МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут **ІКНІ**

Кафедра **ПЗ**

ЗВІТ

До лабораторної роботи № 9

З дисципліни: *“Операційні системи”*

На тему: *“Організація взаємодії між процесами”*

Лектор:
ст. викл. каф. ПЗ
Грицай О. Д.

Виконали:
ст. гр. ПЗ-22
Чаус О. М.
Мартиняк А. В.
Снісар В. І.
Яворська Р. Т.

Прийняла:
ст. викл. каф. ПЗ
Грицай О. Д.

« ____ » _____ 2022 р.
∑= _____ .....:

Львів – 2022

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.

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.

Варіант №7

Реалізувати міжпроцесну взаємодію за допомогою сокетів. Один із процесів має бути сервером, який дозволяє процесам-клієнтам підписатись/відписатись на один із сервісів розсилки (щогодинний прогноз погоди, щохвилинний курс акцій, щоденний курс валют). Для збереження інформації на сервері можна використати бази даних.

Теоретичні відомості

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.

Передавання інформації від одного процесу до іншого. Інформація може передаватися кількома способами:

– колективна пам’ять;

– канали (труби) – це псевдофайл, який один процес записує, а інший зчитує.

Схема для каналу:

– *сокети* – підтримувані ядром механізми, що приховують особливості середовища і дозволяють взаємодіяти процесам, як на одному комп’ютері, так і в мережі.

Схема для сокетів:

- поштові скриньки (тільки у Windows) – однонапрямлені з можливістю широкомовної розсилки;
- виклик віддаленої процедури – процес А може викликати процедуру в процесі В і отримувати назад дані.

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.

У WINDOWS підтримуються наступні *механізми міжпроцесної взаємодії*:

- Clipboard
- COM
- Data Copy
- DDE
- File Mapping
- Mailslots
- Pipes
- RPC
- Windows Sockets

Windows Sockets. Сокети Windows - це незалежний від протоколу інтерфейс. Він використовує можливості зв'язку базових протоколів. У Windows Sockets дескриптор сокета за бажанням може використовуватися як файловий дескриптор зі стандартними функціями вводу-виводу файлів.

Windows Sockets засновані на сокетах, вперше популяризованих Berkeley Software Distribution (BSD). Додаток, що використовує Windows Sockets, може зв'язуватися з іншими реалізаціями сокетів на інших типах систем.

Ключовий момент: Windows Sockets - це незалежний від протоколу інтерфейс, здатний підтримувати поточні та нові мережеві можливості. Для отримання додаткових відомостей дивіться

Хід роботи

1. Програмна реалізація клієнта:

```
#include <string>
#define WIN32_LEAN_AND_MEAN

#include <windows.h>
#include <processthreadsapi.h>
#include <winsock2.h>
#include <ws2tcpip.h>
```

```
#include <stdlib.h>
#include <stdio.h>
#include <tchar.h>
#include <strsafe.h>
#include <string>
#include <iostream>
#include "clientwindow.h"
#include "ui_clientwindow.h"

// Need to link with Ws2_32.lib, Mswsock.lib, and Advapi32.lib
#pragma comment (lib, "Ws2_32.lib")
#pragma comment (lib, "Mswsock.lib")
#pragma comment (lib, "AdvApi32.lib")

#define DEFAULT_BUFLEN 512
#define DEFAULT_PORT "27015"

WSADATA wsaData;
SOCKET ConnectSocket = INVALID_SOCKET;
struct addrinfo *result = NULL,
                *ptr = NULL,
                hints;
const char *sendbuf = "this is a test";
char recvbuf[DEFAULT_BUFLEN];
int iResult;
int recvbuflen = DEFAULT_BUFLEN;

bool is_launched = false;
bool is_subscribed_weather = false;
bool is_subscribed_stocks = false;
bool is_subscribed_currency = false;

Ui::ClientWindow *ui_global;

ClientWindow::ClientWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::ClientWindow)
{
    ui->setupUi(this);
    ui_global = ui;
}

ClientWindow::~ClientWindow()
{
    delete ui;
}

HANDLE thread_handle;
DWORD thread_id;

int handle_startup_work()
{
    // Initialize Winsock
    iResult = WSAStartup(MAKEWORD(2,2), &wsaData);

    ZeroMemory( &hints, sizeof(hints) );
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_protocol = IPPROTO_TCP;

    // Resolve the server address and port
    iResult = getaddrinfo("127.0.0.1", DEFAULT_PORT, &hints, &result);

    // Attempt to connect to an address until one succeeds
    for(ptr=result; ptr != NULL ;ptr=ptr->ai_next) {
        // Create a SOCKET for connecting to server
        ConnectSocket = socket(ptr->ai_family, ptr->ai_socktype,
```

```
            ptr->ai_protocol);

        // Connect to server.
        iResult = connect( ConnectSocket, ptr->ai_addr, (int)ptr->ai_addrlen);
        if (iResult == SOCKET_ERROR) {
            closesocket(ConnectSocket);
            ConnectSocket = INVALID_SOCKET;
            continue;
        }
        break;
    }

    freeaddrinfo(result);

    if (ConnectSocket == INVALID_SOCKET) {
        printf("Unable to connect to server!\n");
        WSACleanup();
        return 1;
    }

    return 0;
}

int send_data(const char* message)
{
    // Send an initial buffer
    iResult = send( ConnectSocket, message, (int)strlen(sendbuf), 0 );
    if (iResult == SOCKET_ERROR) {
        printf("send failed with error: %d\n", WSAGetLastError());
        closesocket(ConnectSocket);
        WSACleanup();
        return 1;
    }

    return 0;
}

int close_socket()
{
    // shutdown the connection since no more data will be sent
    iResult = shutdown(ConnectSocket, SD_SEND);
    if (iResult == SOCKET_ERROR) {
        printf("shutdown failed with error: %d\n", WSAGetLastError());
        closesocket(ConnectSocket);
        WSACleanup();
        return 1;
    }

    // cleanup
    closesocket(ConnectSocket);
    WSACleanup();

    return 0;
}

DWORD WINAPI receive_data_from_server()
{
    // Receive until the peer closes the connection
    do {
        iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);
        if ( iResult > 0 ){
            std::string result = recvbuf;
            char symbol_to_check = result[0];
            result = result.substr(1);
            if (symbol_to_check == ((char*)"w")[0])
            {
                std::string delimiter = "#";
```

```
                size_t pos = 0;
                std::string token;
                std::string array[2];
                int i = 0;
                while ((pos = result.find(delimiter)) != std::string::npos) {
                    token = result.substr(0, pos);
                    array[i] = token;
                    std::cout << token << std::endl;
                    result.erase(0, pos + delimiter.length());
                    i++;
                }
                ui_global->dateTimeLabel-
>setText(QString::fromStdString(array[0]));
                ui_global->tempLabel->setText(QString::fromStdString(result));
            } else if (symbol_to_check == ((char*)"s")[0])
            {
                std::string delimiter = "#";
                size_t pos = 0;
                std::string token;
                std::string array[3];
                int i = 0;
                while ((pos = result.find(delimiter)) != std::string::npos) {
                    token = result.substr(0, pos);
                    array[i] = token;
                    std::cout << token << std::endl;
                    result.erase(0, pos + delimiter.length());
                    i++;
                }
                ui_global->appleLabel-
>setText(QString::fromStdString(array[0]));
                ui_global->teslaLabel-
>setText(QString::fromStdString(array[1]));
                ui_global->amazonLabel-
>setText(QString::fromStdString(result));
            } else if (symbol_to_check == ((char*)"c")[0])
            {
                std::string delimiter = "#";
                size_t pos = 0;
                std::string token;
                std::string array[3];
                int i = 0;
                while ((pos = result.find(delimiter)) != std::string::npos) {
                    token = result.substr(0, pos);
                    array[i] = token;
                    std::cout << token << std::endl;
                    result.erase(0, pos + delimiter.length());
                    i++;
                }
                ui_global->dollarLabel-
>setText(QString::fromStdString(array[0]));
                ui_global->euroLabel-
>setText(QString::fromStdString(array[1]));
                ui_global->poundLabel->setText(QString::fromStdString(result));
            }
        }
        else if ( iResult == 0 )
            printf("Connection closed\n");
        else
            printf("recv failed with error: %d\n", WSAGetLastError());

    } while( iResult > 0 );
    return 0;
}


void ClientWindow::on_subscribeButton_clicked()
{
```

```
    if ((!is_subscribed_weather) && (!is_subscribed_stocks) &&
(!is_subscribed_currency))
    {
        handle_startup_work();
        thread_handle = CreateThread(NULL, 0,
(LPTHREAD_START_ROUTINE)receive_data_from_server, NULL, 0, &thread_id);
    }
    std::string requestStr = "s";
    if(ui->weatherCheckBox->isChecked()){
            QStringList citiesList;
            QString choiceCity = "Львів";
            bool ok = false;
            citiesList << "Львів" << "Київ" << "Івано-Франківськ" << "Харків"
<< "Дніпро";
            choiceCity = QInputDialog::getItem(this, "", "Оберіть місто,
обновлення погоди\nз якого ви хочете бачити: ",
                                               citiesList, false ,&ok);
            requestStr += "1";
            ui->cityLabel->setText(choiceCity);
    } else requestStr += "0";
    requestStr += (ui->stocksCheckBox->isChecked()) ? "1" : "0";
    requestStr += (ui->currencyCheckBox->isChecked()) ? "1" : "0";
    if (ui->stocksCheckBox->isChecked())
    {
        is_subscribed_stocks = true;
    }
    if (ui->currencyCheckBox->isChecked())
    {
        is_subscribed_currency = true;
    }
    if (ui->weatherCheckBox->isChecked())
    {
        is_subscribed_weather = true;
    }
    const char* request = requestStr.c_str();
    send_data(request);
}


void ClientWindow::on_unsubscribeButton_clicked()
{
    if (ui->weatherCheckBox->isChecked())
    {
        is_subscribed_weather = false;
        ui->dateTimeLabel->setText("");
        ui->cityLabel->setText("");
        ui->tempLabel->setText("");
    }
    if (ui->stocksCheckBox->isChecked())
    {
        is_subscribed_stocks = false;
        ui->teslaLabel->setText("");
        ui->appleLabel->setText("");
        ui->amazonLabel->setText("");
    }
    if (ui->currencyCheckBox->isChecked())
    {
        is_subscribed_currency = false;
        ui->dollarLabel->setText("");
        ui->euroLabel->setText("");
        ui->poundLabel->setText("");
    }
    std::string request = std::string("u") + ((ui->weatherCheckBox-
>isChecked()) ? "1" : "0") +
        ((ui->stocksCheckBox->isChecked()) ? "1" : "0") +
        ((ui->currencyCheckBox->isChecked()) ? "1" : "0");
    send_data(request.c_str());
```

```
    if ((!is_subscribed_weather) && (!is_subscribed_stocks) &&
(!is_subscribed_currency))
    {
        TerminateThread(thread_handle, 0);
        close_socket();
    }
}
```

2. Реалізація Сервера:

```
#include <winsock2.h>
#include <iostream>
#include <vector>
#include <string>
#include <time.h>
#include <cstdlib>
using namespace std;

#define MAX_CLIENTS 10
#define DEFAULT_BUFLEN 4096

#pragma comment(lib, "ws2_32.lib") // Winsock library
#pragma warning(disable:4996)

SOCKET server_socket;

vector<string> history;


typedef struct SocketData {
    SOCKET socket;
} SOCKET_DATA, * SOCKET_DATA_PTR;


bool is_avaliable = true;
int send_data(SOCKET socket, const char* message)
{
    // Send an initial buffer
    while (!is_avaliable)
    {
        Sleep(100);
    }
    is_avaliable = false;
    send(socket, message, (int)strlen(message), 0);
    is_avaliable = true;

    return 0;
}



DWORD WINAPI subscribe_weather(LPVOID lpRaram)
{
    SOCKET_DATA_PTR Args = SOCKET_DATA_PTR(lpRaram);
    SOCKET socket = Args->socket;
    //delete Args;
    while (true)
    {
        time_t theTime = time(NULL);
        struct tm* aTime = localtime(&theTime);

        int day = aTime->tm_mday;
        int month = aTime->tm_mon + 1; // Month is 0 - 11, add 1 to get a jan-dec 1-12
concept
        int year = aTime->tm_year + 1900; // Year is # years since 1900
        int hour = aTime->tm_hour;
        int minutes = aTime->tm_min;
```

```
            int seconds = aTime->tm_sec;
            string weather_string = "w" + to_string(year) + "." +
                  to_string(month) + "." +
                  to_string(day) + "|" +
                  to_string(hour) + ":" +
                  to_string(minutes) + ":" +
                  to_string(seconds) + "#" + to_string((float(rand()) / float((RAND_MAX)) *
30.0));

            send_data(socket, weather_string.c_str());

            Sleep(15000);
      }
}


DWORD WINAPI subscribe_stocks(LPVOID lpRaram)
{
      SOCKET_DATA_PTR Args = SOCKET_DATA_PTR(lpRaram);
      SOCKET socket = Args->socket;
      //delete Args;
      while (true)
      {
            string stocks = "s" + to_string((float(rand()) / float((RAND_MAX)) * 140)) +
"#" +
                  to_string((float(rand()) / float((RAND_MAX)) * 180)) + "#" +
                  to_string((float(rand()) / float((RAND_MAX)) * 130));

            send_data(socket, stocks.c_str());

            Sleep(5000);
      }
}


DWORD WINAPI subscribe_currency(LPVOID lpRaram)
{
      SOCKET_DATA_PTR Args = SOCKET_DATA_PTR(lpRaram);
      SOCKET socket = Args->socket;
      //delete Args;

      while (true)
      {
            string currency = "c" + to_string((float(rand()) / float((RAND_MAX)) * 45)) +
"#" +
                  to_string((float(rand()) / float((RAND_MAX)) * 40)) + "#" +
                  to_string((float(rand()) / float((RAND_MAX)) * 50));

            send_data(socket, currency.c_str());

            Sleep(5000);
      }
}


DWORD WINAPI handle_socket_connection(LPVOID lpRaram)
{
      SOCKET_DATA_PTR Args = SOCKET_DATA_PTR(lpRaram);
      SOCKET socket = Args->socket;
      //delete Args;
      bool is_subscribed_weather = false;
      bool is_subscribed_stocks = false;
      bool is_subscribed_currency = false;
      char client_message[DEFAULT_BUFLEN];
      int iResult;
      HANDLE subscriptions[3] = { 0, 0, 0 };
      int weather = 0, stocks = 0, currency = 0;
      int* flag_weather = &weather;
      int* flag_stocks = &stocks;
```

```
    int* flag_currency = &currency;
    DWORD id;
    // Receive until the peer closes the connection
    do {
           iResult = recv(socket, client_message, DEFAULT_BUFLEN, 0);
           if (iResult > 0) {
                  std::string converted_string = client_message;
                  if (converted_string[0] == 's')
                  {
                         SOCKET_DATA args;
                         args.socket = socket;
                         if (converted_string[1] == ((char*)"1")[0])
                         {
                                *flag_weather = 0;
                                subscriptions[0] = CreateThread(NULL, 0,
(LPTHREAD_START_ROUTINE)subscribe_weather, &args, 0, &id);
                         }
                         if (converted_string[2] == ((char*)"1")[0])
                         {
                                *flag_stocks = 0;
                                subscriptions[1] = CreateThread(NULL, 0,
(LPTHREAD_START_ROUTINE)subscribe_stocks, &args, 0, &id);
                         }
                         if (converted_string[3] == ((char*)"1")[0])
                         {
                                *flag_currency = 0;
                                subscriptions[2] = CreateThread(NULL, 0,
(LPTHREAD_START_ROUTINE)subscribe_currency, &args, 0, &id);
                         }
                  }
                  else if (converted_string[0] == 'u')
                  {
                         if (converted_string[1] == ((char*)"1")[0])
                         {
                                TerminateThread(subscriptions[0], 0);
                         }
                         else if (converted_string[2] == ((char*)"1")[0])
                         {
                                TerminateThread(subscriptions[1], 0);
                         }
                         else if (converted_string[3] == ((char*)"1")[0])
                         {
                                TerminateThread(subscriptions[2], 0);
                         }
                  }
           }
           else if (iResult == 0)
                  printf("Connection closed\n");
           else
                  printf("recv failed with error: %d\n", WSAGetLastError());

    } while (iResult > 0);
    return 0;
}


int main() {
    system("title Server");

    puts("Start server... DONE.");
    WSADATA wsa;
    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0) {
           printf("Failed. Error Code: %d", WSAGetLastError());
           return 1;
    }

    // create a socket
    if ((server_socket = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET) {
           printf("Could not create socket: %d", WSAGetLastError());
```

```
		return 2;
	}
	// puts("Create socket... DONE.");

	// prepare the sockaddr_in structure
	sockaddr_in server;
	server.sin_family = AF_INET;
	server.sin_addr.s_addr = INADDR_ANY;
	server.sin_port = htons(27015);

	// bind socket
	if (bind(server_socket, (sockaddr*)&server, sizeof(server)) == SOCKET_ERROR) {
		printf("Bind failed with error code: %d", WSAGetLastError());
		return 3;
	}

	// puts("Bind socket... DONE.");

	// listen to incoming connections
	listen(server_socket, MAX_CLIENTS);

	// accept and incoming connection
	puts("Server is waiting for incoming connections...\nPlease, start one or more client-
side app.");

	// size of our receive buffer, this is string length
	// set of socket descriptors
	// fd means "file descriptors"
	fd_set readfds; // https://docs.microsoft.com/en-us/windows/win32/api/winsock/ns-
winsock-fd_set
	SOCKET client_socket[MAX_CLIENTS] = {};
	DWORD id;

	while (true) {
		// clear the socket fdset
		FD_ZERO(&readfds);

		// add master socket to fdset
		FD_SET(server_socket, &readfds);

		// add child sockets to fdset
		for (int i = 0; i < MAX_CLIENTS; i++) {
			SOCKET s = client_socket[i];
			if (s > 0) {
				FD_SET(s, &readfds);
			}
		}

		// wait for an activity on any of the sockets, timeout is NULL, so wait
indefinitely
		if (select(0, &readfds, NULL, NULL, NULL) == SOCKET_ERROR) {
			printf("select function call failed with error code : %d",
WSAGetLastError());
			return 4;
		}

		// if something happened on the master socket, then its an incoming connection
		SOCKET new_socket; // new client socket
		sockaddr_in address;
		int addrlen = sizeof(sockaddr_in);
		if (FD_ISSET(server_socket, &readfds)) {
			if ((new_socket = accept(server_socket, (sockaddr*)&address, &addrlen)) <
0) {
				perror("accept function error");
				return 5;
			}

			// inform server side of socket number - used in send and recv commands
```

```
                printf("New connection, socket fd is %d, ip is: %s, port: %d\n",
new_socket, inet_ntoa(address.sin_addr), ntohs(address.sin_port));

                // populate arguments for socket and start handling it
                SOCKET_DATA_PTR args = (SOCKET_DATA_PTR)HeapAlloc(GetProcessHeap(),
HEAP_ZERO_MEMORY,
                        sizeof(SOCKET_DATA));;
                args->socket = new_socket;
                CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)handle_socket_connection,
args, 0, &id);
            }
    }

    WSACleanup();
}
```

Результат виконання програми

Рис. 1 Клієнт

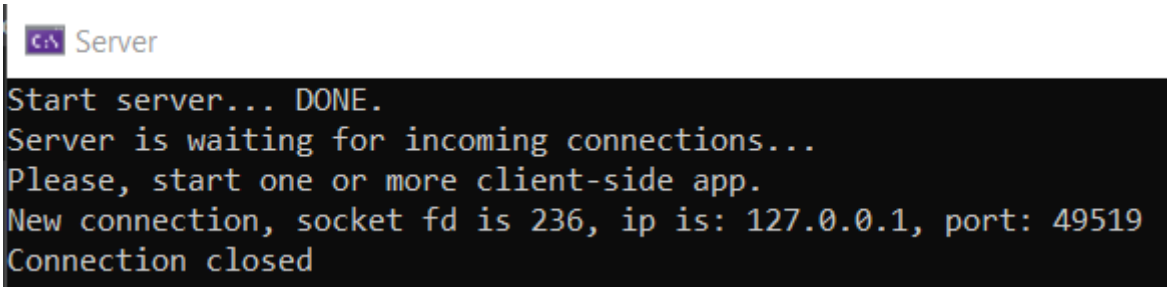

Рис. 2 Сервер

Висновки

На цій лабораторній роботі ми ознайомилися із способами міжпроцесної взаємодії та класичним прикладом «виробник – споживач». Навчилися працювати із процесами використання способів міжпроцесної взаємодії, синхронізували їх роботу. Реалізували

міжпроцесну взаємодію за допомогою сокетів. Один із процесів – сервер, який дозволяє клієнтам підписатись чи відписатись на один із сервісів розсилки (щогодинний прогноз погоди, щохвилинний курс акцій, щоденний курс валют). Програми створені на мові C++. Реалізував сервер розсилки, який здатний опрацьовувати декілька клієнтів.