Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Інститут комп’ютерних наук та інформаційних технологій

Кафедра програмного забезпечення



**Звіт**

Про виконання лабораторної роботи №9

**на тему:**

«Організація взаємодії між процесами.»

З дисципліни «Операційні системи»

**Лектор:**

ст. викл. каф. ПЗ.

Грицай О.Д.

**Виконав:**

ст. гр. ПЗ-26

Балій М.І.

**Прийняв:**

доц. каф. ПЗ

Горечко О.М.

« \_\_ » \_\_\_\_\_\_\_\_ 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.

**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.

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.

**Постановка 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 -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 програми, яка запускає серверну частину:

#include "dialog.h"

#include "./ui\_dialog.h"

STARTUPINFOA si[20];

PROCESS\_INFORMATION pi[20];

Dialog::**Dialog**(QWidget \*parent)

: QDialog(*parent*)

, ui(*new* Ui::Dialog)

{

ui->setupUi(*this*);

ui->l\_message->hide();

}

Dialog::~***Dialog***()

{

*delete* ui;

}

void Dialog::**setError**(QString error) {

ui->l\_message->show();

ui->l\_message->setText(error);

QTimer::singleShot(1500, ui->l\_message, &QLabel::hide);

}

void Dialog::**on\_b\_start\_clicked**()

{

int prAmmount = ui->sB\_prAmmount->value();

*if*(prAmmount == 0) {

setError("ammount can not be zero");

*return*;

}

int typeMethod = 1;

*for*(int i = 0; i < prAmmount; i++) {

ZeroMemory(&si[i], *sizeof*(si[i]));

ZeroMemory(&pi[i], *sizeof*(pi[i]));

si[i].cb = *sizeof*(si[i]);

char path[100] = "OS\_9\_5.exe";

bool bCreateProcess = NULL;

char tmp[10];

sprintf\_s(*tmp*, " %d",typeMethod);

strcat\_s(*path*, tmp);

bCreateProcess = CreateProcessA(NULL, LPSTR(path), NULL, NULL, FALSE, 0, NULL, NULL, &si[i], &pi[i]);

*if*(!bCreateProcess) {

setError("cannot create a process");

*return*;

}

}

}

void Dialog::**on\_pushButton\_clicked**()

{

STARTUPINFOA siPipes;

PROCESS\_INFORMATION piPipes;

char path[500] = "D:\\Workspace\\cpp\\InterProcessCommunication\\Project56\\x64\\Debug\\Project56.exe";

ZeroMemory(&siPipes, *sizeof*(siPipes));

ZeroMemory(&piPipes, *sizeof*(piPipes));

siPipes.cb = *sizeof*(siPipes);

bool bCreateProcess = CreateProcessA(NULL, LPSTR(path), NULL, NULL, FALSE, 0, NULL, NULL, &siPipes, &piPipes);

*if*(!bCreateProcess) {

setError("cannot create a process");

*return*;

}

}

void Dialog::**on\_pushButton\_2\_clicked**()

{

STARTUPINFOA siPipes;

PROCESS\_INFORMATION piPipes;

char path[500] = "D:\\Workspace\\cpp\\InterProcessCommunication\\Project57\\x64\\Debug\\Project57.exe";

ZeroMemory(&siPipes, *sizeof*(siPipes));

ZeroMemory(&piPipes, *sizeof*(piPipes));

siPipes.cb = *sizeof*(siPipes);

bool bCreateProcess = CreateProcessA(NULL, LPSTR(path), NULL, NULL, FALSE, 0, NULL, NULL, &siPipes, &piPipes);

*if*(!bCreateProcess) {

setError("cannot create a process");

*return*;

}

}

Код серверної частини:

Server:

#include <iostream>

#include <string>

#include <vector>

#include <Windows.h>

std::vector<HANDLE> myPipeVector;

HANDLE hPipe;

std::vector<HANDLE> myThreadVector;

HANDLE hTempThread;

const int BUFFER\_SIZE = 1024;

DWORD WINAPI ThreadFunc(LPVOID param)

{

std::cout << "Thread func\n";

int nPipeIndex = (int)param;

while (true) {

char buffer[BUFFER\_SIZE];

DWORD bytesRead, bytesWrite;

if (ReadFile(myPipeVector[nPipeIndex], buffer, BUFFER\_SIZE, &bytesRead,

NULL))

{

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

}

else

{

std::cout << "Read file error" << std::endl;

}

for (int i = 0; i < myPipeVector.size() - 1; i++)

{

std::cout << "Sending message to pipe " << nPipeIndex << std::endl;

if (!WriteFile(myPipeVector[i], buffer, bytesRead, &bytesWrite,

NULL))

{

std::cout << "WriteFile failed. Error: " << GetLastError() <<

std::endl;

return 1;

}

std::cout << "Sending message to pipe " << nPipeIndex << "complete" << std::endl;

}

}

std::cout << "End of thread func!\n";

return 0;

}

int main() {

int nPipesCounter = 0;

while (true) {

hPipe = CreateNamedPipe(L"\\\\.\\pipe\\MyPipe", PIPE\_ACCESS\_DUPLEX,

PIPE\_TYPE\_MESSAGE | PIPE\_READMODE\_MESSAGE | PIPE\_WAIT,

PIPE\_UNLIMITED\_INSTANCES, BUFFER\_SIZE \* 16, BUFFER\_SIZE \* 16, 0, NULL);

if (hPipe == INVALID\_HANDLE\_VALUE)

{

std::cout << "CreateNamedPipe failed. Error: " << GetLastError() <<

std::endl;

break;

}

//Pipe was successfully created

myPipeVector.push\_back(hPipe);

std::cout << "Waiting for client connection..." << std::endl;

if (!ConnectNamedPipe(myPipeVector[nPipesCounter], NULL))

{

std::cout << "ConnectNamedPipe failed. Error: " << GetLastError()

<< std::endl;

break;

}

std::cout << "Client connected." << std::endl;

hTempThread = CreateThread(NULL, 0, ThreadFunc, (LPVOID)nPipesCounter,

0, NULL);

myThreadVector.push\_back(hTempThread);

nPipesCounter++;

}

int nNumOfThreads = myThreadVector.size();

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

{

WaitForSingleObject(myThreadVector[i], INFINITE);

}

//free system resources

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

{

CloseHandle(myThreadVector[i]);

}

int nNumOfPipes = myPipeVector.size();

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

{

DisconnectNamedPipe(myPipeVector[i]);

CloseHandle(myPipeVector[i]);

}

return 0;

}

Client:

#include <iostream>

#include <string>

#include <windows.h>

using namespace std;

HANDLE hPipe;

HANDLE hThread;

const int BUFFER\_SIZE = 1024;

HANDLE hMutex;

DWORD WINAPI ThreadFunc(LPVOID param)

{

while (true) {

char buffer[BUFFER\_SIZE];

DWORD bytesRead;

WaitForSingleObject(hMutex, INFINITE);

if (ReadFile(hPipe, buffer, BUFFER\_SIZE, &bytesRead, NULL)) {

std::cout << "The number of read bytes = " << bytesRead << '\n';

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

return 1;

}

else {

std::cout << "ReadFile failed. Error: " << GetLastError() <<

std::endl;

}

ReleaseMutex(hMutex);

}

std::cout << "End of thread func!\n";

return 0;

}

int main() {

hMutex = CreateMutex(NULL, FALSE, NULL);

hPipe = CreateFile(L"\\\\.\\pipe\\MyPipe", GENERIC\_READ | GENERIC\_WRITE, 0,

NULL, OPEN\_EXISTING, 0, NULL);

if (hPipe == INVALID\_HANDLE\_VALUE) {

std::cout << "CreateFile failed. Error: " << GetLastError() << endl;

return 1;

}

hThread = CreateThread(NULL, 0, ThreadFunc, (LPVOID)NULL, 0, NULL);

while (true) {

WaitForSingleObject(hMutex, INFINITE);

string message;

std::cout << "Enter message: ";

getline(cin, message);

DWORD nWrittenBytes;

if (!WriteFile(hPipe, message.c\_str(), message.length() + 1,

&nWrittenBytes, NULL)) {

std::cout << "WriteFile failed. Error: " << GetLastError() << endl;

}

else {

std::cout << "The number of written bytes = " << nWrittenBytes

<< '\n';

}

ReleaseMutex(hMutex);

}

CloseHandle(hPipe);

return 0;

}

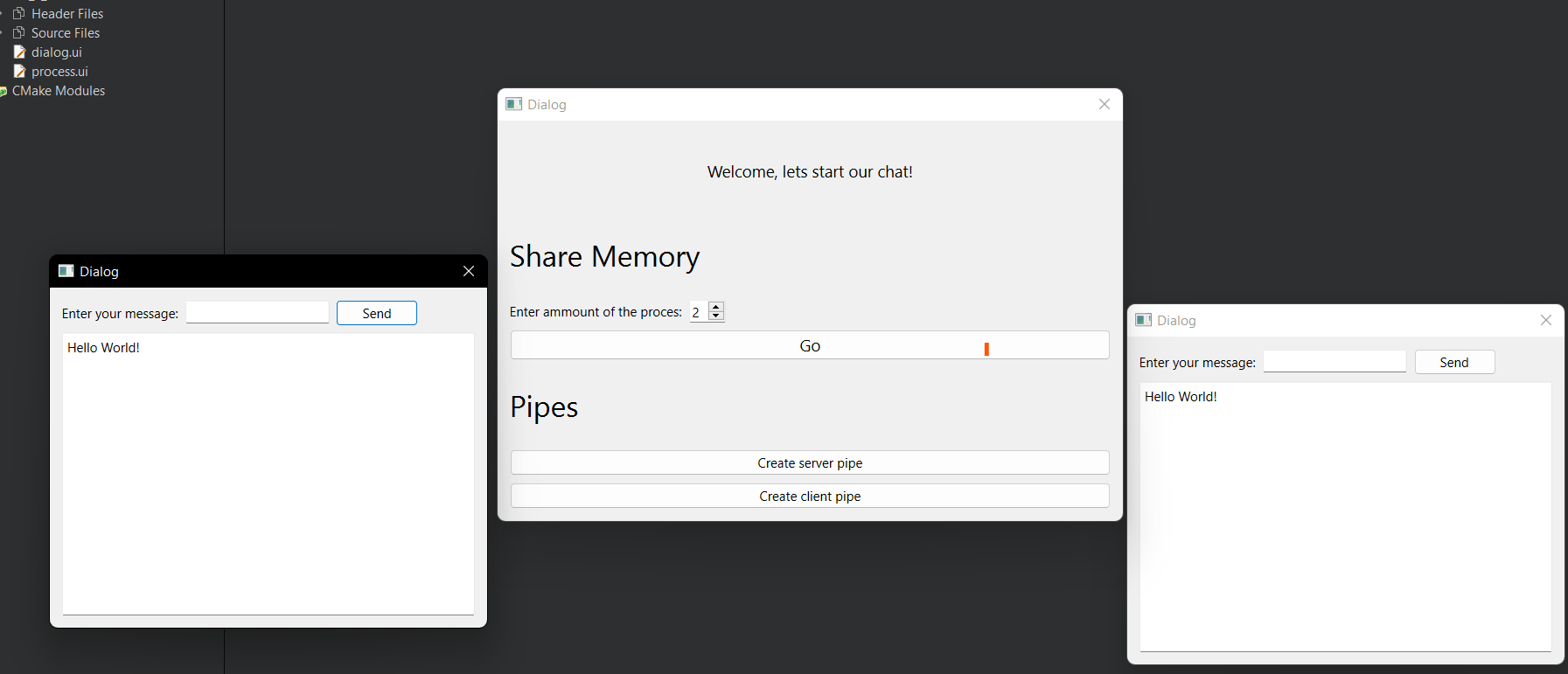


Рис.1. Графічний інтерфейс програми та міжпроцесна взаємодія на основі SharedMemory.

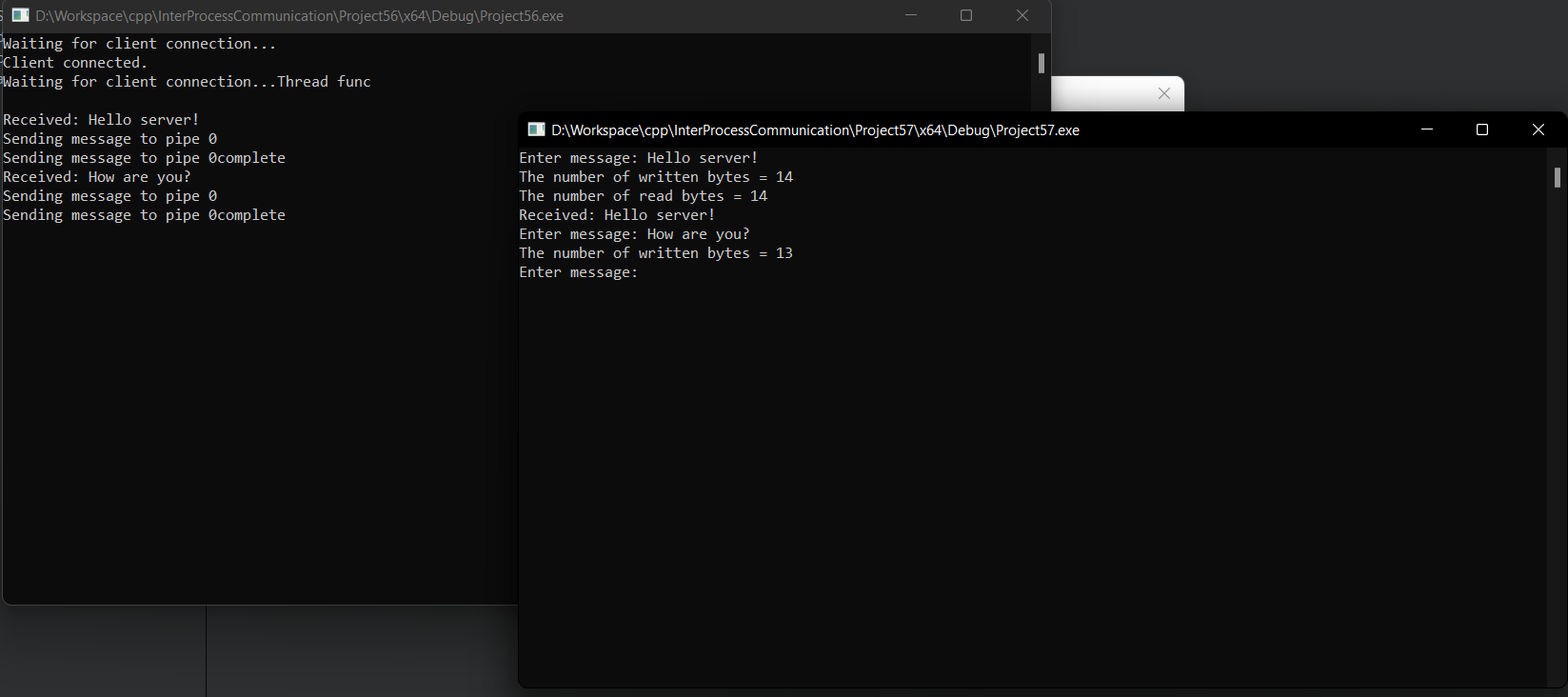
****

Рис.2. Міжпроцесна взаємодія на основі Pipes.

.

**Висновки**

Під час виконання лабораторної роботи я ознайомився з методами міжпроцесної взаємодії. Вдосконалив навики роботи із процесами та графічними інтерфейсами. Запрограмував взаємодію між процесами за допомогою пайпів, та синхронізував їхню роботу. Код проекту завантажив на github - <https://github.com/Kolia913/OS_LAB9_VAR5>.