МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Український державний університет науки і технологій**

Кафедра «Комп’ютерні інформаційні технології»

**Лабораторна робота №12**

**з дисципліни «Операційні системи»**

**на тему: «Мультиплексування.»**

Виконав:

Студент гр. ПЗ2011

Кулик С.В.

Прийняв:

Андрющенко В.О.

Дніпро, 2023

**Тема:** Мультиплексування.

**Мета:** Написати програми, які реалізують взаємодію клієнтів і сервера з використанням механізму мультиплексування.

**Короткі теоретичні відомості**

Мультиплексування - це механізм, який дозволяє обробляти одночасно багато з'єднань вводу/виводу на одному потоці виконання. Використання мультиплексування може значно поліпшити ефективність обробки з'єднань, особливо у випадку з багатьма клієнтами та великим обсягом одночасних вхідно-вихідних операцій.

Один із найпоширеніших механізмів мультиплексування в Linux - це функція select(). Вона дозволяє контролювати стан декількох сокетів та обробляти події вводу/виводу на цих сокетах. Функція select() очікує на події на визначених сокетах, і коли будь-яка подія стає доступною (наприклад, прийом даних або готовність до відправлення), вона повертає контроль до програми, щоб ви могли відповідно реагувати.

У клієнт-серверних додатках мультиплексування дозволяє серверу обробляти багато клієнтських з'єднань одночасно без необхідності створення окремих потоків або процесів для кожного з'єднання. Замість цього, сервер може використовувати функцію select() для відстеження подій на всіх сокетах клієнтів та реагувати на них відповідним чином. Це забезпечує ефективне використання ресурсів та зниження витрат на управління потоками або процесами.

Мультиплексування дозволяє створювати ефективні та масштабовані мережеві додатки. Воно дозволяє серверу обслуговувати багато клієнтів одночасно і забезпечує покращення продуктивності системи. Функція select() є потужним інструментом для реалізації мультиплексованого введення/виведення в мові програмування C++ та інших мовах програмування.

Програми, що використовують мультиплексування, можуть бути складнішими у розробці порівняно зі звичайними синхронними додатками. Однак, вони надають більшу гнучкість та ефективність у роботі з багатьма з'єднаннями одночасно. Мультиплексування є важливою технологією для будь-якого розробника, який стикається з розробкою мережевих додатків.

**Завдання**

Написати програми, які реалізують взаємодію клієнтів і сервера з використанням механізму мультиплексування.

**Текст програми**

**Файл server.cpp**

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

#include <cstring>

#include <cstdlib>

#include <cstdio>

#include <cerrno>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include <sys/select.h>

const int MAX\_CLIENTS = 10;

const int BUFFER\_SIZE = 1024;

void handleError(const std::string& message) {

std::cerr << "Error: " << message << " (" << strerror(errno) << ")" << std::endl;

exit(1);

}

int main() {

int serverSocket, maxSocket, activity, newSocket, clientSocket[MAX\_CLIENTS];

sockaddr\_in serverAddress;

fd\_set readSet;

std::vector<int> clientSockets;

char buffer[BUFFER\_SIZE];

// Create server socket

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

if (serverSocket == -1) {

handleError("Failed to create socket");

}

// Set server details

serverAddress.sin\_family = AF\_INET;

serverAddress.sin\_addr.s\_addr = INADDR\_ANY;

serverAddress.sin\_port = htons(8088);

// Bind server socket to address and port

if (bind(serverSocket, (struct sockaddr\*)&serverAddress, sizeof(serverAddress)) < 0) {

handleError("Failed to bind socket");

}

// Listen for incoming connections

if (listen(serverSocket, 5) < 0) {

handleError("Failed to listen");

}

// Accept incoming connections and handle data

std::cout << "Server started." << std::endl;

// Clear the client sockets

std::memset(clientSocket, 0, sizeof(clientSocket));

while (true) {

// Clear the socket set

FD\_ZERO(&readSet);

// Add server socket to set

FD\_SET(serverSocket, &readSet);

maxSocket = serverSocket;

// Add client sockets to set

for (auto& client : clientSockets) {

if (client > 0) {

FD\_SET(client, &readSet);

}

if (client > maxSocket) {

maxSocket = client;

}

}

// Wait for activity on any socket

activity = select(maxSocket + 1, &readSet, nullptr, nullptr, nullptr);

if (activity < 0) {

handleError("Select error");

}

// New connection

if (FD\_ISSET(serverSocket, &readSet)) {

if ((newSocket = accept(serverSocket, nullptr, nullptr)) < 0) {

handleError("Failed to accept connection");

}

// Add new socket to vector of client sockets

clientSockets.push\_back(newSocket);

std::cout << "New connection accepted" << std::endl;

}

// Data from client

for (auto& client : clientSockets) {

if (FD\_ISSET(client, &readSet)) {

std::memset(buffer, 0, BUFFER\_SIZE);

// Receive data from client

int bytesRead = recv(client, buffer, BUFFER\_SIZE, 0);

if (bytesRead == 0) {

// Connection closed by client

close(client);

clientSockets.erase(std::remove(clientSockets.begin(), clientSockets.end(), client), clientSockets.end());

std::cout << "Connection closed" << std::endl;

} else {

// Process received data

std::cout << "Message from client: " << buffer << std::endl;

// Check for "quit" command

std::string message(buffer);

message.erase(std::remove(message.begin(), message.end(), '\n'), message.end());

std::transform(message.begin(), message.end(), message.begin(), ::tolower);

if (message == "quit") {

close(client);

clientSockets.erase(std::remove(clientSockets.begin(), clientSockets.end(), client), clientSockets.end());

std::cout << "Connection closed" << std::endl;

}

}

}

}

}

// Close server socket

close(serverSocket);

return 0;

}

**Файл client.cpp**

#include <iostream>

#include <string>

#include <cstring>

#include <cstdlib>

#include <cstdio>

#include <cerrno>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

const int BUFFER\_SIZE = 1024;

void handleError(const std::string& message) {

std::cerr << "Error: " << message << " (" << strerror(errno) << ")" << std::endl;

exit(1);

}

int main() {

int clientSocket;

sockaddr\_in serverAddress;

// Create client socket

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

if (clientSocket == -1) {

handleError("Failed to create socket");

}

// Set server details

serverAddress.sin\_family = AF\_INET;

serverAddress.sin\_port = htons(8088);

// Convert IP address from string to binary format

if (inet\_pton(AF\_INET, "127.0.0.1", &(serverAddress.sin\_addr)) <= 0) {

handleError("Invalid address or address not supported");

}

// Connect to server

if (connect(clientSocket, (struct sockaddr\*)&serverAddress, sizeof(serverAddress)) < 0) {

handleError("Connection failed");

}

std::cout << "Connected to server. Enter 'quit' to exit." << std::endl;

while (true) {

// Read message from console

std::string message;

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

// Send message to server

ssize\_t bytesSent = send(clientSocket, message.c\_str(), message.length(), 0);

if (bytesSent < 0) {

handleError("Failed to send message");

}

// Check for "quit" command

if (message == "quit") {

break;

}

}

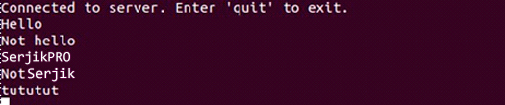
// Close client socket

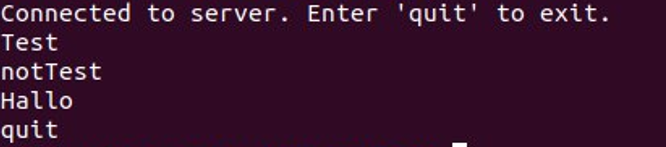
close(clientSocket);

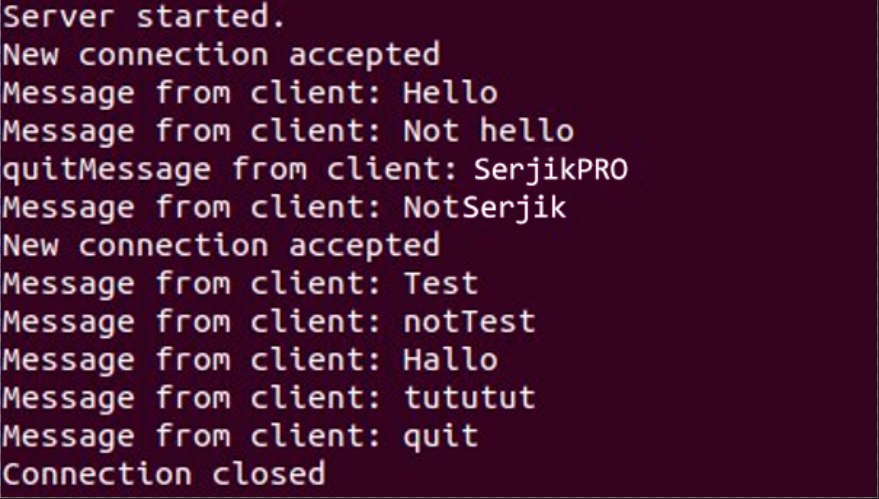
return 0;

}

**Результати програми та тестові прилади**





****

**Висновки**

Створення багатопотокових серверних застосувань дозволяє забезпечити ефективну взаємодію між клієнтами і сервером, де клієнти можуть відправляти дані серверу, а сервер обробляє ці дані і повертає результат клієнту. Використання потоків дозволяє обробляти багато запитів одночасно, що покращує продуктивність і швидкість відгуку системи.

Програма, яка була написана для гри, демонструє таку взаємодію. Кожен клієнт підключається до сервера через сокети і взаємодіє з ним шляхом передачі чисел, що є спробами вгадати випадкове число. Сервер обробляє ці спроби, порівнюючи їх з випадковим числом і відправляє відповіді клієнтам, які показують, чи була спроба вгадана, або вказують на те, чи потрібно обрати більше чи менше число.

Програма використовує потоки для обробки багатьох клієнтів одночасно. Кожен клієнт виконується в окремому потоці, що дозволяє їм незалежно взаємодіяти з сервером.