МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Український державний університет науки і технологій**

Кафедра «Комп’ютерні інформаційні технології»

**Лабораторна робота №11**

**з дисципліни «Операційні системи»**

**на тему: «Створення багатопотокових серверних застосувань»**

Виконав:

Студент гр. ПЗ2011

Кулик С.В.

Прийняв:

Андрющенко В.О.

Дніпро, 2023

**Тема:** Створення багатопотокових серверних застосувань.

**Мета:**

- Написати програми, які реалізують взаємодію клієнтів і сервера через сокети з використанням потоків.

- Клієнт повинен послати дані серверу, а сервер – обробити отримані дані і послати результат клієнтові.

**Короткі теоретичні відомості**

Існують дві мети, які переслідують програмісти, використовуючи потоки: дозволити додатку паралельно працювати над декількома відносно незалежними завданнями; використовувати переваги багатопроцесорних систем для підвищення продуктивності додатка. Як не парадоксально це може здатися на перший погляд, але ці завдання ортогональні один одному і навіть можуть вступати в протиріччя. Тому, ще на етапі проектування ви повинні визначитися з пріоритетами цих завдань, оскільки це і визначить всю структуру додатка. Очевидно, що у будь-якому випадку завдання, над яким працює додаток, повинне володіти однією істотною властивістю: його можна розділити на декілька більш менш незалежних підзадач, над якими можна працювати паралельно. Інакше використання багатопотоковості просто неможливе. Інколи таке розпаралелювання само по собі складає складне алгоритмічне завдання, не завжди воно в принципі можливе. Але часто воно виникає природним чином. Ось дві характерних ситуації. По-перше, серверні застосування, де обслуговування кожного клієнта є самостійним завданням. По-друге, це призначене для користувача застосування, яке виконує якусь тривалу операцію (проводить складні математичні обчислення, обробляє графіку, веде пошук в базі даних...) у фоновому режимі. Тобто, під час неї користувач повинен мати можливість продовжувати працювати з додатком. Тут завдання дві: сама тривала операція і робота графічного інтерфейсу. Якими ж способами можна організувати паралельну роботу програми над декількома завданнями? Почнемо з того, що для цього, в принципі, можна обійтися взагалі без потоків (звичайно, про використання многопроцессорності при цьому мові не йде). Найпростіший і очевидніший спосіб організувати паралельну роботу додатка над декількома завданнями – це звичайно доручити кожну з них окремому потоку. Найчастіше потоки використовують саме такий чином. Головне достоїнство цього рішення – його простота (в усякому разі, до тих пір, поки справа не доходить до синхронізації). За перемиканням між завданнями стежить система, програміст пише код, що працює над кожним завданням так, як якби це була незалежна програма. Можливості багатопроцесорних систем також виявляються задіяні. На практиці цей підхід прекрасно працює в більшості випадків. Так, для тривалої операції у фоновому режимі це ідеальний варіант. Уявивши собі структуру багатопотокового застосування, ми плавно переходимо до теми синхронізації потоків. Перш ніж приступити до обговорення різних механізмів синхронізації, спробуємо уявити собі декілька характерних ситуацій, в яких потрібна синхронізація. Одне з достоїнств багатопоточності – це легкий доступ до загальних даних для всіх потоків в процесі. Конкретний приклад: призначене для користувача застосування, де робочий потік виконує тривалу операцію у фоновому режимі. Згідно з правилами програмістів хорошого тону, слід інформувати користувача про хід операції. Для цього робочий потік повинен буде періодично записувати в якусь структуру даних інформацію на зразок прочитаних мегабайтів, числа оброблених файлів, ну і звичайно ж загальний відсоток виконання. Головний потік, що відповідає за GUI, повинен періодично прочитувати ці дані і відображувати на екрані. Якщо обмежитися лише одним числом, скажемо відсотком виконання, можна було б не турбуватися, але якщо даних багато, може статися так, що головний потік почне читати дані якраз в той момент, коли робочий потік їх модифікує. В результаті користувач ризикує побачити на екрані повну нісенітницю. Щоб уникнути цього, необхідно гарантувати, що із загальними даними одночасно може працювати лише одні потік.

**Завдання**

Реалізувати взаємодію процесів за схемою «клієнт-сервер» з використанням для передачі запиту та отримання відповіді сокетів. Клієнтська і серверна програми запускаються на різних комп’ютерах. Сервер підтримує декілька клієнтів одночасно. Реалізувати роботу сервера в багатопотоковому режимі та в режимі мультиплексування. Варіант 6 - Обробка сервером масивів (перетасування елементів).

**Текст програми**

**Файл server.cpp**

#include <iostream>

#include <vector>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

#include <cstring>

#include <algorithm>

#include <random>

#include <chrono>

#include <sstream>

#include <sys/types.h>

#include <sys/wait.h>

#include <poll.h>

#include <arpa/inet.h>

#define MAX\_BUFFER\_SIZE 1024

#define MAX\_CLIENTS 10

void sortArray(int\* array, int size)

{

std::sort(array, array + size);

}

int main()

{

int serverSocket, clientSockets[MAX\_CLIENTS];

struct sockaddr\_in serverAddress, clientAddress;

socklen\_t clientAddressLength = sizeof(clientAddress);

std::vector<pid\_t> childProcesses;

// Create server socket

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

if (serverSocket < 0)

{

std::cerr << "Error creating server socket" << std::endl;

return 1;

}

// Set up server address

serverAddress.sin\_family = AF\_INET;

serverAddress.sin\_port = htons(1234);

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

// Bind the server socket to the specified IP and port

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

{

std::cerr << "Error binding server socket" << std::endl;

return 1;

}

// Listen for incoming connections

if (listen(serverSocket, 5) < 0)

{

std::cerr << "Error listening for connections" << std::endl;

return 1;

}

std::cout << "Server is listening..." << std::endl;

// Initialize client sockets array

for (int i = 0; i < MAX\_CLIENTS; i++)

{

clientSockets[i] = -1;

}

struct pollfd fds[MAX\_CLIENTS + 1];

fds[0].fd = serverSocket;

fds[0].events = POLLIN;

int numClients = 0;

while (true)

{

int pollStatus = poll(fds, numClients + 1, -1);

if (pollStatus < 0)

{

std::cerr << "Error in poll" << std::endl;

return 1;

}

if (fds[0].revents & POLLIN)

{

// New connection

int newSocket = accept(serverSocket, (struct sockaddr\*)&clientAddress, &clientAddressLength);

if (newSocket < 0)

{

std::cerr << "Error accepting new connection" << std::endl;

return 1;

}

std::cout << "New connection, socket fd is " << newSocket << ", IP is " << inet\_ntoa(clientAddress.sin\_addr)

<< ", port is " << ntohs(clientAddress.sin\_port) << std::endl;

// Add new client socket to the array

for (int i = 0; i < MAX\_CLIENTS; i++)

{

if (clientSockets[i] == -1)

{

clientSockets[i] = newSocket;

fds[numClients + 1].fd = newSocket;

fds[numClients + 1].events = POLLIN;

numClients++;

break;

}

}

}

for (int i = 1; i < numClients + 1; i++)

{

if (fds[i].revents & POLLIN)

{

// Data received from a client

int clientSocket = fds[i].fd;

char buffer[MAX\_BUFFER\_SIZE];

// Receive data from the client

int bytesRead = read(clientSocket, buffer, sizeof(buffer));

if (bytesRead <= 0)

{

// Client disconnected

std::cout << "Client disconnected, socket fd is " << clientSocket << std::endl;

close(clientSocket);

fds[i].fd = -1;

std::swap(fds[i], fds[numClients]);

clientSockets[i - 1] = -1;

clientSockets[numClients - 1] = clientSocket;

numClients--;

continue;

}

// Process the received request

int array[MAX\_BUFFER\_SIZE];

std::stringstream ss(buffer);

int num;

int index = 0;

while (ss >> num)

{

array[index++] = num;

}

// Sort the array

sortArray(array, index);

// Convert the sorted array back to a string

std::stringstream response;

for (int j = 0; j < index; j++)

{

response << array[j] << " ";

}

// Send the sorted array back to the client

send(clientSocket, response.str().c\_str(), response.str().length(), 0);

}

}

}

// Close all client sockets

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

{

close(clientSockets[i]);

}

// Close the server socket

close(serverSocket);

return 0;

}

**Файл client.cpp**

#include <iostream>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

#include <cstring>

#include <arpa/inet.h>

#include <sstream>

#define MAX\_BUFFER\_SIZE 1024

int main()

{

int clientSocket;

struct sockaddr\_in serverAddress;

char buffer[MAX\_BUFFER\_SIZE];

// Create client socket

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

if (clientSocket < 0)

{

std::cerr << "Error creating client socket" << std::endl;

return 1;

}

// Set up server address

serverAddress.sin\_family = AF\_INET;

serverAddress.sin\_port = htons(1234);

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

{

std::cerr << "Invalid server address" << std::endl;

return 1;

}

// Connect to the server

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

{

std::cerr << "Error connecting to the server" << std::endl;

return 1;

}

std::cout << "Connected to the server" << std::endl;

// Read input from the user

std::cout << "Enter numbers separated by spaces: ";

std::string input;

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

// Send the request to the server

send(clientSocket, input.c\_str(), input.length() + 1, 0);

// Receive and print the response from the server

int bytesRead = read(clientSocket, buffer, sizeof(buffer));

if (bytesRead > 0)

{

std::cout << "Sorted array: " << buffer << std::endl;

}

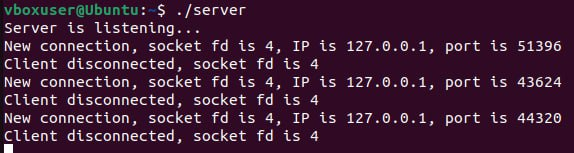
// Close the client socket

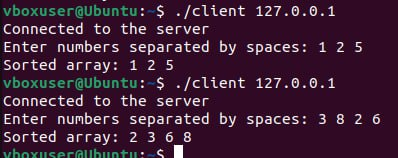
close(clientSocket);

return 0;

}

**Результати програми та тестові прилади**





**Висновки**

У результаті виконання лабораторної роботи набуто навичок створення багатопотокових серверних застосувань, розроблено програму, яка реалізує взаємодію клієнтів і сервера через сокети з використанням потоків.