**Тема роботи:** Створення та керування процесами засобами API в операційній системі WINDOWS.

**Мета роботи:** Ознайомитися з розпаралеленням процесу, дослідити прискорення виконання завдання. Навчитися створювати процеси, керувати їх виконанням та моніторити основні параметри, використовуючи WinAPI-функції.

**Теоретичні відомості**

**Ядро операційної системи** є основною частиною, що забезпечує інтерфейс між прикладними програмами та апаратним забезпеченням комп’ютера. Функції ядра відокремленні від функцій решти виконавчої системи. Ядро відповідає за системні механізми і не бере участі в управлінні завданнями, що пов'язані зі системною політикою. До завдань виконавчої системи належать практично всі, крім планування і диспетчеризації потоків. За межами ядра виконавча система представляє потоки та інші поділювані ресурси у вигляді об'єктів. Управління цими об'єктами вимагає певних витрат, так як потрібні дескриптори, що дозволяють маніпулювати об'єктами, засоби захисту та квоти ресурсів, що резервуються при їх створенні. Ядро реалізує набір більш простих об'єктів, що називаються об'єктами ядра (kernel objects).

Система дозволяє створювати і оперувати декількома типами таких об'єктів, в тому числі: **маркерами доступу** (access token objects), **файлами** (file objects), **відображеннями** (проекціями) **файлів** (file-mapping objects), портами завершення введення виведення (I / O completion port objects), завданнями (jobs), поштовими скриньками (mailslot objects), м'ютексами (mutex objects), каналами (pipe objects), процесами (thread objects) і таймерами очікування (waitable timer objects).

**Створення Win32 процесу здійснюється викликом однієї з функцій**: CreateProcess, CreateProcessAsUser (для Win NT/2000) і CreateProcessWithLogonW (починаючи з Win2000) і відбувається у декілька етапів:

*▪ Відкривається файл образу (EXE), який виконуватиметься в процесі. Якщо виконуваний файл не є Win32 додатком, то шукається образ підтримки (support image) для запуску цієї програми. Наприклад, якщо виконується файл з розширенням .bat, запускається cmd.exe.*

*▪ Створюється об'єкт Win32 «процес».*

*▪ Створюється первинний потік (стек, контекст і об'єкт «потік»).*

*▪ Підсистема Win32 повідомляється про створення нового процесу і потоку.*

*▪ Починається виконання первинного потоку.*

*▪ У контексті нового процесу і потоку ініціалізується адресний простір (наприклад, завантажуються необхідні DLL) і починається виконання програми.*

Для створення нового процесу та його головного потоку використовується функція CreateProcess().

*BOOL CreateProcessA(*

*LPCSTR lpApplicationName,*

*LPSTR lpCommandLine,*

*LPSECURITY\_ATTRIBUTES lpProcessAttributes, LPSECURITY\_ATTRIBUTES lpThreadAttributes,*

*BOOL bInheritHandles,*

*DWORD dwCreationFlags,*

*LPVOID lpEnvironment,*

*LPCSTR lpCurrentDirectory,*

*LPSTARTUPINFOA lpStartupInfo,*

*LPPROCESS\_INFORMATION lpProcessInformation);*

Якщо функція виконалась успішно, то повертається ненульове значення. Якщо сталась помилка, то повернеться нуль.

Структура STARTUPINFO використовується для визначення віконного терміналу, стандартного дескриптора і зовнішнього вигляду для вікна нового процесу

typedef struct \_STARTUPINFOA {

DWORD cb; LPSTR lpReserved;

LPSTR lpDesktop; LPSTR lpTitle;

DWORD dwX; DWORD dwY;

DWORD dwXSize; DWORD dwYSize;

DWORD dwXCountChars;

DWORD dwYCountChars;

DWORD dwFillAttribute;

DWORD dwFlags;

WORD wShowWindow;

WORD cbReserved2;

LPBYTE lpReserved2;

HANDLE hStdInput;

HANDLE hStdOutput;

HANDLE hStdError; }

STARTUPINFOA, \*LPSTARTUPINFOA

**Індивідуальне завдання:**

**І -** Дослідити прискорення виконання процесу табулювання функції на



заданому проміжку ( )

розпаралеливши його виконання.

**II -** Створити десктопний застосунок для вивчення поняття процес в операційній системі Windows.

**Індивідуальне завдання:** Процес Visual Studio; tracert -w 300 google.com; процес визначення кількості чисел Фібоначчі у масиві >10000 елементів.

**Виконання роботи:**

**Окремий процес табулювання:**

#include <math.h>

#include <iostream>

#include <cstdlib>

#include <string>

#include <iomanip>

#include <windows.h>

#include <tchar.h>

using namespace std;

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

{

if (argc != 4)

{

std::cerr << "Usage: child.exe <a> <b> <c>" << std::endl;

return 1;

}

double a = std::stod(argv[1]);

double b = std::stod(argv[2]);

int n = std::atoi(argv[3]);

double step = (b - a) / n;

double result;

for (double x = a; x <= b; x += step)

{

result = x + x\*(sin(x) + cos(x));

std::cout << std::fixed << std::setprecision(4);

std::cout << x << "\t|\t" << result << std::endl;

}

return 0;

}

**Основний процес:**

#include <windows.h>

#include <iostream>

#include <string>

#include <math.h>

#include <sstream>

#include <tchar.h>

#include <vector>

using namespace std;

int main()

{

int processCount = 0;

string check;

double a = 1;

double b = 7;

int totalN = 100000;

cout << "Do you want to set amount of proceses ? (yes or no) ";

cin >> check;

vector<HANDLE> process\_handles;

if (check == "yes") {

cout << "Enter amount of proceses: ";

cin >> processCount;

cout << "Enter first and second border: ";

cin >> a >> b;

cout << "Enter amount of steps: ";

cin >> totalN;

struct timespec start2;

timespec\_get(&start2, TIME\_UTC);

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

double start = a + i \* ((b - a) / processCount);

double end = start + ((b - a) / processCount);

int numN = totalN / processCount;

STARTUPINFO si = {};

PROCESS\_INFORMATION pi = {};

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

wchar\_t cmdLine[256];

\_snwprintf\_s(cmdLine, sizeof(cmdLine), L"E:\\Навчання\\Мої файли\\2-курс\\ОС-лаб\\lab-3\\ConsoleApplication2\\x64\\Debug\\ConsoleApplication2.exe %f %f %d", start, end, numN);

if (CreateProcess(NULL, cmdLine, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si, &pi)) {

printf("pid=%d, tid=%d\n", pi.dwProcessId, pi.dwThreadId);

process\_handles.push\_back(pi.hProcess);

CloseHandle(pi.hThread);

}

else {

std::cerr << "Error creating process: " << GetLastError() << std::endl;

return 1;

}

}

for (HANDLE handle : process\_handles) {

WaitForSingleObject(handle, INFINITE);

CloseHandle(handle);

}

struct timespec end2;

timespec\_get(&end2, TIME\_UTC);

double walltime = (end2.tv\_sec - start2.tv\_sec) + (end2.tv\_nsec - start2.tv\_nsec) / 1000000000.0;

cout << "Time of execution :" << walltime << "s \n" << endl;

}

else {

processCount = 1;

double\* time\_count;

time\_count = new double[6];

for (int j = 1; j < 12; ++j) {

struct timespec start2;

timespec\_get(&start2, TIME\_UTC);

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

double start = a + i \* ((b - a) / processCount);

double end = start + ((b - a) / processCount);

int numN = totalN / processCount;

STARTUPINFO si = {};

PROCESS\_INFORMATION pi = {};

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

wchar\_t cmdLine[256];

\_snwprintf\_s(cmdLine, sizeof(cmdLine), L"E:\\Навчання\\Мої файли\\2-курс\\ОС-лаб\\lab-3\\ConsoleApplication2\\x64\\Debug\\ConsoleApplication2.exe %f %f %d", start, end, numN);

if (CreateProcess(NULL, cmdLine, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si, &pi)) {

printf("pid=%d, tid=%d\n", pi.dwProcessId, pi.dwThreadId);

process\_handles.push\_back(pi.hProcess);

CloseHandle(pi.hThread);

}

else {

std::cerr << "Error creating process: " << GetLastError() << std::endl;

return 1;

}

}

for (HANDLE handle : process\_handles) {

WaitForSingleObject(handle, INFINITE);

CloseHandle(handle);

}

struct timespec end2;

timespec\_get(&end2, TIME\_UTC);

double walltime = (end2.tv\_sec - start2.tv\_sec) + (end2.tv\_nsec - start2.tv\_nsec) / 1000000000.0;

cout << "Time of execution :" << walltime << "s \n" << endl;

time\_count[j - 1] = walltime;

processCount++;

}

int k = 0;

double min\_count = 100;

for (int j = 0; j < 6; ++j) {

if (min\_count > time\_count[j]) {

min\_count = time\_count[j];

k = j + 1;

}

}

cout << "The most efficient way to perform this process in: " << k << "threads" << endl;

}

return 0;

}

1. **Код програми**

#pragma once

#include <windows.h>

#include <iostream>

#include <tchar.h>

#include <chrono>

#include <Psapi.h>

#include <list>

#include <tlhelp32.h>

#include <msclr/marshal\_cppstd.h>

#include <Windows.h>

#include <sstream>

#include <string>

#include <conio.h>

#include <vector>

#include <algorithm>

namespace LabOC3PartTwo {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Collections::Generic;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

using namespace System::Diagnostics;

using namespace System::IO;

using namespace System::Runtime::InteropServices;

using namespace msclr::interop;

int count = 0;

PROCESS\_INFORMATION notionProcessInfo;

PROCESS\_INFORMATION resmonProcessInfo;

PROCESS\_INFORMATION insertionSortProcessInfo;

/\*std::vector<PROCESS\_INFORMATION> processList;

std::vector<DWORD> IDList;\*/

struct ProcessInfoPair {

DWORD key; // Ключ (наприклад, ідентифікатор процесу)

PROCESS\_INFORMATION value; // Значення (інформація про процес)

int index;

};

std::vector<ProcessInfoPair> processList;

/// <summary>

/// Summary for MyForm

/// </summary>

public ref class MyForm : public System::Windows::Forms::Form

{

public:

MyForm(void)

{

InitializeComponent();

//

//TODO: Add the constructor code here

//

}

protected:

/// <summary>

/// Clean up any resources being used.

/// </summary>

~MyForm()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::CheckBox^ checkBox\_VS;

private: System::Windows::Forms::CheckBox^ checkBox\_tracert;

private: System::Windows::Forms::CheckBox^ checkBox\_Fibonachi;

private: System::Windows::Forms::Button^ button\_inputPrioruty;

private: System::Windows::Forms::Button^ button\_resume;

private: System::Windows::Forms::Button^ button\_suspend;

private: System::Windows::Forms::ComboBox^ comboBox\_priority;

private: System::Windows::Forms::CheckedListBox^ checkedListBox1;

private: System::Windows::Forms::Button^ button\_Set\_affinity;

private: System::Windows::Forms::DataGridView^ dataGridView1;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Name;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ ID;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Status;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Priority;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Time;

private: System::ComponentModel::IContainer^ components;

protected:

protected:

private:

/// <summary>

/// Required designer variable.

/// </summary>

int indexVS, indexResmon, indexSort;

private: System::Windows::Forms::TextBox^ textBox\_id;

private: System::Windows::Forms::Button^ button1;

private: System::Windows::Forms::Timer^ timer1;

private: System::Windows::Forms::Button^ button\_create;

#pragma region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

void InitializeComponent(void)

{

this->components = (gcnew System::ComponentModel::Container());

this->checkBox\_VS = (gcnew System::Windows::Forms::CheckBox());

this->checkBox\_tracert = (gcnew System::Windows::Forms::CheckBox());

this->checkBox\_Fibonachi = (gcnew System::Windows::Forms::CheckBox());

this->button\_inputPrioruty = (gcnew System::Windows::Forms::Button());

this->button\_resume = (gcnew System::Windows::Forms::Button());

this->button\_suspend = (gcnew System::Windows::Forms::Button());

this->comboBox\_priority = (gcnew System::Windows::Forms::ComboBox());

this->checkedListBox1 = (gcnew System::Windows::Forms::CheckedListBox());

this->button\_Set\_affinity = (gcnew System::Windows::Forms::Button());

this->dataGridView1 = (gcnew System::Windows::Forms::DataGridView());

this->Name = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->ID = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->Status = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->Priority = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->Time = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->button\_create = (gcnew System::Windows::Forms::Button());

this->textBox\_id = (gcnew System::Windows::Forms::TextBox());

this->button1 = (gcnew System::Windows::Forms::Button());

this->timer1 = (gcnew System::Windows::Forms::Timer(this->components));

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView1))->BeginInit();

this->SuspendLayout();

//

// checkBox\_VS

//

this->checkBox\_VS->AutoSize = true;

this->checkBox\_VS->Location = System::Drawing::Point(78, 467);

this->checkBox\_VS->Margin = System::Windows::Forms::Padding(3, 4, 3, 4);

this->checkBox\_VS->Name = L"checkBox\_VS";

this->checkBox\_VS->Size = System::Drawing::Size(124, 24);

this->checkBox\_VS->TabIndex = 0;

this->checkBox\_VS->Text = L"VisualStudio";

this->checkBox\_VS->UseVisualStyleBackColor = true;

//

// checkBox\_tracert

//

this->checkBox\_tracert->AutoSize = true;

this->checkBox\_tracert->Location = System::Drawing::Point(78, 522);

this->checkBox\_tracert->Margin = System::Windows::Forms::Padding(3, 4, 3, 4);

this->checkBox\_tracert->Name = L"checkBox\_tracert";

this->checkBox\_tracert->Size = System::Drawing::Size(81, 24);

this->checkBox\_tracert->TabIndex = 1;

this->checkBox\_tracert->Text = L"tracert";

this->checkBox\_tracert->UseVisualStyleBackColor = true;

//

// checkBox\_Fibonachi

//

this->checkBox\_Fibonachi->AutoSize = true;

this->checkBox\_Fibonachi->Location = System::Drawing::Point(78, 573);

this->checkBox\_Fibonachi->Margin = System::Windows::Forms::Padding(3, 4, 3, 4);

this->checkBox\_Fibonachi->Name = L"checkBox\_Fibonachi";

this->checkBox\_Fibonachi->Size = System::Drawing::Size(104, 24);

this->checkBox\_Fibonachi->TabIndex = 2;

this->checkBox\_Fibonachi->Text = L"Fibonachi";

this->checkBox\_Fibonachi->UseVisualStyleBackColor = true;

//

// button\_inputPrioruty

//

this->button\_inputPrioruty->Location = System::Drawing::Point(643, 499);

this->button\_inputPrioruty->Margin = System::Windows::Forms::Padding(3, 4, 3, 4);

this->button\_inputPrioruty->Name = L"button\_inputPrioruty";

this->button\_inputPrioruty->Size = System::Drawing::Size(176, 35);

this->button\_inputPrioruty->TabIndex = 4;

this->button\_inputPrioruty->Text = L"Change priority";

this->button\_inputPrioruty->UseVisualStyleBackColor = true;

this->button\_inputPrioruty->Click += gcnew System::EventHandler(this, &MyForm::button\_inputPrioruty\_Click);

//

// button\_resume

//

this->button\_resume->Location = System::Drawing::Point(642, 657);

this->button\_resume->Margin = System::Windows::Forms::Padding(3, 4, 3, 4);

this->button\_resume->Name = L"button\_resume";

this->button\_resume->Size = System::Drawing::Size(174, 28);

this->button\_resume->TabIndex = 5;

this->button\_resume->Text = L"Resume";

this->button\_resume->UseVisualStyleBackColor = true;

this->button\_resume->Click += gcnew System::EventHandler(this, &MyForm::button\_resume\_Click);

//

// button\_suspend

//

this->button\_suspend->Location = System::Drawing::Point(643, 609);

this->button\_suspend->Margin = System::Windows::Forms::Padding(3, 4, 3, 4);

this->button\_suspend->Name = L"button\_suspend";

this->button\_suspend->Size = System::Drawing::Size(173, 28);

this->button\_suspend->TabIndex = 6;

this->button\_suspend->Text = L"Suspend";

this->button\_suspend->UseVisualStyleBackColor = true;

this->button\_suspend->Click += gcnew System::EventHandler(this, &MyForm::button3\_Click);

//

// comboBox\_priority

//

this->comboBox\_priority->DropDownStyle = System::Windows::Forms::ComboBoxStyle::DropDownList;

this->comboBox\_priority->FormattingEnabled = true;

this->comboBox\_priority->Items->AddRange(gcnew cli::array< System::Object^ >(6) {

L"REALTIME\_PRIORITY", L"HIGH\_PRIORITY",

L"ABOVE\_NORMAL\_PRIORITY", L"NORMAL\_PRIORITY", L"BELOW\_NORMAL\_PRIORITY", L"IDLE\_PRIORITY"

});

this->comboBox\_priority->Location = System::Drawing::Point(643, 459);

this->comboBox\_priority->Margin = System::Windows::Forms::Padding(3, 4, 3, 4);

this->comboBox\_priority->Name = L"comboBox\_priority";

this->comboBox\_priority->Size = System::Drawing::Size(175, 28);

this->comboBox\_priority->TabIndex = 7;

//

// checkedListBox1

//

this->checkedListBox1->ColumnWidth = 60;

this->checkedListBox1->FormattingEnabled = true;

this->checkedListBox1->Items->AddRange(gcnew cli::array< System::Object^ >(16) {

L"ЦП 0", L"ЦП 1", L"ЦП 2", L"ЦП 3", L"ЦП 4",

L"ЦП 5", L"ЦП 6", L"ЦП 7", L"ЦП 8", L"ЦП 9", L"ЦП 10", L"ЦП 11", L"ЦП 12", L"ЦП 13", L"ЦП 14", L"ЦП 15"

});

this->checkedListBox1->Location = System::Drawing::Point(899, 499);

this->checkedListBox1->Margin = System::Windows::Forms::Padding(3, 4, 3, 4);

this->checkedListBox1->MultiColumn = true;

this->checkedListBox1->Name = L"checkedListBox1";

this->checkedListBox1->Size = System::Drawing::Size(268, 165);

this->checkedListBox1->TabIndex = 1;

//

// button\_Set\_affinity

//

this->button\_Set\_affinity->Location = System::Drawing::Point(1019, 460);

this->button\_Set\_affinity->Margin = System::Windows::Forms::Padding(3, 4, 3, 4);

this->button\_Set\_affinity->Name = L"button\_Set\_affinity";

this->button\_Set\_affinity->Size = System::Drawing::Size(148, 31);

this->button\_Set\_affinity->TabIndex = 10;

this->button\_Set\_affinity->Text = L"Set affinity";

this->button\_Set\_affinity->UseVisualStyleBackColor = true;

this->button\_Set\_affinity->Click += gcnew System::EventHandler(this, &MyForm::button\_Set\_affinity\_Click);

//

// dataGridView1

//

this->dataGridView1->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->dataGridView1->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(5) {

this->Name, this->ID,

this->Status, this->Priority, this->Time

});

this->dataGridView1->Location = System::Drawing::Point(78, 15);

this->dataGridView1->Margin = System::Windows::Forms::Padding(3, 4, 3, 4);

this->dataGridView1->Name = L"dataGridView1";

this->dataGridView1->RowHeadersWidth = 51;

this->dataGridView1->RowTemplate->Height = 24;

this->dataGridView1->Size = System::Drawing::Size(1089, 405);

this->dataGridView1->TabIndex = 12;

//

// Name

//

this->Name->HeaderText = L"Name";

this->Name->MinimumWidth = 6;

this->Name->Name = L"Name";

this->Name->Width = 125;

//

// ID

//

this->ID->HeaderText = L"ID";

this->ID->MinimumWidth = 6;

this->ID->Name = L"ID";

this->ID->Width = 125;

//

// Status

//

this->Status->HeaderText = L"Status";

this->Status->MinimumWidth = 6;

this->Status->Name = L"Status";

this->Status->Width = 125;

//

// Priority

//

this->Priority->HeaderText = L"Priority";

this->Priority->MinimumWidth = 6;

this->Priority->Name = L"Priority";

this->Priority->Width = 125;

//

// Time

//

this->Time->HeaderText = L"Time";

this->Time->MinimumWidth = 6;

this->Time->Name = L"Time";

this->Time->Width = 125;

//

// button\_create

//

this->button\_create->Location = System::Drawing::Point(340, 513);

this->button\_create->Margin = System::Windows::Forms::Padding(3, 4, 3, 4);

this->button\_create->Name = L"button\_create";

this->button\_create->Size = System::Drawing::Size(123, 84);

this->button\_create->TabIndex = 13;

this->button\_create->Text = L"Create";

this->button\_create->UseVisualStyleBackColor = true;

this->button\_create->Click += gcnew System::EventHandler(this, &MyForm::button\_create\_Click);

//

// textBox\_id

//

this->textBox\_id->Location = System::Drawing::Point(344, 462);

this->textBox\_id->Margin = System::Windows::Forms::Padding(3, 4, 3, 4);

this->textBox\_id->Name = L"textBox\_id";

this->textBox\_id->Size = System::Drawing::Size(116, 26);

this->textBox\_id->TabIndex = 14;

//

// button1

//

this->button1->Location = System::Drawing::Point(643, 558);

this->button1->Margin = System::Windows::Forms::Padding(3, 4, 3, 4);

this->button1->Name = L"button1";

this->button1->Size = System::Drawing::Size(173, 29);

this->button1->TabIndex = 15;

this->button1->Text = L"button\_kill";

this->button1->UseVisualStyleBackColor = true;

this->button1->Click += gcnew System::EventHandler(this, &MyForm::button1\_Click);

//

// timer1

//

this->timer1->Enabled = true;

this->timer1->Interval = 1000;

this->timer1->Tick += gcnew System::EventHandler(this, &MyForm::OnTimerTick);

//

// MyForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(9, 20);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(1206, 690);

this->Controls->Add(this->button1);

this->Controls->Add(this->textBox\_id);

this->Controls->Add(this->button\_create);

this->Controls->Add(this->dataGridView1);

this->Controls->Add(this->button\_Set\_affinity);

this->Controls->Add(this->checkedListBox1);

this->Controls->Add(this->comboBox\_priority);

this->Controls->Add(this->button\_suspend);

this->Controls->Add(this->button\_resume);

this->Controls->Add(this->button\_inputPrioruty);

this->Controls->Add(this->checkBox\_Fibonachi);

this->Controls->Add(this->checkBox\_tracert);

this->Controls->Add(this->checkBox\_VS);

this->Margin = System::Windows::Forms::Padding(3, 4, 3, 4);/\*

this->Name = L"MyForm";\*/

this->Text = L"MyForm";

this->Load += gcnew System::EventHandler(this, &MyForm::MyForm\_Load);

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView1))->EndInit();

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

String^ GetProcessNameByPID(DWORD processId) {

HANDLE hProcess = OpenProcess(PROCESS\_QUERY\_INFORMATION | PROCESS\_VM\_READ, FALSE, processId);

if (hProcess != NULL) {

WCHAR processName[MAX\_PATH];

if (GetModuleBaseNameW(hProcess, NULL, processName, sizeof(processName) / sizeof(WCHAR))) {

CloseHandle(hProcess);

std::wstring wstr(processName); // Конвертація в std::wstring

return msclr::interop::marshal\_as<String^>(wstr);

}

CloseHandle(hProcess);

}

return L"Unknown";

}

int NameGrid(DWORD name, int index) {

index = dataGridView1->RowCount - 1;

dataGridView1->Rows->Add(GetProcessNameByPID(name));

dataGridView1->Refresh();

return index;

}

void IdGrid(PROCESS\_INFORMATION& processInfo, int index) {

DWORD dwProcessId = processInfo.dwProcessId; // Припустимо, що у вас є dwProcessId типу DWORD

int processIdInt = static\_cast<int>(dwProcessId); // Конвертувати DWORD у int

String^ processIdString = System::Convert::ToString(processIdInt);

dataGridView1->Rows[index]->Cells[1]->Value = processIdString;

dataGridView1->Refresh();

}

void StatusGrid(int index, String^ str) {

dataGridView1->Rows[index]->Cells[2]->Value = str;

dataGridView1->Refresh();

}

void StatusGrid(PROCESS\_INFORMATION& name, int index) {

dataGridView1->Rows[index]->Cells[2]->Value = GetStatus(name);

dataGridView1->Refresh();

}

String^ GetStatus(PROCESS\_INFORMATION& processInfo) {

if (processInfo.hProcess) {

DWORD exitCode;

if (GetExitCodeProcess(processInfo.hProcess, &exitCode)) {

if (exitCode == STILL\_ACTIVE) {

return "R";

}

else {

return "S";

}

}

else {

return "E";

}

}

else {

return "E";

}

}

void PriorityGrid(HANDLE& hProcess, int index) {

dataGridView1->Rows[index]->Cells[3]->Value = GetPriotiry(hProcess);

}

String^ GetPriotiry(HANDLE& hProcess) {

DWORD exitCode = GetPriorityClass(hProcess);

String^ str = exitCode.ToString();

if (exitCode == 128) {

str = "HIGH";

}

else if (exitCode == 32768) {

str = "ABOVE\_NORMAL";

}

else if (exitCode == 32) {

str = "NORMAL";

}

else if (exitCode == 16384) {

str = "BELOW\_NORMAL";

}

else if (exitCode == 64) {

str = "IDLE\_NORMAL";

}

else if (exitCode == 256) {

str = "REALTIME\_NORMAL";

}

return str;

}

private:bool SuspendProcess(DWORD processId) {

HANDLE hThreadSnapshot = CreateToolhelp32Snapshot(TH32CS\_SNAPTHREAD, 0);

THREADENTRY32 threadEntry;

threadEntry.dwSize = sizeof(THREADENTRY32);

Thread32First(hThreadSnapshot, &threadEntry);

do

{

if (threadEntry.th32OwnerProcessID == processId)

{

HANDLE hThread = OpenThread(THREAD\_ALL\_ACCESS, FALSE,

threadEntry.th32ThreadID);

SuspendThread(hThread);

CloseHandle(hThread);

}

} while (Thread32Next(hThreadSnapshot, &threadEntry));

CloseHandle(hThreadSnapshot);

return true;

}

private:bool ResumeProcess(DWORD processId) {

HANDLE hThreadSnapshot = CreateToolhelp32Snapshot(TH32CS\_SNAPTHREAD, 0);

THREADENTRY32 threadEntry;

threadEntry.dwSize = sizeof(THREADENTRY32);

Thread32First(hThreadSnapshot, &threadEntry);

do

{

if (threadEntry.th32OwnerProcessID == processId)

{

HANDLE hThread = OpenThread(THREAD\_ALL\_ACCESS, FALSE,

threadEntry.th32ThreadID);

ResumeThread(hThread);

CloseHandle(hThread);

}

} while (Thread32Next(hThreadSnapshot, &threadEntry));

CloseHandle(hThreadSnapshot);

return true;

}

private: System::Void MyForm\_Load(System::Object^ sender, System::EventArgs^ e) {

}

private:void OnTimerTick(Object^ sender, EventArgs^ e) {

checkIfAllRun();

refreshTable();

}

private: void refreshTable() {

for (int i = 0; i < processList.size(); i++) {

::FILETIME F, M, C, D;

GetProcessTimes(reinterpret\_cast<HANDLE>(processList[i].value.hProcess), &F, &M, &C, &D);

double timeP = FileTimeToTimestamp(C) + FileTimeToTimestamp(D);

dataGridView1->Rows[i]->Cells[4]->Value = timeP;

dataGridView1->Refresh();

}

}

private: void checkIfAllRun() {

for (int i = 0; i < processList.size(); i++) {

if (GetStatus(processList[i].value) != "R") {

RemoveRows(processList[i].index);

}

}

}

private: double FileTimeToTimestamp(const ::FILETIME& fileTime) {

ULARGE\_INTEGER uli;

uli.LowPart = fileTime.dwLowDateTime;

uli.HighPart = fileTime.dwHighDateTime;

return static\_cast<long long>(uli.QuadPart) / 10000000.0;

}

bool CreateProcesses(wchar\_t\* cmdLine, ProcessInfoPair& processInfo) {

STARTUPINFO si = {};

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

if (CreateProcess(NULL, cmdLine, NULL, NULL, FALSE, 0, NULL, NULL, &si, &processInfo.value)) {

processInfo.key = processInfo.value.dwProcessId;

processInfo.index = count;

count++;

return true;

}

else {

DWORD error = GetLastError();

LPVOID errorMsg;

FormatMessage(

FORMAT\_MESSAGE\_ALLOCATE\_BUFFER | FORMAT\_MESSAGE\_FROM\_SYSTEM,

NULL,

error,

0,

(LPWSTR)&errorMsg,

0,

NULL

);

std::wcerr << L"Error creating process: " << errorMsg << std::endl;

LocalFree(errorMsg);

return false;

}

}

private: System::Void button\_create\_Click(System::Object^ sender, System::EventArgs^ e) {

if (checkBox\_VS->Checked) {

wchar\_t name[256];

\_snwprintf\_s(name, sizeof(name), L"E:\\Навчання\\Мої файли\\2-курс\\ОС-лаб\\lab-3\\vscode\\x64\\Debug\\vscode.exe");

ProcessInfoPair processInfo;

if (CreateProcesses(name, processInfo)) {

Sleep(300);

processList.push\_back(processInfo);

indexVS = NameGrid(processInfo.value.dwProcessId, processInfo.index);

IdGrid(processInfo.value, processInfo.index);

StatusGrid(processInfo.value, processInfo.index);

PriorityGrid(processInfo.value.hProcess, processInfo.index);

}

}

if (checkBox\_tracert->Checked) {

std::wstring command;

wchar\_t name[256];

int result = \_snwprintf\_s(name, sizeof(name), L"tracert -w 300 google.com");

command = name;

result = \_wsystem(command.c\_str());

ProcessInfoPair processInfo;

if (CreateProcesses(name, processInfo)) {

Sleep(300);

processList.push\_back(processInfo);

indexVS = NameGrid(processInfo.value.dwProcessId, processInfo.index);

IdGrid(processInfo.value, processInfo.index);

StatusGrid(processInfo.value, processInfo.index);

PriorityGrid(processInfo.value.hProcess, processInfo.index);

}

}

if (checkBox\_Fibonachi->Checked) {

wchar\_t name[256];

\_snwprintf\_s(name, sizeof(name), L"E:\\Навчання\\Мої файли\\2-курс\\ОС-лаб\\lab-3\\find\\x64\\Debug\\find.exe");

ProcessInfoPair processInfo;

if (CreateProcesses(name, processInfo)) {

Sleep(300);

processList.push\_back(processInfo);

indexVS = NameGrid(processInfo.value.dwProcessId, processInfo.index);

IdGrid(processInfo.value, processInfo.index);

StatusGrid(processInfo.value, processInfo.index);

PriorityGrid(processInfo.value.hProcess, processInfo.index);

}

}

}

void TerminateProcessIfExists(PROCESS\_INFORMATION& processInfo) {

if (processInfo.hProcess != NULL && processInfo.hThread != NULL) {

TerminateProcess(processInfo.hProcess, 0);

CloseHandle(processInfo.hProcess);

CloseHandle(processInfo.hThread);

ZeroMemory(&processInfo, sizeof(processInfo));

}

if (WaitForSingleObject(processInfo.hProcess, INFINITE) != WAIT\_OBJECT\_0) {

// Помилка очікування завершення процесу

std::wcerr << L"Error waiting for process termination." << std::endl;

}

}

private: System::Void button\_inputPrioruty\_Click(System::Object^ sender, System::EventArgs^ e) {

String^ selectedPriority = (comboBox\_priority->SelectedItem->ToString());

DWORD dwValue = Convert::ToInt32(textBox\_id->Text);

HANDLE hProcess;

ProcessInfoPair processInfoPair;

for (int i = 0; i < processList.size(); i++) {

if (processList[i].key == dwValue)

{

processInfoPair = processList[i];

break;

}

}

int priorityClass;

if (selectedPriority == L"REALTIME\_PRIORITY") {

priorityClass = REALTIME\_PRIORITY\_CLASS;

}

else if (selectedPriority == L"HIGH\_PRIORITY") {

priorityClass = HIGH\_PRIORITY\_CLASS;

}

else if (selectedPriority == L"ABOVE\_NORMAL\_PRIORITY") {

priorityClass = ABOVE\_NORMAL\_PRIORITY\_CLASS;

}

else if (selectedPriority == L"NORMAL\_PRIORITY") {

priorityClass = NORMAL\_PRIORITY\_CLASS;

}

else if (selectedPriority == L"BELOW\_NORMAL\_PRIORITY") {

priorityClass = BELOW\_NORMAL\_PRIORITY\_CLASS;

}

else if (selectedPriority == L"IDLE\_PRIORITY") {

priorityClass = IDLE\_PRIORITY\_CLASS;

}

BOOL success = SetPriorityClass(processInfoPair.value.hProcess, priorityClass);

PriorityGrid(processInfoPair.value.hProcess, processInfoPair.index);

}

private: System::Void button3\_Click(System::Object^ sender, System::EventArgs^ e) {

DWORD dwValue = Convert::ToInt32(textBox\_id->Text);

HANDLE hThread;

ProcessInfoPair processInfoPair;

for (int i = 0; i < processList.size(); i++) {

if (processList[i].key == dwValue)

{

processInfoPair = processList[i];

break;

}

}

hThread = OpenThread(THREAD\_SUSPEND\_RESUME, FALSE, processInfoPair.value.dwThreadId);

SuspendProcess(processInfoPair.value.dwProcessId);

Sleep(300);

StatusGrid(processInfoPair.index, "S");

}

private: System::Void button\_resume\_Click(System::Object^ sender, System::EventArgs^ e) {

DWORD dwValue = Convert::ToInt32(textBox\_id->Text);

HANDLE hThread;

ProcessInfoPair processInfoPair;

for (int i = 0; i < processList.size(); i++) {

if (processList[i].key == dwValue)

{

processInfoPair = processList[i];

break;

}

}

hThread = OpenThread(THREAD\_SUSPEND\_RESUME, FALSE, processInfoPair.value.dwThreadId);

ResumeProcess(processInfoPair.value.dwProcessId);

Sleep(300);

StatusGrid(processInfoPair.index, "R");

}

private: System::Void button\_Set\_affinity\_Click(System::Object^ sender, System::EventArgs^ e) {

DWORD dwValue = Convert::ToInt32(textBox\_id->Text);

ProcessInfoPair processInfoPair;

for (int i = 0; i < processList.size(); i++) {

if (processList[i].key == dwValue)

{

processInfoPair = processList[i];

break;

}

}

HANDLE hThread = processInfoPair.value.hProcess;

DWORD\_PTR affinityMask = 0;

for (int i = 0; i < checkedListBox1->Items->Count; i++) {

if (checkedListBox1->GetItemChecked(i)) {

affinityMask = affinityMask | (1 << i);

}

}

SetProcessAffinityMask(hThread, affinityMask);

}

void ClearCheckList() {

for (int i = 0; i < checkedListBox1->Items->Count; i++) {

checkedListBox1->SetItemChecked(i, false);

}

}

private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e) {

DWORD dwValue = Convert::ToInt32(textBox\_id->Text);

HANDLE hThread;

ProcessInfoPair processInfoPair;

processInfoPair.index = -1;

for (int i = 0; i < processList.size(); i++) {

if (processList[i].key == dwValue)

{

processInfoPair = processList[i];

break;

}

}

if (processInfoPair.index == -1)

return;

TerminateProcessIfExists(processInfoPair.value);

RemoveRows(processInfoPair.index);

}

void RemoveRows(int rowIndexToRemove) {

count--;

dataGridView1->Rows->RemoveAt(rowIndexToRemove);

dataGridView1->Refresh();

if (rowIndexToRemove >= 0 && rowIndexToRemove < processList.size()) {

processList.erase(processList.begin() + rowIndexToRemove);

}

ProcessInfoPair processInfoPair;

for (int i = rowIndexToRemove; i < processList.size(); i++) {

processList[i].index -= 1;

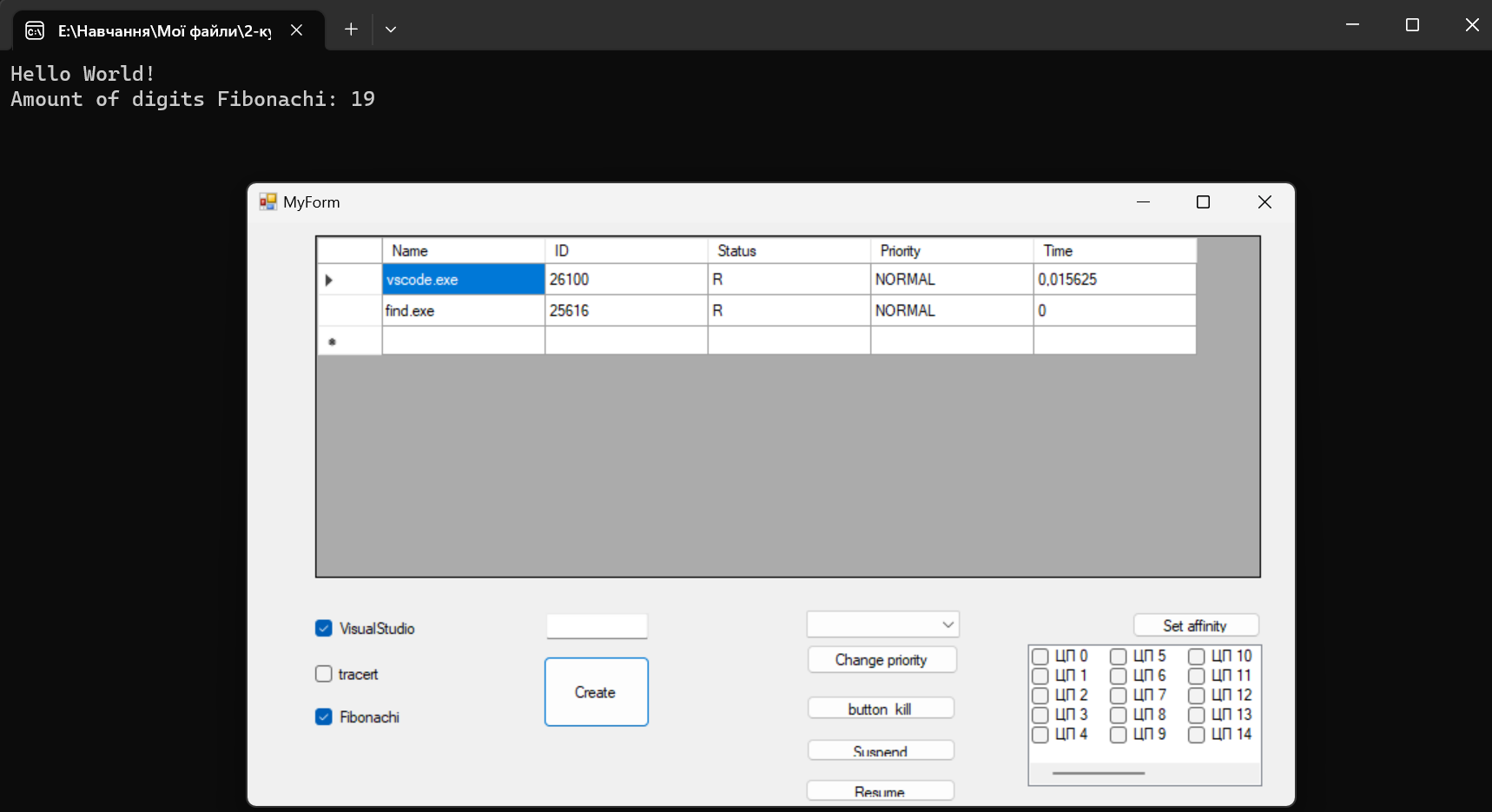
}

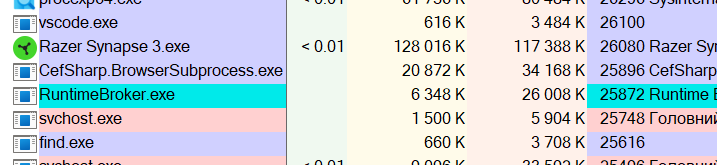
}

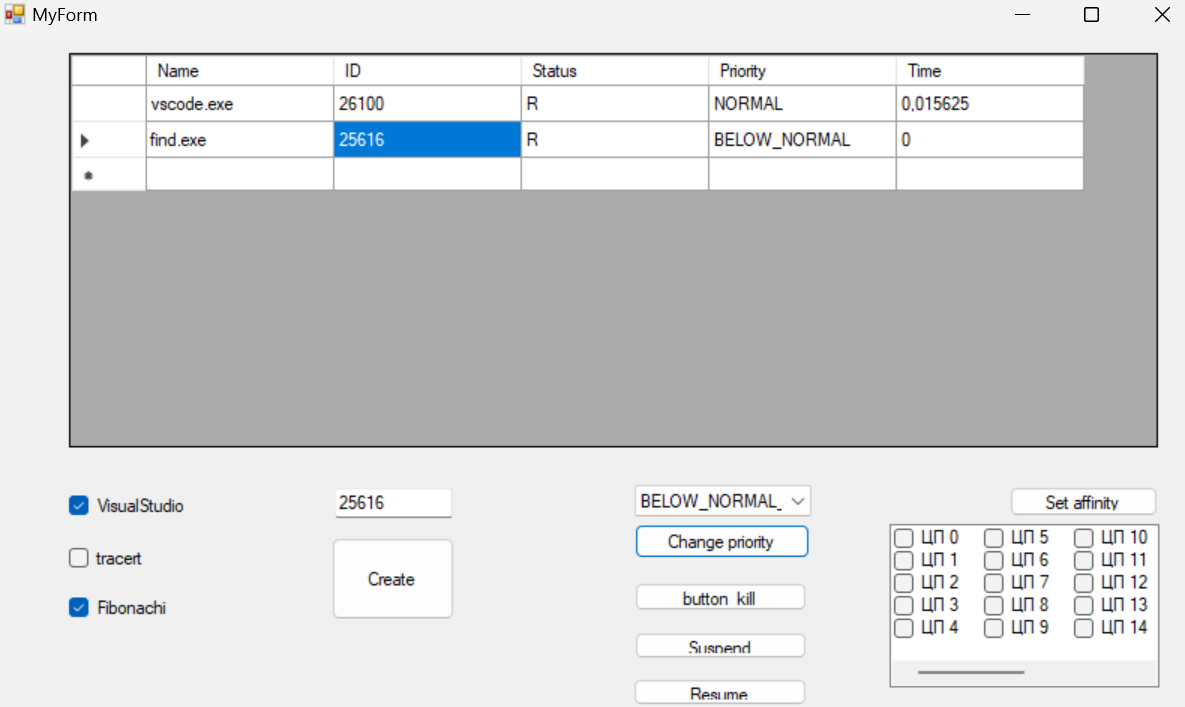
};

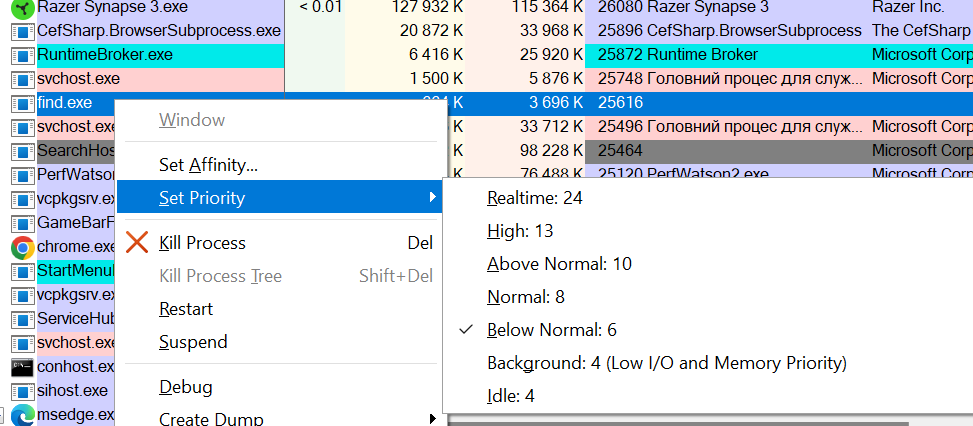
}

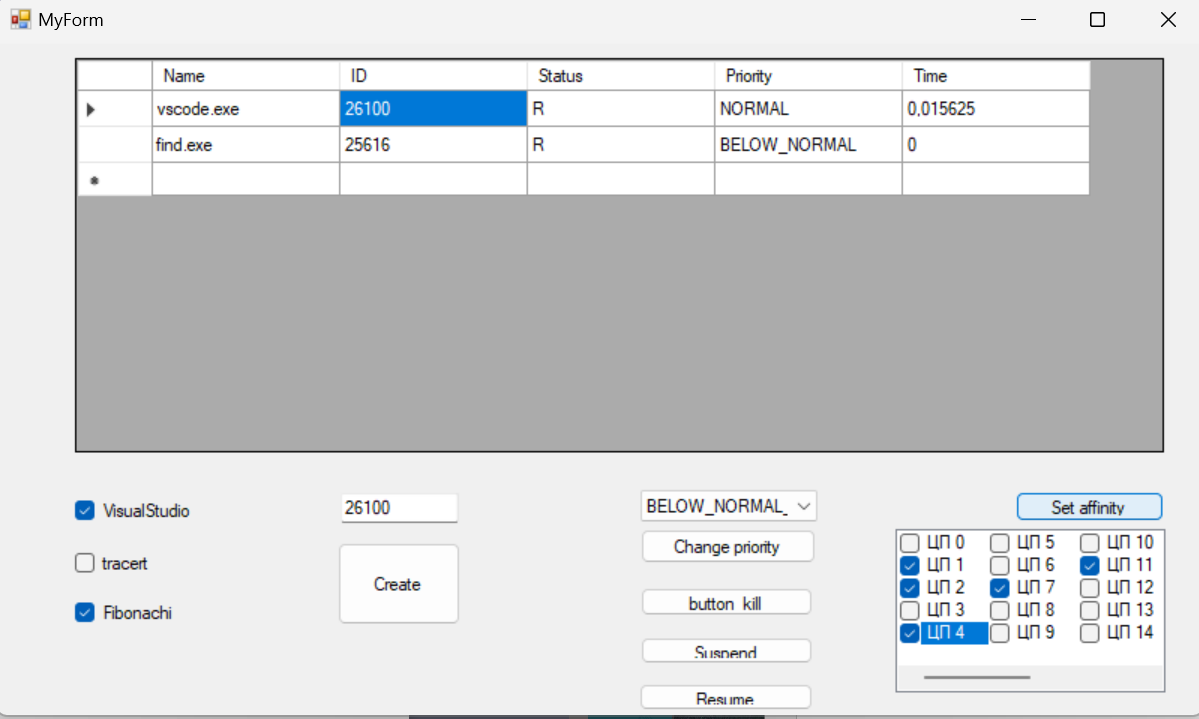
**Результат:**

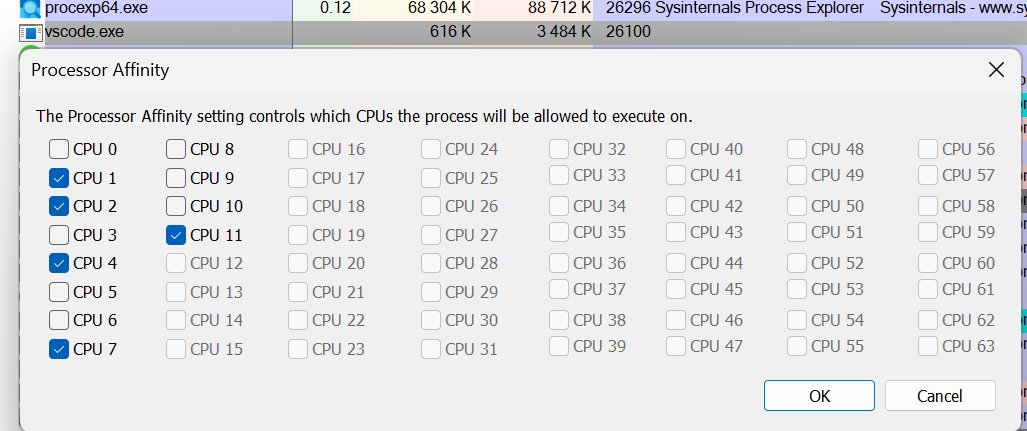
****

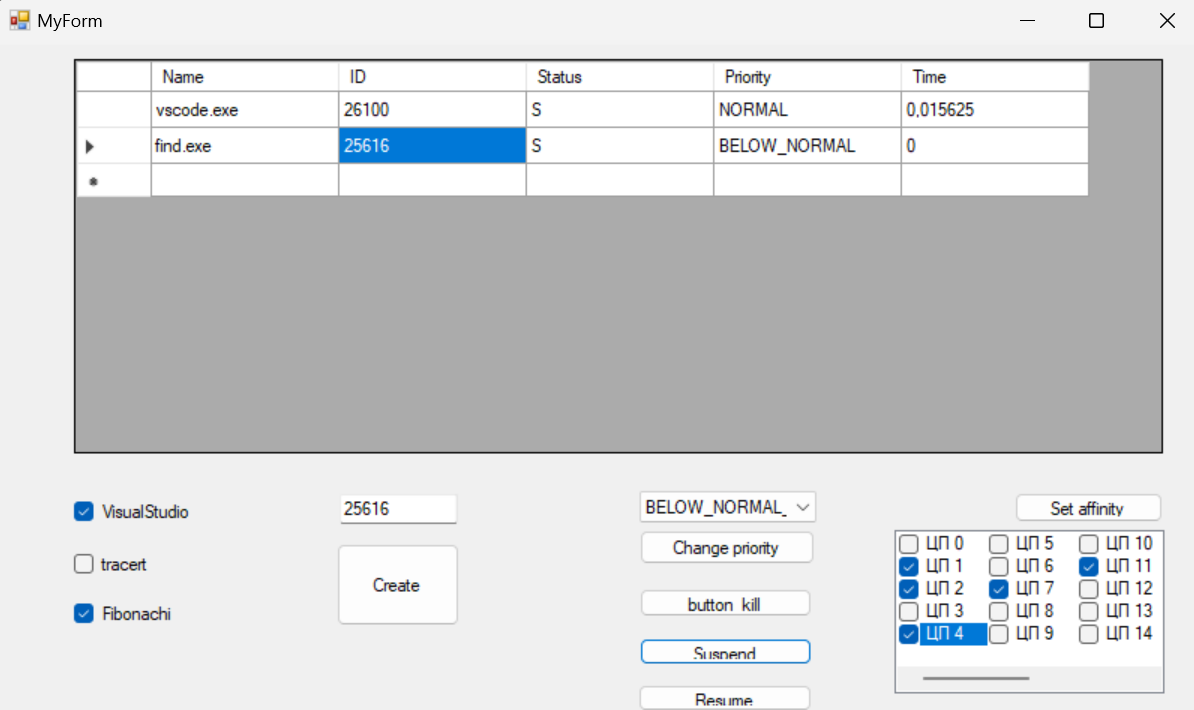
****

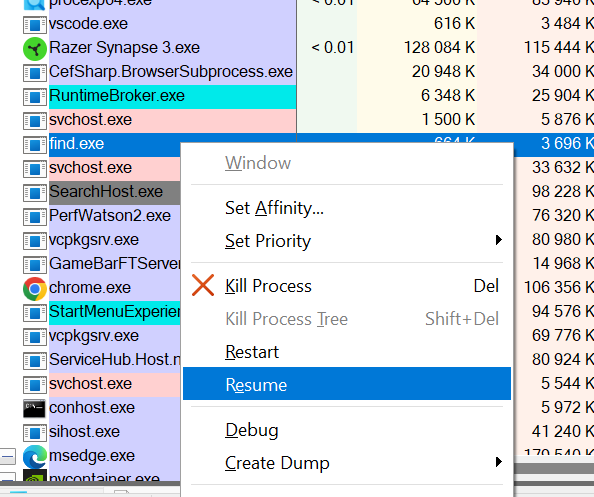
****

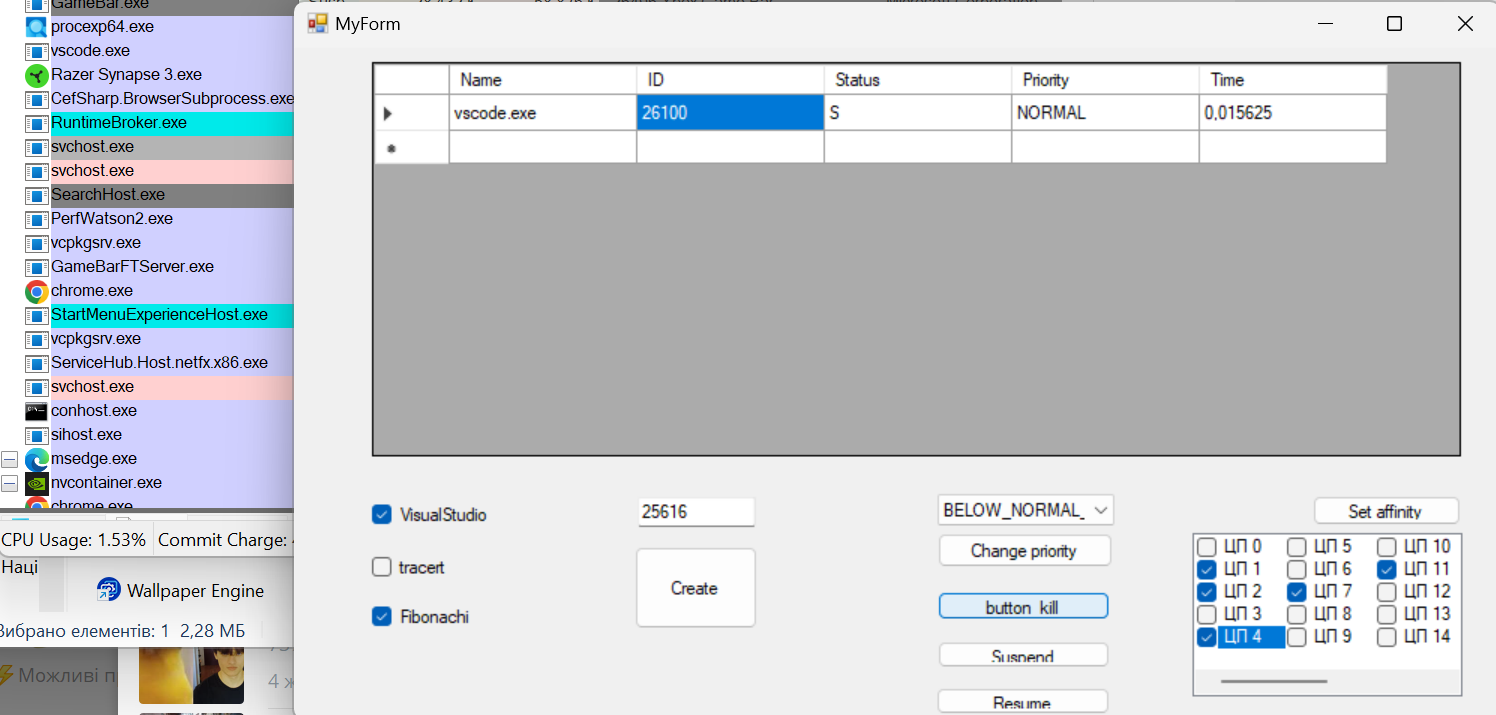
****

****

****

****

****

****

**Висновок:**

Виконавши цю лабораторну роботу, я ознайомився з розпаралеленням процесів, дослідив прискорення виконання процесу, навчився призупиняти, відновлювати, вбивати та створюватии процеси. Я взнав, як у коді задавати пріоритетність процесів, визначати робочі ядра для процеісв, а також навчився використовувати WinAPI-функції.