**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

Інститут **ІКНІ**

Кафедра **ПЗ**

### ЗВІТ

До лабораторної роботи № 11

**З дисципліни:** *“Операційні системи”*

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

**Лектор:**

ст. викл. каф. ПЗ

Грицай О. Д.

**Виконали:**

ст. гр. ПЗ - 23

Дьяченко Д.І.

Михалевич П-І.В.

Юрга В.

**Прийняла:**

ст. викл. каф. ПЗ

Грицай О. Д.

« \_\_\_\_ » \_\_\_\_\_\_\_\_ 2021 р.

∑= \_\_\_\_\_ .

Львів – 2021

**Тема роботи:** Організація взаємодії між процесами

**Мета роботи:** 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.

***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.

У WINDOWS підтримуються наступні механізми міжпроцесної взаємодії:

● Clipboard

● COM

● Data Copy

● DDE

● File Mapping

● Mailslots

● Pipes

● RPC

● Windows Sockets

***Windows Sockets Windows Sockets*** - це незалежний від протоколу інтерфейс. Він використовує переваги комунікаційних можливостей базових протоколів. У Windows Sockets 2 дескриптор сокету може додатково використовуватися як файловий дескриптор зі стандартними функціями вводу / виводу файлів. Сокети Windows засновані на сокетах, вперше популяризованих Berkeley Software Distribution (BSD). Додаток, що використовує Windows Sockets, може спілкуватися з іншою реалізацією сокета в інших типах систем. Однак не всі постачальники транспортних послуг підтримують усі доступні варіанти. Windows Sockets - це незалежний від протоколу інтерфейс, здатний підтримувати поточні та нові можливості мереж.

**Завдання**

Варіант-7

Реалізувати міжпроцесну взаємодію використовуючи ***Сокети*** :

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. (робота в межах однієї системи)

**Хід роботи**

**Код програми:**

**Server.h:**#pragma once

#include "Includes.h"

#include "Weather.h"

#include "Currency.h"

#include "Shares.h"

void ServerThread(void\* Param);

class Server

{

public:

friend void ServerThread(void\* Param);

// Write seconds, later it will be multiplied by 1000.

// time\_1 - weather

// time\_2 - currency

// time\_3 - shares

Server(int time\_1 = 60, int time\_2 = 10, int time\_3 = 25);

void Start();

void InteructWithClient(int index);

~Server();

private:

SOCKET m\_sListen;

SOCKADDR\_IN m\_addr;

int m\_sizeofaddr = sizeof(m\_addr);

std::vector<SOCKET> Connections;

int Counter = 0;

Weather\* m\_weatherMail;

Currency\* m\_currencyMail;

Shares\* m\_sharesMail;

};

struct ClientIndex {

int index;

Server\* server;

};

**Server.cpp**

#pragma warning(disable: 4996)

#include "Server.h"

void ServerThread(void\* Param) {

ClientIndex\* structClient = (ClientIndex\*)Param;

structClient->server->InteructWithClient(structClient->index);

}

Server::Server(int time\_1, int time\_2, int time\_3) {

m\_weatherMail = new Weather(time\_1 \* 1000);

m\_currencyMail = new Currency(time\_2 \* 1000);

m\_sharesMail = new Shares(time\_3 \* 1000);

WSAData wsaData;

WORD DLLVersion = MAKEWORD(2, 1);

if (WSAStartup(DLLVersion, &wsaData) != 0) {

std::cout << "Error: WSAStartup() fails.\n" << std::endl;

exit(1);

}

m\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

m\_addr.sin\_port = htons(11111);

m\_addr.sin\_family = AF\_INET;

m\_sListen = socket(AF\_INET, SOCK\_STREAM, NULL);

}

void Server::Start() {

bind(m\_sListen, (SOCKADDR\*)&m\_addr, sizeof(m\_addr));

listen(m\_sListen, SOMAXCONN);

SOCKET newConnection;

for (int i = 0; i < 100; i++) {

newConnection = accept(m\_sListen, (SOCKADDR\*)&m\_addr, &m\_sizeofaddr);

if (newConnection == 0) {

std::cout << "Error: can't connect client.\n";

}

else {

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

SendString("Welcome to the mailing service)\nIf you want to subscribe on some services, choose onr from this:\n"

"1) Weather mailing;\n2) Currency mailing;\n3) Shares mailing;\n"

"Note: If you will need to unsubscribe from one of services write: \"Delete\" and number of service.", newConnection);

Connections.push\_back(newConnection);

Counter++;

ClientIndex structClient = ClientIndex{ i, this };

CreateThread(NULL, NULL, (LPTHREAD\_START\_ROUTINE)ServerThread, (void\*)(&structClient), NULL, NULL);

}

}

}

void Server::InteructWithClient(int index) {

int msg\_size;

while (true) {

int err = recv(Connections[index], (char\*)&msg\_size, sizeof(int), NULL);

if (err == SOCKET\_ERROR) {

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

break;

}

char\* msg = new char[msg\_size + 1];

msg[msg\_size] = '\0';

recv(Connections[index], msg, msg\_size, NULL);

if (!strcmp(msg, "1")) {

if (!m\_weatherMail->Find(Connections[index])) {

m\_weatherMail->AddSubscribed(Connections[index]);

SendString("You have been subscribed on weather mailing. Thanks!", Connections[index]);

}

else {

SendString("You have been already subscribed on weather mailing. There is no point to do it again.", Connections[index]);

}

}

else if (!strcmp(msg, "2")) {

if (!m\_currencyMail->Find(Connections[index])) {

m\_currencyMail->AddSubscribed(Connections[index]);

SendString("You have been subscribed on currency mailing. Thanks!", Connections[index]);

}

else {

SendString("You have been already subscribed on currency mailing. There is no point to do it again.", Connections[index]);

}

}

else if (!strcmp(msg, "3")) {

if (!m\_sharesMail->Find(Connections[index])) {

m\_sharesMail->AddSubscribed(Connections[index]);

SendString("You have been subscribed on shares mailing. Thanks!", Connections[index]);

}

else {

SendString("You have been already subscribed on shares mailing. There is no point to do it again.", Connections[index]);

}

}

else if (!strcmp(msg, "Delete 1")) {

SendString(m\_weatherMail->Delete(Connections[index]), Connections[index]);

}

else if (!strcmp(msg, "Delete 2")) {

SendString(m\_currencyMail->Delete(Connections[index]), Connections[index]);

}

else if (!strcmp(msg, "Delete 3")) {

SendString(m\_sharesMail->Delete(Connections[index]), Connections[index]);

}

else {

SendString("Unknown command, try to write it again.", Connections[index]);

}

delete[] msg;

}

}

Server::~Server() {

for (auto& con : Connections) {

SendString("Server turns off!\nExiting...", con);

closesocket(con);

}

closesocket(m\_sListen);

WSACleanup();

}

**Currency.h**

#pragma once

#include "Mailing.h"

#include "Includes.h"

void CurrencyThread(void\* Param);

class Currency : public Mailing

{

friend void CurrencyThread(void\* Param);

public:

Currency(int time = 25000);

virtual void SendMailThread() override;

};

**Currency.cpp**

#include "Currency.h"

Currency::Currency(int time) : Mailing(time) {

CreateThread(NULL, NULL, (LPTHREAD\_START\_ROUTINE)CurrencyThread, (void\*)this, NULL, NULL);

}

void Currency::SendMailThread() {

while (true) {

srand(time(NULL));

double euro = ((rand() % 150) / 100.) + 30.6;

std::ostringstream euroStr;

euroStr << std::fixed << std::setprecision(2) << euro;

double dollar = ((rand() % 150) / 100.) + 27.2;

std::ostringstream dollarStr;

dollarStr << std::fixed << std::setprecision(2) << dollar;

double chf = ((rand() % 150) / 100.) + 29.5;

std::ostringstream chfStr;

chfStr << std::fixed << std::setprecision(2) << chf;

std::string mes = "Currency:\n\t1 (EUR) - " + euroStr.str() + " (UAH)\n"

+ "\t1 (USD) - " + dollarStr.str() + " (UAH)\n"

+ "\t1 (CHF) - " + chfStr.str() + " (UAH)\n";

SendMail(mes);

Sleep(m\_period);

}

}

void CurrencyThread(void\* Param) {

Currency\* This = (Currency\*)Param;

This->SendMailThread();

}

**CustomFunctions.h**

#pragma once

#include <winsock2.h>

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

#include <unordered\_set>

void SendString(std::string mes, SOCKET Connection);

**CustomFunctons.cpp**

#include "CustomFunctions.h"

void SendString(std::string mes, SOCKET Connection) {

int msg\_size = mes.size();

send(Connection, (char\*)&msg\_size, sizeof(int), NULL);

send(Connection, mes.c\_str(), msg\_size, NULL);

}

**Mailing.h**

#pragma once

#include "Includes.h"

class Mailing

{

public:

Mailing(int time);

std::string Delete(SOCKET connection);

bool Find(SOCKET connection);

void AddSubscribed(SOCKET newClient);

virtual void SendMailThread() = 0;

void SendMail(std::string mes);

virtual ~Mailing() = default;

protected:

int m\_period;

std::unordered\_set<SOCKET> m\_subscribed;

};

**Mailing.cpp**

#include "Mailing.h"

Mailing::Mailing(int time) : m\_period(time) {

}

void Mailing::AddSubscribed(SOCKET newClient) {

m\_subscribed.insert(newClient);

}

void Mailing::SendMail(std::string mes) {

for (const auto& connection : m\_subscribed) {

SendString(mes, connection);

}

}

bool Mailing::Find(SOCKET connection) {

if (m\_subscribed.find(connection) != m\_subscribed.end()) {

return true;

}

else {

return false;

}

}

std::string Mailing::Delete(SOCKET connection) {

if (Find(connection)) {

m\_subscribed.erase(connection);

return "You have been successfully unsubscribed. Sorry for annoying you ;(";

}

else {

return "You haven't been subscribed on our service.";

}

}

**Shares.h**

#pragma once

#include "Mailing.h"

#include "Includes.h"

void SharesThread(void\* Param);

class Shares : public Mailing

{

friend void SharesThread(void\* Param);

public:

Shares(int time = 10000);

virtual void SendMailThread() override;

};

**Shares.cpp**

#include "Shares.h"

Shares::Shares(int time) : Mailing(time) {

CreateThread(NULL, NULL, (LPTHREAD\_START\_ROUTINE)SharesThread, (void\*)this, NULL, NULL);

}

void Shares::SendMailThread() {

while (true) {

srand(time(NULL));

double apple = ((rand() % 300) / 100.) + 161.6;

std::ostringstream appleStr;

appleStr << std::fixed << std::setprecision(2) << apple;

double microsoft = ((rand() % 700) / 100.) + 323.2;

std::ostringstream microsoftStr;

microsoftStr << std::fixed << std::setprecision(2) << microsoft;

double airbus = ((rand() % 300) / 100.) + 98.5;

std::ostringstream airbusStr;

airbusStr << std::fixed << std::setprecision(2) << airbus;

std::string mes = "Shares:\n\tApple - " + appleStr.str() + " (USD)\n"

+ "\tMicrosoft - " + microsoftStr.str() + " (USD)\n"

+ "\tAirbus - " + airbusStr.str() + " (USD)\n";

SendMail(mes);

Sleep(m\_period);

}

}

void SharesThread(void\* Param) {

Shares\* This = (Shares\*)Param;

This->SendMailThread();

}

**Source.cpp**

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

#pragma warning(disable: 4996)

#include "Server.h"

int main() {

Server server(30, 20, 10);

server.Start();

system("pause");

return 0;

}

**Weather.h**

#pragma once

#include "Mailing.h"

#include "Includes.h"

void WeatherThread(void\* Param);

class Weather : public Mailing

{

friend void WeatherThread(void\* Param);

public:

Weather(int time = 60000);

virtual void SendMailThread() override;

};

**Weather.cpp**

#include "Weather.h"

Weather::Weather(int time) : Mailing(time) {

CreateThread(NULL, NULL, (LPTHREAD\_START\_ROUTINE)WeatherThread, (void\*)this, NULL, NULL);

}

void Weather::SendMailThread() {

while (true) {

srand(time(NULL));

int temp = rand() % 45 - 20;

int humidity = rand() % 50 + 50;

double pressure = ((rand() % 300) / 10.0) + 985.;

std::ostringstream pressureStr;

pressureStr << std::fixed << std::setprecision(1) << pressure;

std::string mes = "Weather:\n\tTemperature: " + std::to\_string(temp)

+ " C\n\tHumidity: " + std::to\_string(humidity)

+ " %\n\tPressure: " + pressureStr.str() + " hPA\n";

SendMail(mes);

Sleep(m\_period);

}

}

void WeatherThread(void\* Param) {

Weather\* This = (Weather\*)Param;

This->SendMailThread();

}

**Includes.h**

#pragma once

#include <winsock2.h>

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

#include <unordered\_set>

#include <cmath>

#include <ctime>

#include <iomanip>

#include <sstream>

#include "CustomFunctions.h"

**Client.h**

#pragma once

#include "Includes.h"

void ClientThread(void\* Param);

class Client

{

friend void ClientThread(void\* Param);

public:

Client();

~Client() = default;

void Start();

private:

void ClientClassThread();

SOCKET m\_connection;

SOCKADDR\_IN m\_addr;

int m\_sizeofaddr = sizeof(m\_addr);

};

**Client.cpp**

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

#pragma warning(disable: 4996)

#include "Client.h"

Client::Client() {

WSAData wsaData;

WORD DLLVersion = MAKEWORD(2, 1);

if (WSAStartup(DLLVersion, &wsaData) != 0) {

std::cout << "Error" << std::endl;

exit(1);

}

m\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

m\_addr.sin\_port = htons(11111);

m\_addr.sin\_family = AF\_INET;

m\_connection = socket(AF\_INET, SOCK\_STREAM, NULL);

}

void Client::Start() {

if (connect(m\_connection, (SOCKADDR\*)&m\_addr, sizeof(m\_addr)) != 0) {

std::cout << "Error: failed connect to server.\n";

return;

}

std::cout << "Connected!\n";

CreateThread(NULL, NULL, (LPTHREAD\_START\_ROUTINE)ClientThread, (void \*)this, NULL, NULL);

std::string msg1;

while (true) {

std::getline(std::cin, msg1);

if (msg1 == "Exit") {

break;

}

int msg\_size = msg1.size();

send(m\_connection, (char\*)&msg\_size, sizeof(int), NULL);

int err = send(m\_connection, msg1.c\_str(), msg\_size, NULL);

if (err == SOCKET\_ERROR) {

std::cout << "\nServer turns off!\nExiting...";

break;

}

Sleep(10);

}

std::cout << "\nThanks for using our service!";

}

void Client::ClientClassThread() {

int msg\_size;

while (true) {

int err = recv(m\_connection, (char\*)&msg\_size, sizeof(int), NULL);

if (err == SOCKET\_ERROR) {

std::cout << "\nServer turns off!\nExiting...";

std::cout << "\nThanks for using our service!";

break;

}

char\* msg = new char[msg\_size + 1];

msg[msg\_size] = '\0';

recv(m\_connection, msg, msg\_size, NULL);

std::cout << msg << std::endl;

if (!strcmp(msg, "Server turns off!\nExiting...")) {

break;

}

delete[] msg;

}

}

void ClientThread(void\* Param) {

Client\* This = (Client\*)Param;

This->ClientClassThread();

}

**Client/Source.cpp**

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

#pragma warning(disable: 4996)

#include "Client.h"

int main(int argc, char\* argv[]) {

Client client;

client.Start();

std::cout << "\nThanks for using our service!";

system("pause");

return 0;

}

**Розподілення роботи:**

Дьяченко Данило – Реалізував дані у вигляді окремих класів, та підключив їх до основної програми, допомагав в розробленні міжпроцесної взаємодії. Оформив звіт.

Михалевич Павло-Іван - Розроблив частину головного процесу і реалізовує взаємодію між процесами за допомогою сокетів.

Юрга Володимир – Реалізував функції виводу даних, підписку та відписку від серверу. Оформив презентацію.

**Результати:**

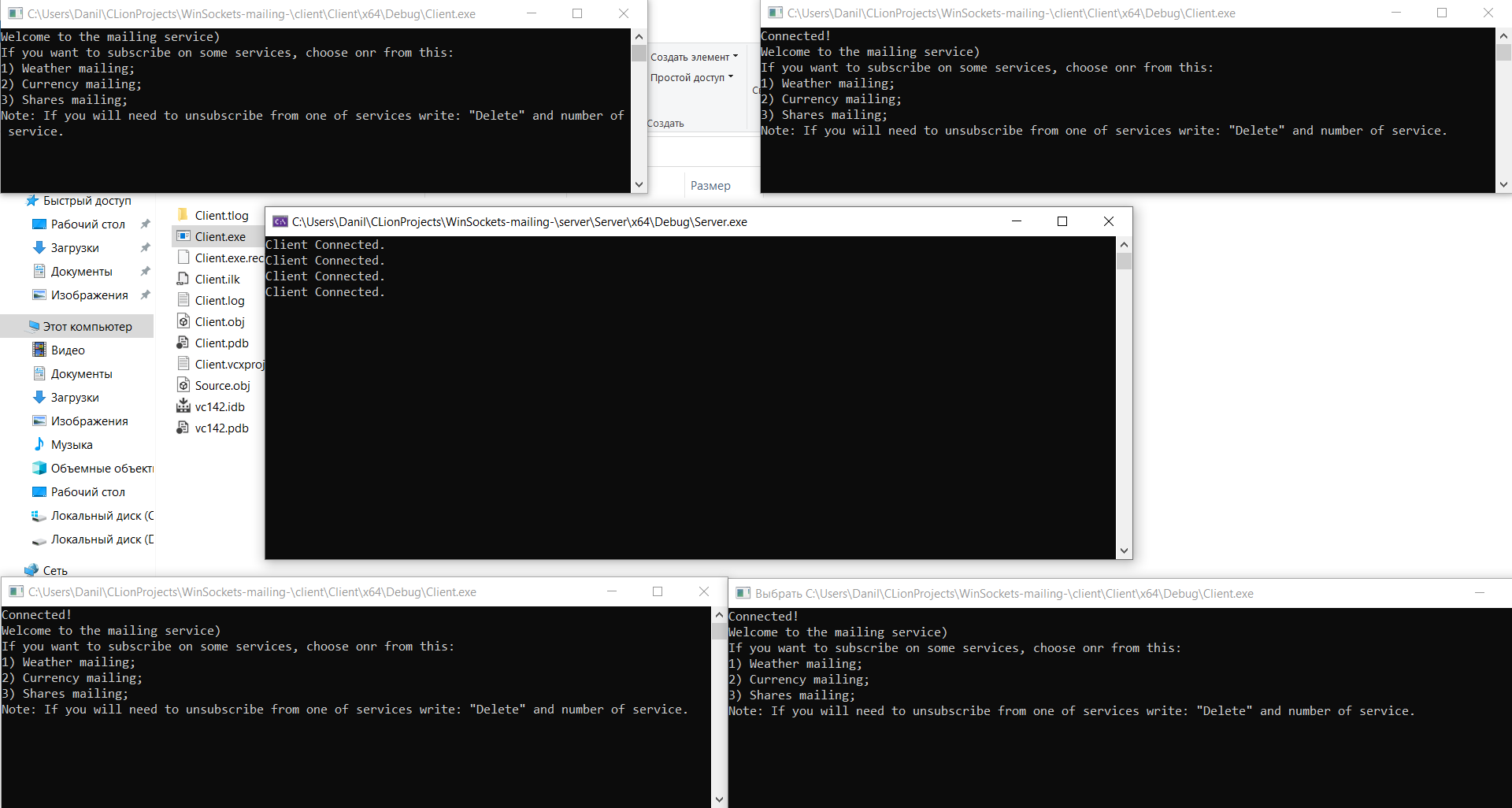


Рис 1. Запуск серверів

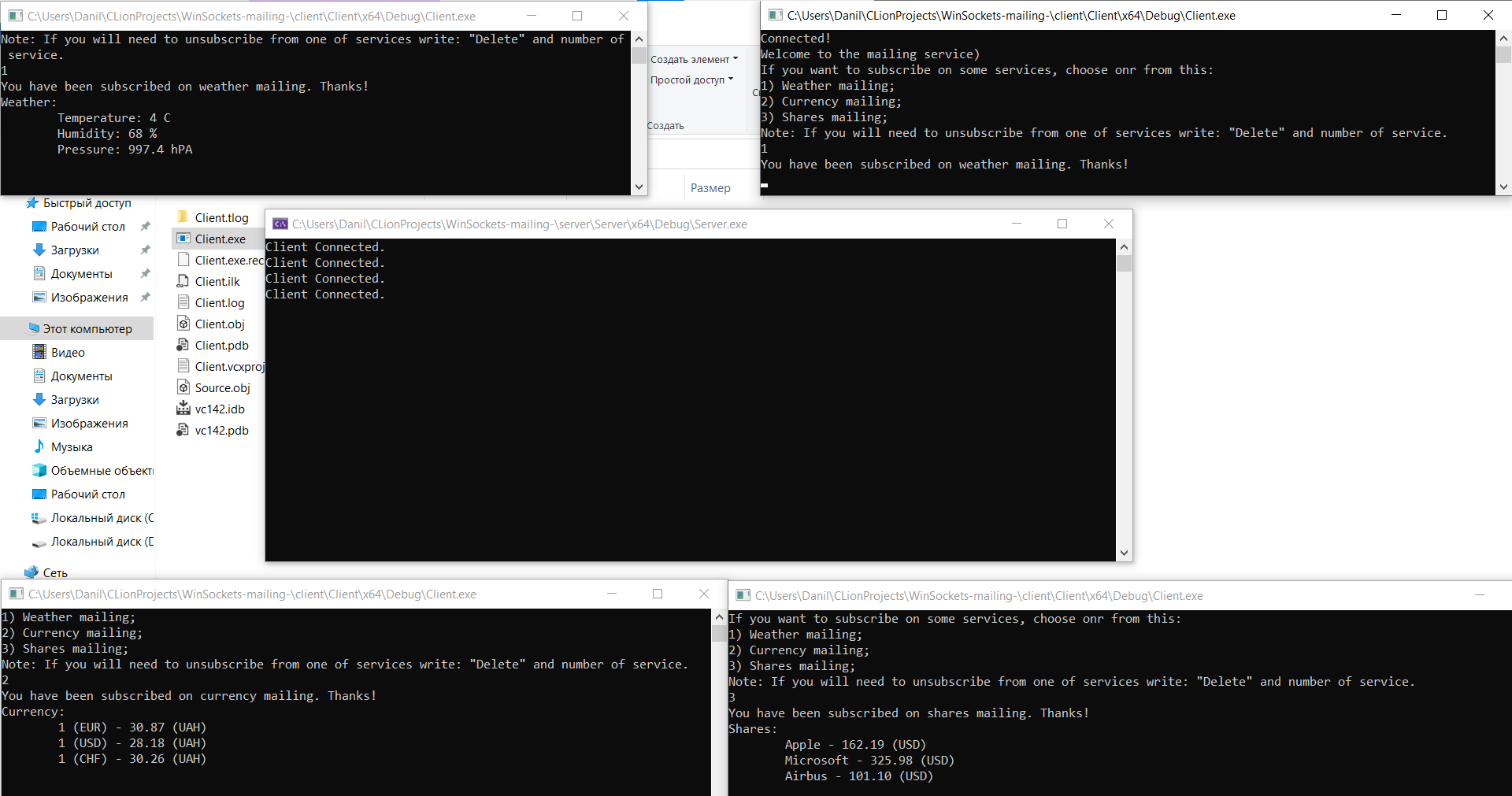


Рис 2. Різні підписки

**Висновок**

На даній лабораторній роботі я ознайомився зі способами міжпроцесної взаємодії. Ознайомився з класичним прикладом взаємодії між процесами на прикладі задачі «виробник – споживач». Навчився працювати із процесами з використанням способів міжпроцесної взаємодії, синхронізувати їхню роботу.

Реазлізував програму міжпроцессорної взаємодії моделі Клієнт-Сервер.