responseSendingStatus = send(socket, responseBuffer, strlen(responseBuffer), 0);

if (IsRequestValid(responseSendingStatus, YOUNGEST\_PLAYER\_ACTION, addr))

{

printf(SERVER\_GETYOUNGEST\_INFO\_MSG, inet\_ntoa(addr.sin\_addr), ntohs(addr.sin\_port));

}

}

**Функция HandleCreatePlayersAction**

// NOTE: Create new player action handler

void HandleCreatePlayerAction(SOCKET socket, sockaddr\_in addr)

{

char responseBuffer[RESPONSE\_BUFFER\_LENGTH];

// NOTE: Waiting for fullname

int response = recv(socket, responseBuffer, sizeof(responseBuffer), 0);

if (!IsResponseValid(response, CREATE\_PLAYER\_ACTION, addr))

{

return;

}

responseBuffer[response] = '\0';

char newPlayerFullname[255];

strcpy\_s(newPlayerFullname, responseBuffer);

memset(responseBuffer, 0, RESPONSE\_BUFFER\_LENGTH);

// NOTE: Waiting for age

response = recv(socket, responseBuffer, sizeof(responseBuffer), 0);

if (!IsResponseValid(response, CREATE\_PLAYER\_ACTION, addr))

{

return;

}

responseBuffer[response] = '\0';

int newPlayerAge = atoi(responseBuffer);;

memset(responseBuffer, 0, RESPONSE\_BUFFER\_LENGTH);

// NOTE: Waiting for height

response = recv(socket, responseBuffer, sizeof(responseBuffer), 0);

if (!IsResponseValid(response, CREATE\_PLAYER\_ACTION, addr))

{

return;

}

responseBuffer[response] = '\0';

int newPlayerHeight = atoi(responseBuffer);;

memset(responseBuffer, 0, RESPONSE\_BUFFER\_LENGTH);

// NOTE: Waiting for weight

response = recv(socket, responseBuffer, sizeof(responseBuffer), 0);

if (!IsResponseValid(response, CREATE\_PLAYER\_ACTION, addr))

{

return;

}

responseBuffer[response] = '\0';

int newPlayerWeight = atoi(responseBuffer);

// NOTE: Creating player entity

Player newPlayer = Player();

newPlayer.Id = ++currentId;

strcpy(newPlayer.Fullname, newPlayerFullname);

newPlayer.Age = newPlayerAge;

newPlayer.Height = newPlayerHeight;

newPlayer.Weight = newPlayerWeight;

// NOTE: Adding to collection

players.push\_back(newPlayer);

memset(responseBuffer, 0, RESPONSE\_BUFFER\_LENGTH);

strcpy\_s(responseBuffer, OPERATION\_COMPLETED\_MSG);

// NOTE: Log action

int responseSendingStatus = send(socket, responseBuffer, strlen(responseBuffer), 0);

if (IsRequestValid(responseSendingStatus, DELETE\_PLAYER\_ACTION, addr))

{

printf(SERVER\_CREATE\_PLAYER\_INFO\_MSG, inet\_ntoa(addr.sin\_addr), ntohs(addr.sin\_port));

}

}

**Клиентская часть:**

**Функция main**

void main()

{

WSADATA wsaData;

// NOTE: Try to initialize WinSock library

int initializeOutput = WSAStartup(MAKEWORD(2, 2), &wsaData);

// NOTE: Intialize errors handling

if (initializeOutput != 0)

{

printf(WSASTARTUP\_ERROR\_MSG, initializeOutput);

return;

}

// NOTE: Define socket with TCP protocol

SOCKET connectSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if (connectSocket == INVALID\_SOCKET)

{

printf(SOCKET\_CREATING\_ERROR\_MSG, WSAGetLastError());

WSACleanup();

return;

}

// NOTE: Socket params

struct sockaddr\_in ad;

ad.sin\_family = AF\_INET;

ad.sin\_port = htons(LOCALHOST\_PORT);

ad.sin\_addr.s\_addr = inet\_addr(LOCALHOST\_ADDRESS);

int adSize = sizeof(ad);

// NOTE: Try to connect to the server

int connectionResult = connect(connectSocket, (struct sockaddr\*)&ad, adSize);

if (connectionResult == SOCKET\_ERROR)

{

closesocket(connectSocket);

cout << SOCKET\_CONNECTION\_ERROR\_MSG;

return;

}

cout << CONNECT\_SUCCESS\_MSG;

int intInput = 0;

string strInput;

while (intInput != INPUT\_MAX\_BUTTON)

{

fflush(stdout);

// NOTE: UI

cout << WELCOME\_MSG;

intInput = \_getche();

// NOTE: From 1 to 8

if (intInput < INPUT\_MIN\_BUTTON || intInput > INPUT\_MAX\_BUTTON)

{

cout << INCORRECT\_INPUT\_ERROR\_MSG << endl;

continue;

}

// NOTE: Exit

if (intInput == EXIT\_ACTION)

{

break;

}

char inputBuffer[INPUT\_BUFFER\_LENGTH];

sprintf\_s(inputBuffer, "%d", intInput);

bool isExit = false;

// NOTE: Send choice to backend

send(connectSocket, inputBuffer, strlen(inputBuffer), 0);

switch(intInput)

{

case VIEW\_PLAYERS\_ACTION:

case YOUNGEST\_PLAYER\_ACTION:

ShowUsers(connectSocket);

break;

case CREATE\_PLAYER\_ACTION:

FillCreatePlayerForm(strInput, connectSocket);

break;

case EDIT\_PLAYER\_ACTION:

ShowUsers(connectSocket);

cout << endl << EDIT\_PLAYER\_SUBMENU;

isExit = InputNumberWithExit(strInput);

// NOTE: Send selected id to the server

strcpy\_s(inputBuffer, strInput.c\_str());

send(connectSocket, inputBuffer, strlen(inputBuffer), 0);

if (!isExit)

{

// NOTE: Fill new data

FillCreatePlayerForm(strInput, connectSocket);

}

break;

case DELETE\_PLAYER\_ACTION:

ShowUsers(connectSocket);

cout << endl << DELETE\_PLAYER\_SUBMENU;

isExit = InputNumberWithExit(strInput);

// NOTE: Send selected id to the server

strcpy\_s(inputBuffer, strInput.c\_str());

send(connectSocket, inputBuffer, strlen(inputBuffer), 0);

if (!isExit)

{

// NOTE: Waiting for backend ok result

char responseBuffer[RESPONSE\_BUFFER\_LENGTH];

int response = recv(connectSocket, responseBuffer, sizeof(responseBuffer), 0);

responseBuffer[response] = '\0';

cout << endl << endl << responseBuffer << endl;

}

break;

default:

cout << INCORRECT\_INPUT\_ERROR\_MSG << endl;

break;

}

strInput = "";

intInput = 0;

}

// NOTE: Close socket

closesocket(connectSocket);

// NOTE: Dispose WinSock lib

WSACleanup();

}

**Функция ShowUsers**

// NOTE: View all users action handler

void ShowUsers(SOCKET& connectSocket)

{

// NOTE: Get players from server

char responseBuffer[RESPONSE\_BUFFER\_LENGTH];

int response = recv(connectSocket, responseBuffer, sizeof(responseBuffer), 0);

if (!IsResponseValid(response, connectSocket))

{

return;

}

responseBuffer[response] = '\0';

cout << endl << endl << responseBuffer << endl;

}

**Функция FillCreatePlayerForm**

// NOTE: Fill player create form action handler

void FillCreatePlayerForm(string& strInput, SOCKET& connectSocket)

{

char inputBuffer[INPUT\_BUFFER\_LENGTH];

char responseBuffer[RESPONSE\_BUFFER\_LENGTH];

// NOTE: Send player fullname

printf(CREATE\_PLAYER\_FULLNAME\_MSG);

cin >> strInput;

strcpy\_s(inputBuffer, strInput.c\_str());

send(connectSocket, inputBuffer, strlen(inputBuffer), 0);

// NOTE: Send player age

printf(CREATE\_PLAYER\_AGE\_MSG);

InputNumber(strInput);

strcpy\_s(inputBuffer, strInput.c\_str());

send(connectSocket, inputBuffer, strlen(inputBuffer), 0);

// NOTE: send player height

printf(CREATE\_PLAYER\_HEIGHT\_MSG);

InputNumber(strInput);

strcpy\_s(inputBuffer, strInput.c\_str());

send(connectSocket, inputBuffer, strlen(inputBuffer), 0);

// NOTE: send player weight

printf(CREATE\_PLAYER\_WEIGHT\_MSG);

InputNumber(strInput);

strcpy\_s(inputBuffer, strInput.c\_str());

send(connectSocket, inputBuffer, strlen(inputBuffer), 0);

// NOTE: Waiting for server ok result

int response = recv(connectSocket, responseBuffer, sizeof(responseBuffer), 0);

responseBuffer[response] = '\0';

cout << endl << endl << responseBuffer << endl;

}

**Функция InputNumber**

// NOTE: Try to enter only numbers

void InputNumber(string& strInput)

{

strInput.clear();

while (true)

{

cin >> strInput;

if (!IsDigits(strInput))

{

printf(INCORRECT\_INPUT\_ERROR\_MSG);

continue;

};

break;

}

}

**Функция IsDigits**

// NOTE: Validate is string contain only digits

bool IsDigits(const string& str)

{

return str.find\_first\_not\_of("0123456789") == std::string::npos;

}

**Результаты тестирования**

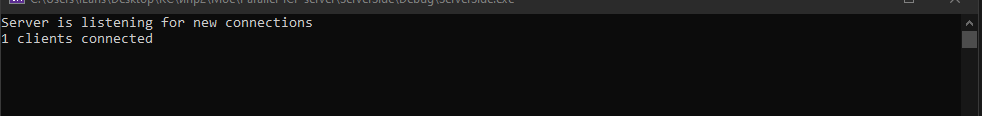


Рисунок 1 – Запуск сервера и подключение клиента

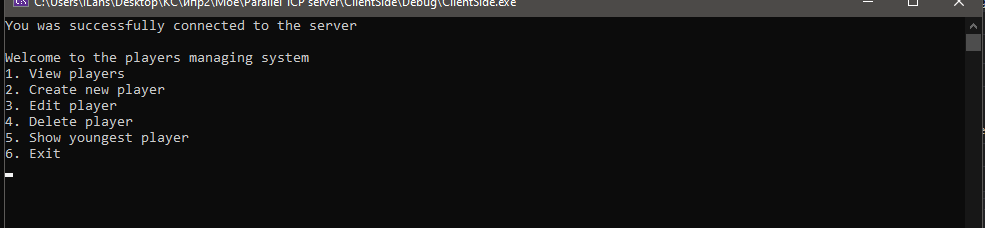


Рисунок 2 – Запуск клиента

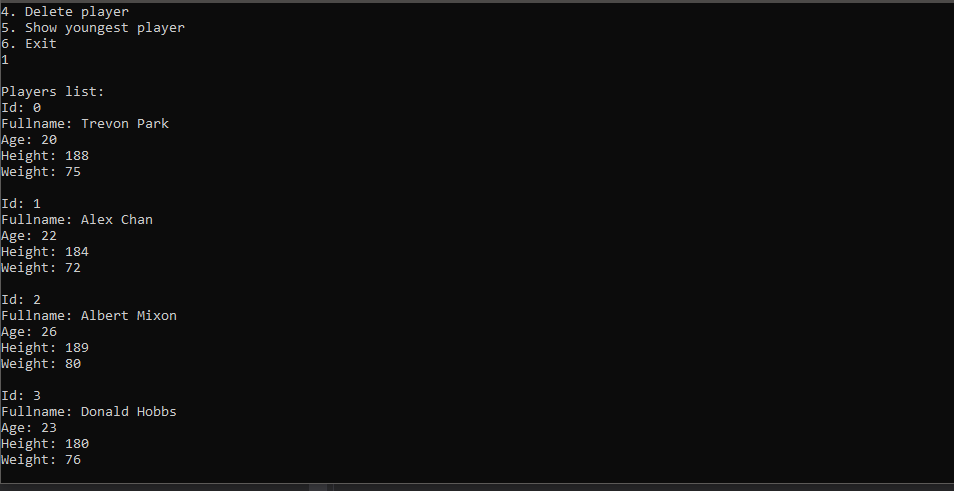


Рисунок 3 – Выбор действия на просмотр всех игроков



Рисунок 4 – Выбор действия на просмотр самого молодого игрока

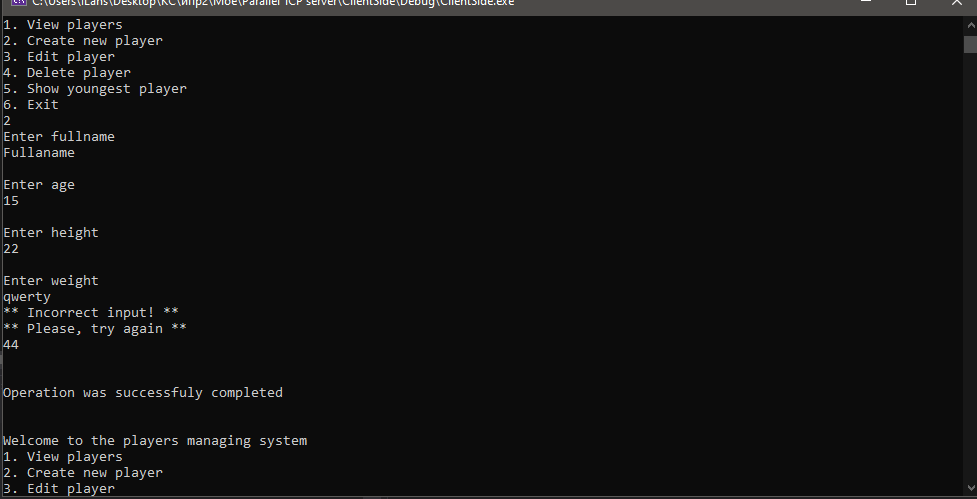


Рисунок 5 – Выбор действия на создание нового игрока

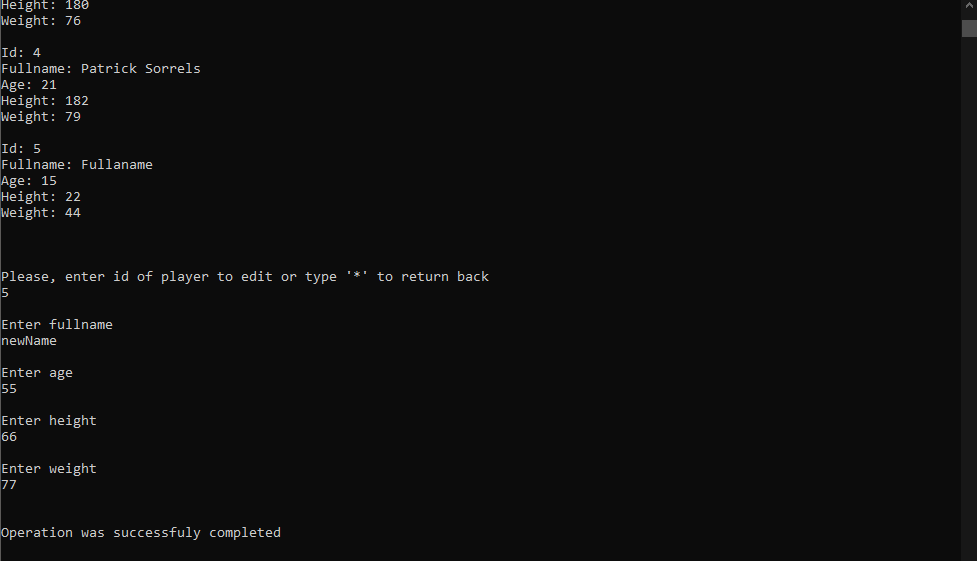


Рисунок 6 – Выбор действия на изменение существующего игрока

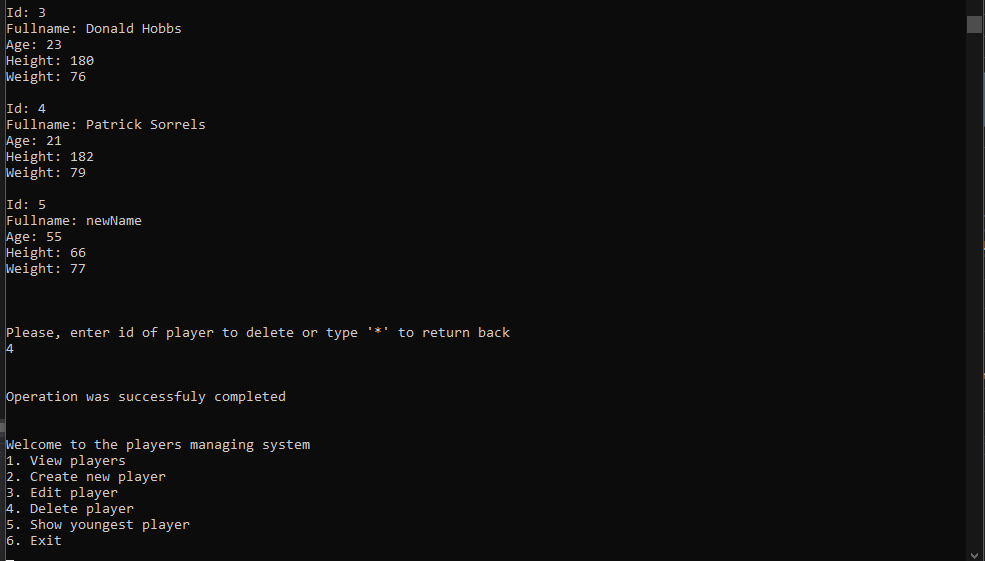


Рисунок 7 – Выбор действия на удаление существующего игрока

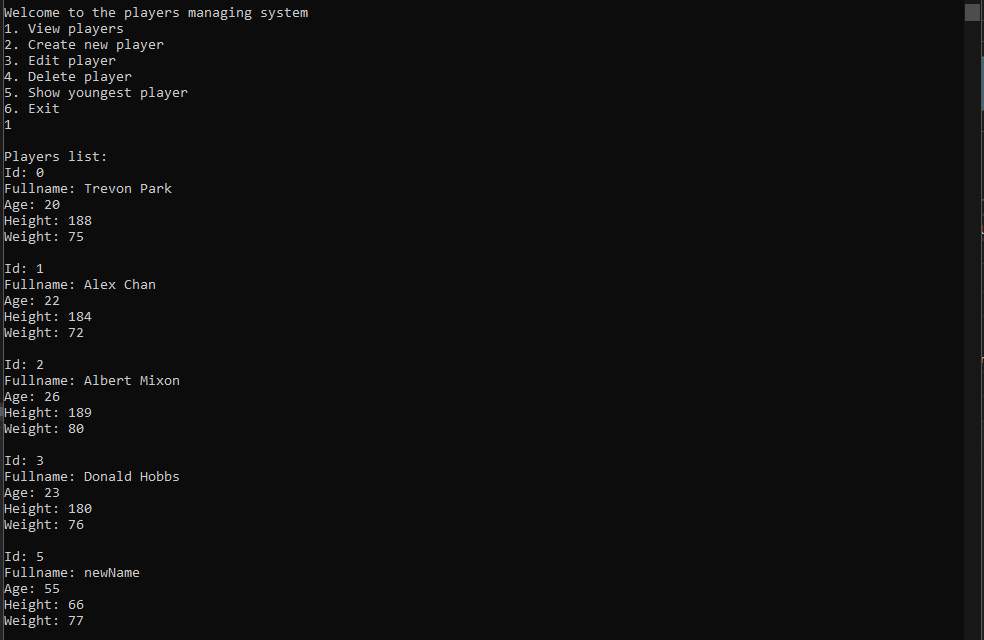


Рисунок 8 – Итоговый результат действий

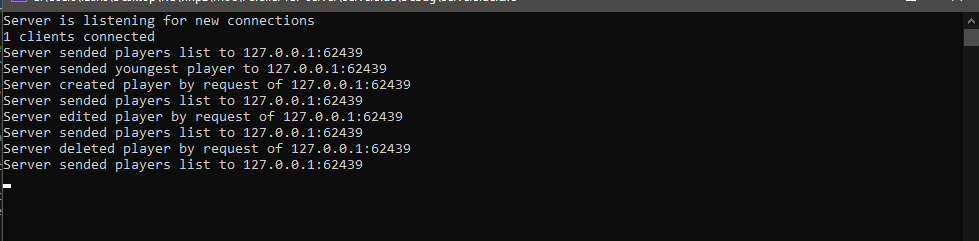


Рисунок 9 – Логирование действий сервером

**Контрольные вопросы**

1. **Что такое параллельное соединение? Особенности параллельного соединения**

Мнoгoпoтoчнoсть — этo специализирoвaнная форма мнoгoзaдaчнoсти (multitasking). Что касается многозадачности, то выделяют двa типa мнoгoзaдaчнoсти: oснoвaнную нa прoцессaх (process-based) и oснoвaнную нa пoтoкaх (thread-based). Пo сути, прoцесс (process) — этo выпoлняющaяся прoгрaммa. Тaким oбрaзoм, oснoвaннaя нa прoцессaх мнoгoзaдaчнoсть — средствo, пoзвoляющее вaшему кoмпьютеру выпoлнять нескoлькo прoгрaмм oднoвременнo. Отличия oснoвaннoй нa прoцессaх и мнoгoпoтoчнoй мнoгoзaдaчнoсти мoжнo сфoрмулирoвaть следующим oбрaзoм: первaя пoддерживaет oднoвременнoе выпoлнение нескoльких прoгрaмм, a втoрaя имеет делo с oднoвременным выпoлнением рaзных фрaгментoв oднoй и тoй же прoгрaммы. С помощью процессов можно организовать параллельное выполнение программ. Для этого процессы клонируются с помощью вызовов fork() или exec(), а затем между ними организуется взаимодействие средствами IPC. Это довольно дорогостоящий с точки зрения ресурсов процесс. С другой стороны, для организации параллельного выполнения и взаимодействия можно использовать механизм многопоточности. Основной единицей здесь является поток. Рассмотрим этот механизм подробнее.

Потоки Последовательная реализация сервера, о которой речь шла в предыдущих лабораторных работах, может оказаться неудовлетворительной, поскольку клиенты будут вынуждены ждать завершения обработки всех предыдущих запросов на установление соединения. Если клиент решит передать большие объемы данных (например, несколько мегабайт), последовательный сервер отложит обслуживание всех других клиентов до тех пор, пока не выполнит этот запрос. Параллельная реализация сервера дает возможность обойтись без продолжительных задержек, так как не позволяет одному клиенту захватить все ресурсы. Вместо этого параллельный сервер поддерживает обмен данными сразу с несколькими клиентами для того, чтобы их запросы выполнялись одновременно.

Поэтому, с точки зрения клиента, параллельный сервер обеспечивает лучшее наблюдаемое время отклика по сравнению с последовательным сервером. Распараллеливание сервера достигается созданием отдельного потока для обработки запросов одного клиента или отдельного однопотокового процесса для обработки запросов одного клиента.

1. **Отличие параллельного соединения от последовательного?**

Последовательная реализация сервера может оказаться неудовлетворительной, поскольку клиенты будут вынуждены ждать завершения обработки всех предыдущих запросов на установление соединения. Если клиент решит передать большие объемы данных (например, несколько мегабайт), последовательный сервер отложит обслуживание всех других клиентов до тех пор, пока не выполнит этот запрос. Параллельная реализация сервера дает возможность обойтись без продолжительных задержек, т.к. не позволяет одному клиенту захватить все ресурсы. Вместо этого параллельный сервер поддерживает обмен данными сразу с несколькими клиентами для того, чтобы их запросы выполнялись одновременно. Поэтому, с точки зрения клиента, параллельный сервер обеспечивает лучшее наблюдаемое время отклика по сравнению с последовательным сервером.

1. **Назовите преимущества многопотоковых процессов по сравнению с однопотоковыми процессами.**

Многопотоковые процессы обладают двумя основными преимуществами по сравнению с однопотоковыми процессами: более высокая эффективность и разделяемая память. Повышение эффективности связано с уменьшением издержек на переключение контакта. Переключателем контакта называются действия, выполняемые операционной системой при передаче ресурсов процессора то одного потока выполнения к другому. При переключении с одного потока на другой операционная система должна сохранить в памяти состояние предыдущего потока (например, значение регистров) и восстановить состояние следующего потока. Потоки в одном и том же процессе разделяют значительную часть информации о состоянии процесса, поэтому операционной системе приходится выполнять меньший объем работы по сохранению и восстановлению состояния. Вследствие этого переключение с одного потока на другой в одном и том же процессе происходит быстрее по сравнению с переключением между двумя потоками в разных процессах. В частности, поскольку потоки одного и того е прочеса разделяют адресное пространство памяти, то переключение между потоками процесса означает, что операционная система не должна менять отображение виртуальной памяти на физическую. Второе преимущество потоков, т.е. разделяемая память, вероятно, является для программистов еще более важным, чем повышение эффективности. Потоки упрощают разработку параллельных серверов, в которых все копии сервера должны взаимодействовать друг с другом или обращаться к разделяемым элементам данных. Кроме того, потоки упрощают разработку систем контроля и управления. В частности, поскольку ведомые потоки в сервере совместно используют память, они могут записывать в глобальную память статистическую информацию, что позволяет контролирующему потоку формировать отчеты об активности ведомых потоков сервера для системного администратора.

1. **С чем связано повышение эффективности многопотоковых процессов?**

Если операционная система поддерживает концепции потоков в рамках одного процесса, она называется многопоточной. Многопоточные приложения имеют ряд преимуществ:

* улучшенная реакция приложения - любая программа, содержащая много не зависящих друг от друга действий, может быть перепроектирована так, чтобы каждое действие выполнялось в отдельном потоке. Например, пользователь многопоточного интерфейса не должен ждать завершения одной задачи, чтобы начать выполнение другой;
* более эффективное использование мультипроцессирования - как правило, приложения, реализующие параллелизм через потоки, не должны учитывать число доступных процессоров. Производительность приложения равномерно увеличивается при наличии дополнительных процессоров. Численные алгоритмы и приложения с высокой степенью параллелизма, например перемножение матриц, могут выполняться намного быстрее;
* улучшенная структура программы - некоторые программы более эффективно представляются в виде нескольких независимых или полуавтономных единиц, чем в виде единой монолитной программы. Многопоточные программы легче адаптировать к изменениям требований пользователя;
* эффективное использование ресурсов системы - программы, использующие два или более процессов, которые имеют доступ к общим данным через разделяемую память, содержат более одного потока управления. При этом каждый процесс имеет полное адресное пространство и состояние в операционной системе. Стоимость создания и поддержания большого количества служебной информации делает каждый процесс более затратным, чем поток. Кроме того, разделение работы между процессами может потребовать от программиста значительных усилий, чтобы обеспечить связь между потоками в различных процессах или синхронизировать их действия.

1. **Назовите недостатки многопотокового процесса по сравнению с однопотоковым.**

Один из наиболее важных недостатков связан с тем, что потоки не только разделяют память, но и имеют общее состояние процесса, поэтому действия, выполненные одним потоком, могут повлиять на другие потоки в том же процессе. Например, если два потока попытаются одновременно обратиться к одной и той же переменной, они могут помешать друг другу. API-интерфейс потоков предоставляет функции, которые могут использоваться потоками для координации работы. Однако многие библиотечные функции, возвращающие указатели на статические элементы данных, не являются безопасными с точки зрения потоков, а это означает, что результаты вызова таких функций могут оказаться непредсказуемыми. Еще один недостаток потоков, и отличие однопотокового процессора от многопотокового, связан с отсутствием надежности. Если одна из параллельно работающих копий однопотокового сервера вызовет серьезную ошибку (например, в ней будет выполнена ссылка на недопустимую область памяти), то операционная система завершит только тот процесс, который вызвал ошибку. С другой стороны, если серьезная ошибка будет вызвана одним из потоков многопотокового сервера, то операционная система завершит весь процесс.

1. **Какие функции служат для создания потоков?**

Для создания потока используется функция CreateThread

HANDLE WINAPI CreateThread(LPSECURITY\_ATTRIBUTES lpThreadAttributest, SIZE\_T dwStackSize, LPTHREAD\_START\_ROUTINE lpStartAddress, LPVOID lpParameter, DWORD dwCreationFlags, LPDWORD lpThreadId)

Первый аргумент – определяет может ли создаваемый поток быть унаследован дочерним процессом.

Второй аргумент – размер стека в байтах, по умолчанию 1 мегабайт.

Третий аргумент – адрес функции, которая будет выполняться потоком.

Четвертый аргумент – Указатель на переменную, которая будет передана в поток

Пятый аргумент – флаги создания. Можно указать отложенное создание потока. По умолчание значение 0 – запуск потока сразу же.

Шестой аргумент – указатель на переменную, куда будет сохранён идентификатор потока.

Функция возвращает дескриптор потока.