**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

**ІКНІ**

Катедра **ПЗ**



**ЗВІТ**

До лабораторної роботи №9

**На тему:** *“*Організація взаємодії між процесами*”*

**Лекторка:**

доц. каф. ПЗ

Грицай О. Д.

**Виконав:**

ст. гр. ПЗ-24

Скрипник М. В.

**Прийняв:**

доц. каф. ПЗ

Горечко О. М.

Львів – 2023

**Тема**: Організація взаємодії між процесами.

**Мета:** Ознайомитися зі способами міжпроцесної взаємодії. Ознайомитися з класичним прикладом взаємодії між процесами на прикладі задачі «виробник – споживач». Навчитися працювати із процесами з використанням способів міжпроцесної взаємодії, синхронізувати їхню роботу.

**Теоретичні відомості**

Існує досить великий клас засобів ОС, за допомогою яких забезпечується взаємна синхронізація процесів і потоків. Потреба в синхронізації потоків виникає тільки в мультипрограмній ОС і залежить від спільного використання апаратних та інформаційних ресурсів обчислювальної системи. Синхронізація потрібна для запобігання перегонам та безвиході під час обміну даними між потоками, поділу даних, доступу до процесора і пристроїв введення-виведення.

У багатьох ОС ці засоби називаються **засобами міжпроцесної взаємодії** – **Inter Process Communications (IPC)**, що відображає історичну первинність поняття процес відносно поняття потік. Зазвичай до засобів IPC належать не тільки засоби міжпроцесної синхронізації, але й засоби обміну даними. Будь-яка взаємодія процесів або потоків залежить від їх синхронізації, яка полягає в узгодженні їх швидкостей через припинення потоку до настання деякої події й подальшої його активізації під час настання цієї події. Синхронізація лежить в основі будь-якої взаємодії потоків, незалежно від того, чи пов’язана ця взаємодія з розподілом ресурсів або з обміном даними. Наприклад, потік-одержувач повинен звертатися за даними тільки після того, як вони поміщені в буфер потоком-відправником. Якщо ж потік-одержувач звернувся до даних до моменту їх надходження в буфер, то він має бути припинений. Синхронізація також потрібна у разі спільного використання апаратних ресурсів. Наприклад, коли активному потоку потрібен доступ до послідовного порту, а з цим портом у монопольному режимі працює інший потік, який перебуває у стані очікування, то ОС припиняє активний потік і не активізує його доти, доки потрібний йому порт не звільниться. Часто потрібна також синхронізація з подіями, які не належать до обчислювальної системи, наприклад реакція на натискання комбінації клавіш **Ctrl+C**. Для синхронізації потоків прикладних програм програміст може використовувати як власні засоби та прийоми синхронізації, так і засоби ОС. Наприклад, два потоки одного прикладного процесу можуть координувати свою роботу за допомогою доступної для них обох глобальної логічної змінної, яка набуває значення одиниці при здійсненні деякої події, наприклад вироблення одним потоком даних, потрібних для продовження роботи іншого. Однак у багатьох випадках більш ефективними або навіть єдино можливими є засоби синхронізації, що надаються ОС у формі системних викликів. Так, потоки, що належать різним процесам, не мають можливості втручатися будь-яким чином у роботу один одного. Без посередництва ОС вони не можуть призупинити один одного або сповістити про подію, що відбулася.

Засоби синхронізації використовуються ОС не тільки для синхронізації прикладних процесів, але й для її внутрішніх потреб. Зазвичай розробники ОС надають у розпорядження прикладних і системних програмістів широкий спектр засобів синхронізації. Ці засоби можуть утворювати ієрархію, коли на основі більш простих засобів будуються більш складні, а також бути функціонально спеціалізованими, наприклад засоби для синхронізації потоків одного процесу, засоби для синхронізації потоків різних процесів під час обміну даними і т. ін.

Часто функціональні можливості різних системних викликів синхронізації перекриваються, тому для вирішення одного завдання програміст може скористатися кількома викликами залежно від особистих переваг.

Ситуації, коли процесам доводиться взаємодіяти:

– передавання інформації від одного процесу до іншого;

– контроль над діяльністю процесів (наприклад, коли вони змагаються за один ресурс);

– узгодження дій процесів (наприклад, коли один процес доставляє дані, а інший їх виводить на друк. Якщо узгодженості не буде, то другий процес може почати друк раніше, ніж надійдуть дані).

Два останні випадки стосуються і потоків. У першому випадку потоки не мають проблем, тому що вони використовують загальний адресний простір.

Існує дві основні моделі міжпроцесорної комунікації: спільна пам'ять та передача повідомлень. У моделі спільної пам’яті встановлюється область пам’яті, яка поділяється процесами співпраці. Потім процеси можуть обмінюватися інформацією, читаючи та записуючи дані в спільний регіон. У моделі передачі повідомлень, спілкування відбувається за допомогою повідомлень, що обмінюються між взаємодіючими процесами.

Міжпроцесовий зв’язок із використанням спільної пам'яті вимагає комунікаційних процесів для встановлення області спільної пам'яті. Зазвичай область спільної пам’яті міститься в адресному просторі процесу, створюючи сегмент спільної пам’яті. Як правило, операційна система намагається запобігти доступу одного процесу до пам’яті іншого процесу. Загальна пам’ять вимагає, щоб два більше обробляли процес, щоб зменшити це обмеження. Потім вони можуть обмінюватися інформацією, читаючи та записуючи спільні ділянки

Форма даних та місцеположення визначаються цими процесами і не підконтрольні операційній системі. Процеси також відповідають за те, щоб вони не записували в одне місце одночасно.

Щоб проілюструвати концепцію процесів співпраці, розглянемо проблему виробник-споживач, яка є звичайною парадигмою для співпрацюючих процесів. Процес виробника виробляє інформацію, яку споживає споживчий процес. Щоб дозволити процесам виробника та споживача одночасно працювати, ми повинні мати доступний буфер елементів, який може бути заповнений виробником і спорожнений споживачем. Цей буфер розміщуватиметься в області пам’яті, яка поділяється між виробництвом та споживчими процесами. Виробник може вироблятись між тим, як споживатиме консолідований інший спосіб. Виробник і споживач повинні бути синхронізовані, щоб споживач не намагався споживати товар, який ще не був вироблений. Можна використовувати два типи буферів.

Незв'язаний буфер не обмежує розмір буфера. Споживачеві, можливо, доведеться чекати нових товарів, але виробник завжди може випускати нові товари. Обмежений буфер передбачає фіксований розмір буфера. У цьому випадку споживач повинен чекати, якщо буфер порожній, а виробник повинен чекати, якщо буфер заповнений.

**Pipes**

Існує два типи труб для двостороннього зв'язку: анонімні труби та названі труби. Анонімні труби дозволяють пов'язаним процесам передавати інформацію один одному. Як правило, анонімна труба використовується для перенаправлення стандартного вводу або виводу дочірнього процесу, щоб він міг обмінюватися даними зі своїм батьківським процесом. Для обміну даними в обох напрямках (дуплексна робота) необхідно створити дві анонімні труби. Батьківський процес записує дані в одну трубку за допомогою своєї ручки запису, тоді як дочірній процес зчитує дані з цієї трубки за допомогою своєї ручки читання. Аналогічно, дочірній процес записує дані в іншу трубку і батьківський процес зчитує з нього.

Анонімні труби не можуть використовуватися через мережу, а також не можуть використовуватися між спорідненими процесами. Названі труби використовуються для передачі даних між процесами, які не пов'язані між собою, і між процесами на різних комп'ютерах. Як правило, сервер з іменованим каналом створює іменовану трубу з добре відомим ім'ям або ім'ям, яке слід повідомити своїм клієнтам. Клієнтський процес з іменованою трубкою, який знає ім'я труби, може відкрити інший його кінець, за умови обмежень доступу, визначених сервером процес з іменованим каналом. Після того як і сервер, і клієнт підключилися до труби, вони можуть обмінюватися даними, виконуючи операції читання і запису на трубі. Анонімні канали забезпечують ефективний спосіб перенаправлення стандартного вводу або виводу на дочірні процеси на одному комп’ютері. Названі труби забезпечують простий інтерфейс програмування для передачі даних між двома процесами, незалежно від того, чи вони перебувають на одному комп’ютері чи по мережі.

**Завдання**

1. Реалізувати алгоритм моделювання заданої задачі за допомогою окремих

процесів згідно індивідуального завдання.

2. Реалізувати синхронізацію роботи процесів.

3. Забезпечити зберігання результатів виконання завдання.

4. Результати виконання роботи відобразити у звіті.

**Індивідуальне завдання**

**Варіант - 5**

Створити програму-чат спілкування між процесами. Процес, що запускає чат, надає можливість вибору варіанту методу спілкування з 2-3 методів міжпроцесної взаємодії.

**Хід роботи**

Код сервера

**Server.h**

#pragma once

#include <iostream>

#include <windows.h>

#include <thread>

#include <vector>

#pragma comment(lib, "ws2\_32.lib")

#define DEFAULT\_PORT 1488

#define BUFFER\_SIZE 1024

class Server {

public:

Server();

~Server();

void start();

void stop();

private:

SOCKET serverSocket;

std::vector<std::thread> clientThreads;

void clientHandler(SOCKET clientSocket);

};

**Server.срр**

#include "Server.h"

#include "Windows.h"

HANDLE hFile = CreateFileA("output.txt", GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

HANDLE clientThreadsMutex;

Server::Server() {

WSADATA wsaData;

if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {

std::cerr << "WSAStartup failed.\n";

exit(EXIT\_FAILURE);

}

clientThreadsMutex = CreateMutex(NULL, FALSE, NULL);

if (clientThreadsMutex == NULL) {

std::cerr << "Mutex creation failed.\n";

closesocket(serverSocket);

WSACleanup();

exit(EXIT\_FAILURE);

}

if ((serverSocket = socket(AF\_INET, SOCK\_STREAM, 0)) == INVALID\_SOCKET) {

std::cerr << "Socket creation failed.\n";

WSACleanup();

exit(EXIT\_FAILURE);

}

struct sockaddr\_in serverAddr;

serverAddr.sin\_family = AF\_INET;

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

serverAddr.sin\_port = htons(DEFAULT\_PORT);

if (bind(serverSocket, (struct sockaddr\*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR) {

std::cerr << "Binding failed.\n";

closesocket(serverSocket);

WSACleanup();

exit(EXIT\_FAILURE);

}

if (listen(serverSocket, SOMAXCONN) == SOCKET\_ERROR) {

std::cerr << "Listening failed.\n";

closesocket(serverSocket);

WSACleanup();

exit(EXIT\_FAILURE);

}

std::cout << "Server is listening on port " << DEFAULT\_PORT << "...\n";

char buf[256];

sprintf\_s(buf, sizeof(buf), "Server is listening on port %d...\n", DEFAULT\_PORT);

// Write the formatted string to the file

DWORD bytesWritten;

if (!WriteFile(hFile, buf, strlen(buf), &bytesWritten, NULL))

{

std::cerr << "Error writing to file. Error code: " << GetLastError() << std::endl;

CloseHandle(hFile);

exit(1);

}

}

Server::~Server() {

CloseHandle(clientThreadsMutex);

closesocket(serverSocket);

WSACleanup();

}

void Server::start() {

while (true) {

SOCKET clientSocket;

struct sockaddr\_in clientAddr;

int addrLen = sizeof(clientAddr);

if ((clientSocket = accept(serverSocket, (struct sockaddr\*)&clientAddr, &addrLen)) == INVALID\_SOCKET) {

std::cerr << "Accepting client failed.\n";

break;

}

std::cout << "Client connected.\n";

char buf[256];

sprintf\_s(buf, sizeof(buf), "Client connected.\n");

// Write the formatted string to the file

DWORD bytesWritten;

if (!WriteFile(hFile, buf, strlen(buf), &bytesWritten, NULL))

{

std::cerr << "Error writing to file. Error code: " << GetLastError() << std::endl;

CloseHandle(hFile);

exit(1);

}

WaitForSingleObject(clientThreadsMutex, INFINITE);

std::thread clientThread(&Server::clientHandler, this, clientSocket);

ReleaseMutex(clientThreadsMutex);

clientThread.detach();

clientThreads.push\_back(std::move(clientThread));

if (clientThreads.size() >= 10) {

std::cout << "Reached maximum number of clients (10). Closing the server.\n";

break;

}

}

}

//some comment sd

void Server::stop() {

closesocket(serverSocket);

WSACleanup();

}

void Server::clientHandler(SOCKET clientSocket) {

char buffer[BUFFER\_SIZE];

while (true) {

int bytesReceived = recv(clientSocket, buffer, BUFFER\_SIZE, 0);

if (bytesReceived == SOCKET\_ERROR || bytesReceived == 0) {

std::cout << "Client disconnected.\n";

char buf[256];

sprintf\_s(buf, sizeof(buf), "Client disconnected.\n");

// Write the formatted string to the file

DWORD bytesWritten;

if (!WriteFile(hFile, buf, strlen(buf), &bytesWritten, NULL))

{

std::cerr << "Error writing to file. Error code: " << GetLastError() << std::endl;

CloseHandle(hFile);

exit(1);

}

break;

}

buffer[bytesReceived] = '\0';

std::cout << "Received: " << buffer << std::endl;

char buf[256];

sprintf\_s(buf, sizeof(buf), "Received: %s\n", buffer);

// Write the formatted string to the file

DWORD bytesWritten;

if (!WriteFile(hFile, buf, strlen(buf), &bytesWritten, NULL))

{

std::cerr << "Error writing to file. Error code: " << GetLastError() << std::endl;

CloseHandle(hFile);

exit(1);

}

}

WaitForSingleObject(clientThreadsMutex, INFINITE);

closesocket(clientSocket);

ReleaseMutex(clientThreadsMutex);

}

**Client.h**

#pragma once

#include <iostream>

//#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <string>

#pragma comment(lib, "ws2\_32.lib")

class Client {

public:

void connectToServer(const std::string& serverAddress, int port);

void sendMessage(const std::string& message);

void setUsername(const std::string& name);

void chooseConnectionMethod(int method);

void closeConnection();

private:

SOCKET clientSocket;

std::string username;

std::string selectedMethod;

};

**Client.cpp**

#include "Client.h"

void Client::connectToServer(const std::string& serverAddress, int port) {

WSADATA wsaData;

if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {

std::cerr << "WSAStartup failed.\n";

return;

}

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

if (clientSocket == INVALID\_SOCKET) {

std::cerr << "Socket creation failed.\n";

WSACleanup();

return;

}

struct sockaddr\_in serverAddr;

serverAddr.sin\_family = AF\_INET;

inet\_pton(AF\_INET, serverAddress.c\_str(), &serverAddr.sin\_addr);

serverAddr.sin\_port = htons(port);

if (connect(clientSocket, (struct sockaddr\*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR) {

std::cerr << "Socket connection failed.\n";

closesocket(clientSocket);

WSACleanup();

return;

}

}

void Client::sendMessage(const std::string& message) {

std::string fullMessage = selectedMethod + username + ": " + message;

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

}

void Client::setUsername(const std::string& name) {

username = name;

}

void Client::chooseConnectionMethod(int method) {

if (method == 1) {

selectedMethod = "Socket Connection";

}

else if (method == 2) {

selectedMethod = "Pipe Connection";

}

}

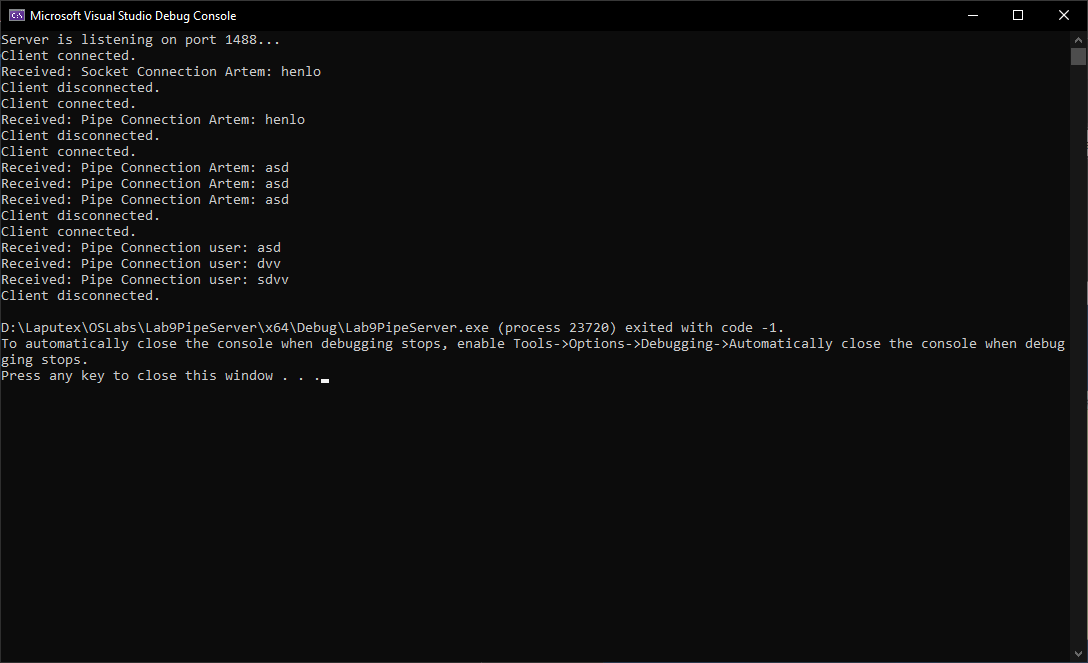
void Client::closeConnection() {

closesocket(clientSocket);

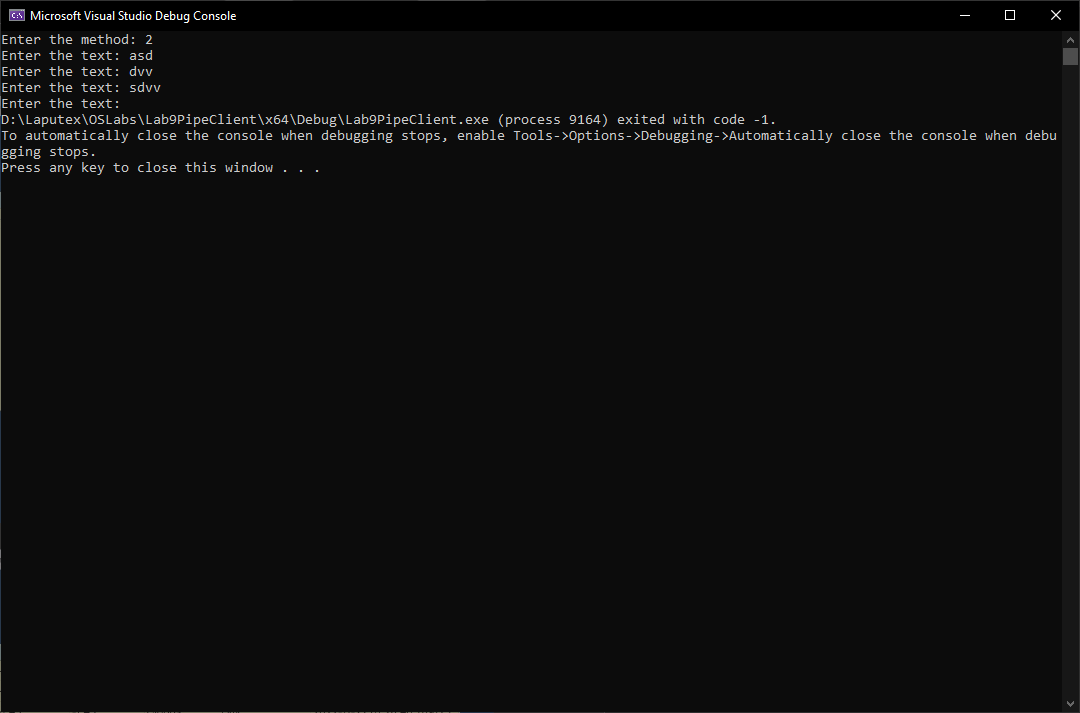
WSACleanup();

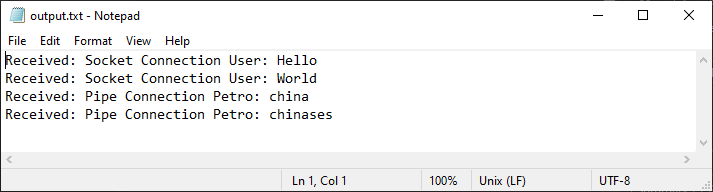
}

**Результати**



*Рис.1 Загальний інтерфейс програми-сервера*

*Рис.2 Сторона клієнта*

*Рис.3 Вивід у файл*

**Висновки**

В ході лабораторної роботи ми отримали поглиблене розуміння принципів міжпроцесної взаємодії, освоїли пайпи та сокети. Реалізація класичного прикладу - сервера та клієнта, дозволила нам застосувати знання на практиці та отримати цінний досвід у програмуванні міжпроцесових взаємодій.

Розподіл роботи в проекті:

1. [Код сервера](#Сервер) – Губик Артем

Код сервера було написано мовою **C++**, а для міжпроцесорної взаємодії було використано пайп і сокет. Використано Winsock API для роботи з сокетами та мережевим взаємодією.

Використання пайпів для міжпроцесорної взаємодії.

1. [Код](#Клієнт)  сокет клієнта – Лащук Роман

Клієнта було написано в програмі Visual Studio мовою С++.

Програма дозволяє клієнтам авторизуватись та надсилати повідомлення. Як механізм міжпроцесорної взаємодії було використано сокет. Для роботи були використані відповідні **WinAPI** функції.

1. [[Код пайпу](#Клієнт)](#Qt)  – Скрипник Максим

Розробив ідентичний до сокетів але використовуючи механізм міжпроцесорної взаємодії пайп. Для роботи були використані відповідні **WinAPI** функції.

1. Звіт та Презентація – Крупник Іван