**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

Інститут **ІКНІ**

Кафедра **ПЗ**

**ЗВІТ**

До лабораторної роботи №3

На тему: «Створення та керування процесами засобами API в операційній системі Windows»

З дисципліни: «Операційні системи»

**Лектор** : ст.викл каф.ПЗ

Грицай О.Д.

**Виконала:** ст.гр.ПЗ-23

Кохман О.В.

**Прийняла:** ст.викл каф.ПЗ

Грицай О.Д.

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2022 р.

\_ \_ \_ \_ \_ \_ \_ \_ .

Львів – 2022

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

**Мета:** Ознайомитись з багатопоточністю в ОС Windows. Навчитись працюватти з процесами , використовуючи WinApi-функції.

**Теоретичні відомості**

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). Ці об'єкти створюються Windows-функціями.

Процес визначають як екземпляр виконуваної програми. Кожному процесу при завантаженні виділяються ресурси операційної системи. Складові елементи процесу: -об'єкт ядра, через який операційна система керує процесом і де зберігається статична інформація про процес - адресний простір де зберігається код програми та дані.

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.

**Індивідуальне завдання**

1. Створити окремий процес, і здійснити в ньому розв’язок задачі згідно варіанту у відповідності до порядкового номера у журнальному списку (підгрупи).
2. Реалізувати розв’язок задачі у 2-ох, 4-ох, 8-ох процесах. Виміряти час роботи процесів за допомогою функцій WinAPI. Порівняти результати роботи в одному і в багатьох процесах.
3. Для кожного процесу реалізувати можливість його запуску, зупинення, завершення та примусове завершення («вбиття»).
4. Реалізувати можливість зміни пріоритету виконання процесу.
5. Продемонструвати результати виконання роботи, а також кількість створених процесів у “Диспетчері задач”, або подібних утилітах (н-д, ProcessExplorer).

Варіант 9:

Вивести для кожного стовпця суму елементів стовпців матриці N\*N (N>1000 задається користувачем, матриця визначається випадково).

**Код програми**

Назва файлу : main.cpp

#include <iostream>

#include <random>

using namespace std;

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

int size = atoi(argv[1]);

random\_device random\_device;

mt19937 generator(random\_device());

uniform\_int\_distribution<> distribution(1, 20);

int\*\* array = new int\* [size];

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

array[i] = new int[size];

}

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

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

array[i][j] = distribution(generator);

}

}

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

int sum = 0;

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

sum += array[i][j];

}

printf("sum = %d\n", sum);

}

return 0;

}

Назва файлу: MyForm.cpp

#include "MyForm.h"

using namespace Main;

int main() {

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

Application::Run(gcnew MyForm());

return 0;

}

Назва файлу: MyForm.h

#pragma once

#include "header.h"

namespace Main {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

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

{

public:

MyForm(void)

{

InitializeComponent();

}

protected:

~MyForm()

{

if (components)

{

delete components;

}

}

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

protected:

private: System::Windows::Forms::Button^ button2;

private: System::Windows::Forms::Button^ button3;

private: System::Windows::Forms::Button^ button4;

private: System::Windows::Forms::Button^ button5;

private: System::Windows::Forms::Button^ button6;

private: System::Windows::Forms::Button^ button7;

private: System::Windows::Forms::TextBox^ textBox1;

private: System::Windows::Forms::TextBox^ textBox2;

private: System::Windows::Forms::TextBox^ textBox3;

private: System::Windows::Forms::TextBox^ textBox4;

private: System::Windows::Forms::TextBox^ textBox5;

private: System::Windows::Forms::TextBox^ textBox6;

private: System::Windows::Forms::TextBox^ textBox7;

private: System::Windows::Forms::Label^ label1;

private: System::Windows::Forms::Label^ label2;

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

private: System::Windows::Forms::Button^ button8;

private: System::Windows::Forms::Button^ button9;

private: System::Windows::Forms::Button^ button10;

private: System::Windows::Forms::Button^ button11;

private: System::Windows::Forms::TextBox^ textBox8;

private: System::Windows::Forms::TextBox^ textBox9;

private:

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

#pragma endregion

PROCESS\_INFORMATION\* processes;

int numberOfProcesses;

private: System::Void createProcesses(System::Object^ sender, System::EventArgs^ e) {

numberOfProcesses = System::Convert::ToInt64(textBox6->Text);

processes = new PROCESS\_INFORMATION[numberOfProcesses];

int numberOfColumns = System::Convert::ToInt64(textBox7->Text);

string path = "C:\\university\\git\\3-sem\\Operating Systems\\lab03\\Sum\\x64\\Debug\\Sum.exe ";

STARTUPINFO si;

PROCESS\_INFORMATION pi;

ZeroMemory(&si, sizeof(STARTUPINFO));

si.cb = sizeof(STARTUPINFO);

dataGridView1->RowCount = numberOfProcesses;

dataGridView1->ColumnCount = 4;

dataGridView1->Columns[0]->HeaderText = "PID";

dataGridView1->Columns[1]->HeaderText = "ThreadID";

dataGridView1->Columns[2]->HeaderText = "Priority";

dataGridView1->Columns[3]->HeaderText = "Status";

for (int i = 1; i <= dataGridView1->RowCount; i++) {

dataGridView1->Rows[i - 1]->HeaderCell->Value = System::Convert::ToString(i);

}

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

string newPath = path + to\_string(numberOfColumns);

if (!CreateProcess(NULL, (LPSTR)newPath.c\_str(), NULL, NULL, TRUE, CREATE\_NEW\_CONSOLE | CREATE\_SUSPENDED, NULL, NULL, & si, & pi)) {

return;

}

dataGridView1->Rows[i]->Cells[0]->Value = pi.dwProcessId;

dataGridView1->Rows[i]->Cells[1]->Value = pi.dwThreadId;

dataGridView1->Rows[i]->Cells[2]->Value = GetPriorityClass(pi.hThread);

dataGridView1->Rows[i]->Cells[3]->Value = "Suspended";

processes[i] = pi;

}

}

private: System::Void stopProcess(System::Object^ sender, System::EventArgs^ e) {

int number = System::Convert::ToInt64(textBox1->Text);

if (SuspendThread(processes[number - 1].hThread) == (DWORD)-1) {

return;

}

dataGridView1->Rows[number-1]->Cells[3]->Value = "Suspended";

}

private: System::Void resumeProcess(System::Object^ sender, System::EventArgs^ e) {

int number = System::Convert::ToInt64(textBox2->Text);

if (ResumeThread(processes[number - 1].hThread) == (DWORD)-1) {

return;

}

dataGridView1->Rows[number - 1]->Cells[3]->Value = "Running";

}

private: System::Void resumeAllProcesses(System::Object^ sender, System::EventArgs^ e) {

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

ResumeThread(processes[i].hThread);

dataGridView1->Rows[i]->Cells[3]->Value = "Running";

}

dataGridView1->Rows[i]->Cells[2]->Value = GetPriorityClass(processes[i].hProcess);

}

private: System::Void stopAllProcesses(System::Object^ sender, System::EventArgs^ e) {

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

SuspendThread(processes[i].hThread);

dataGridView1->Rows[i]->Cells[3]->Value = "Suspended";

}

}

private: System::Void endProcess(System::Object^ sender, System::EventArgs^ e) {

int number = System::Convert::ToInt64(textBox3->Text);

if (!TerminateProcess(processes[number - 1].hProcess, 2)) {

return;

}

CloseHandle(processes[number - 1].hThread);

CloseHandle(processes[number - 1].hProcess);

dataGridView1->Rows[number - 1]->Cells[3]->Value = "Ended";

}

private: System::Void killProcess(System::Object^ sender, System::EventArgs^ e) {

int number = System::Convert::ToInt64(textBox4->Text);

if (!TerminateProcess(processes[number - 1].hProcess, 2)) {

return;

}

CloseHandle(processes[number - 1].hThread);

CloseHandle(processes[number - 1].hProcess);

dataGridView1->Rows[number - 1]->Cells[3]->Value = "Killed";

}

private: System::Void endAllProcesses(System::Object^ sender, System::EventArgs^ e) {

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

TerminateProcess(processes[i].hProcess, 2);

CloseHandle(processes[i].hThread);

CloseHandle(processes[i].hProcess);

dataGridView1->Rows[i]->Cells[3]->Value = "Ended";

}

}

private: System::Void killAllProcesses(System::Object^ sender, System::EventArgs^ e) {

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

TerminateProcess(processes[i].hProcess, 2);

CloseHandle(processes[i].hThread);

CloseHandle(processes[i].hProcess);

dataGridView1->Rows[i]->Cells[3]->Value = "Killed";

}

}

private: System::Void changePriority(System::Object^ sender, System::EventArgs^ e) {

int priorityArray[] = { IDLE\_PRIORITY\_CLASS, BELOW\_NORMAL\_PRIORITY\_CLASS, NORMAL\_PRIORITY\_CLASS,

ABOVE\_NORMAL\_PRIORITY\_CLASS, HIGH\_PRIORITY\_CLASS, REALTIME\_PRIORITY\_CLASS };

int number = System::Convert::ToInt64(textBox5->Text);

int newPriority = System::Convert::ToInt64(textBox8->Text);

if (!SetPriorityClass(processes[number - 1].hProcess, priorityArray[newPriority])) {

return;

}

dataGridView1->Rows[number - 1]->Cells[2]->Value = GetPriorityClass(processes[number - 1].hProcess);

}

private: System::Void measureTime(System::Object^ sender, System::EventArgs^ e) {

int handleCounter = 0;

HANDLE\* threadHandles = new HANDLE[numberOfProcesses];

double start = GetTickCount64();

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

threadHandles[handleCounter++] = processes[i].hThread;

ResumeThread(processes[i].hThread);

}

WaitForMultipleObjects(handleCounter, threadHandles, TRUE, INFINITE);

double finish = GetTickCount64();

double exTime = (finish - start) \* 0.001;

textBox9->Text = System::Convert::ToString(exTime + " sec");

}

};

}

Назва файлу: header.h

#ifndef HEADER\_H

#define HEADER\_H

#pragma once

#include <string>

#include <Windows.h>

using namespace std;

#endif

**Протокол роботи**

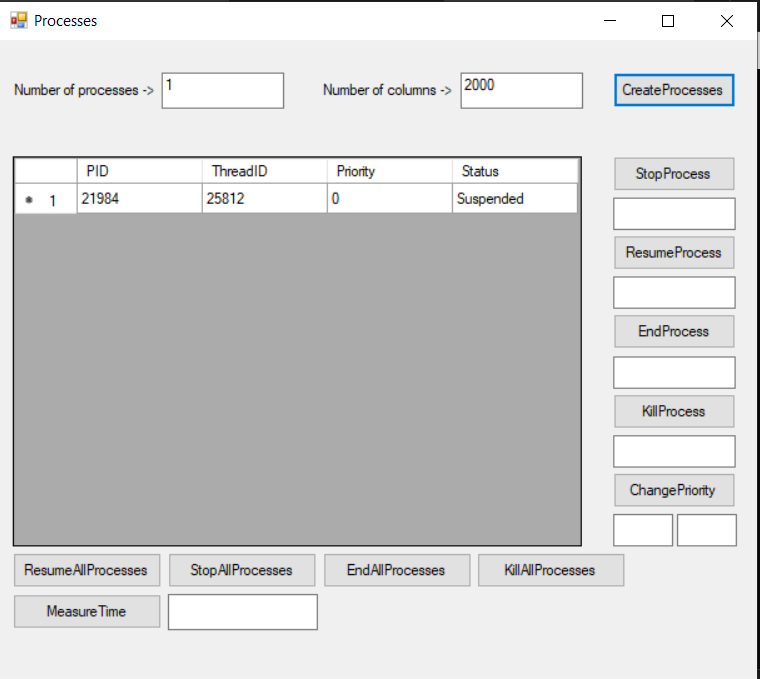
****

Рис. 1 Створення процесу в формі проекту за допомогою кнопки CreateProcesses().

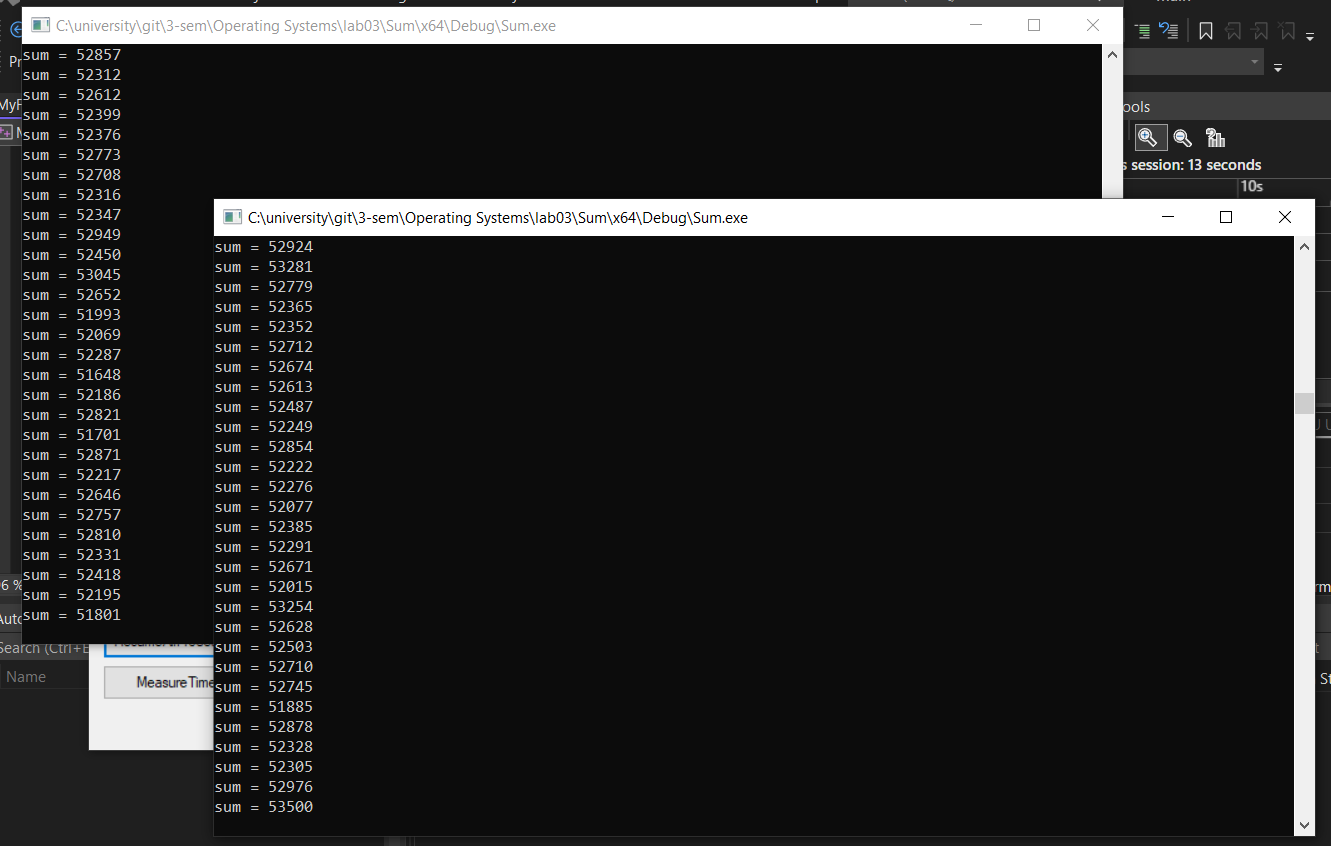


Рис. 2 Запуск призупинених по замовчуванню процесів в стан виконання за допомогою ResumeAllProcesses().

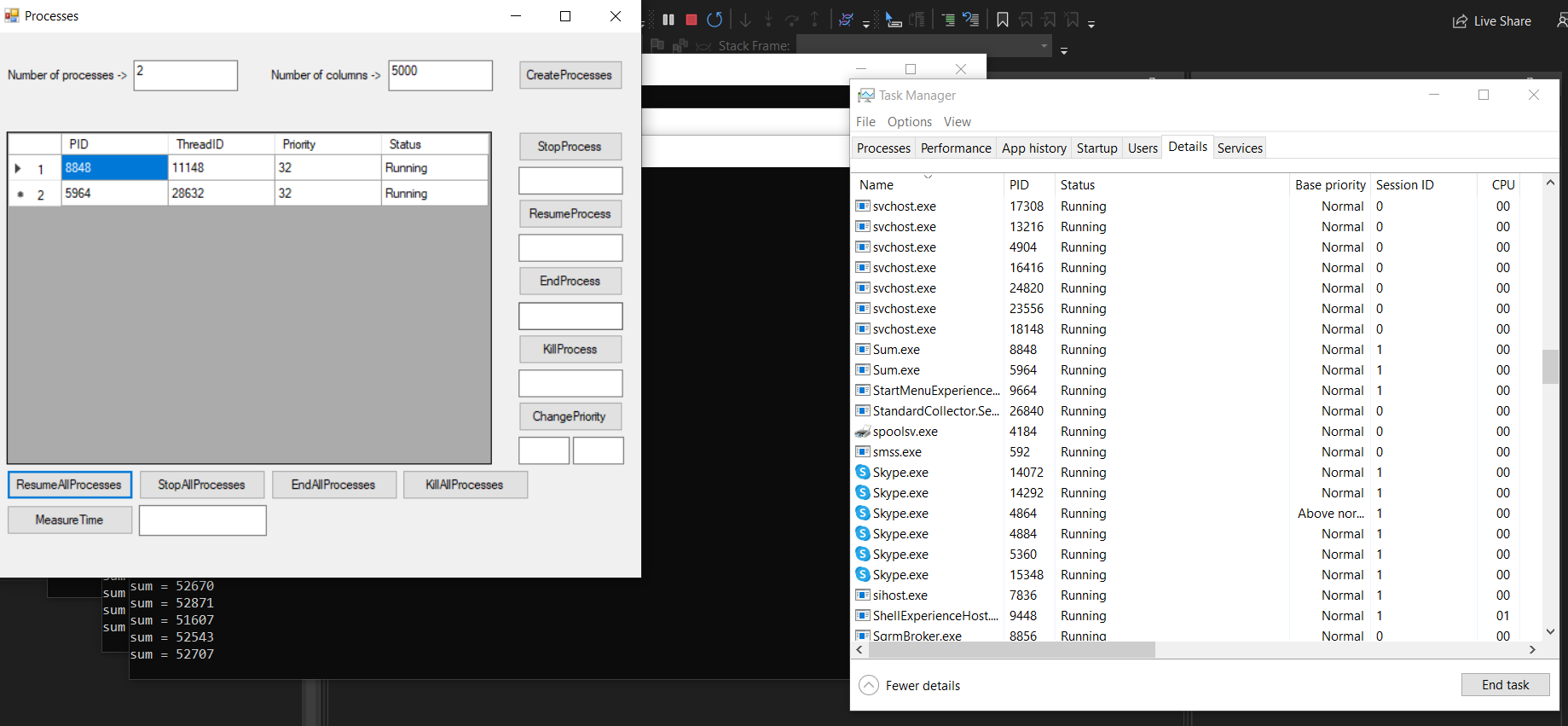


Рис. 3 Створення 2 процесів та запущені процеси на формі програми та у Task Manager, загальні характеристики, такі як PID, ThreadId, Status , Priority.

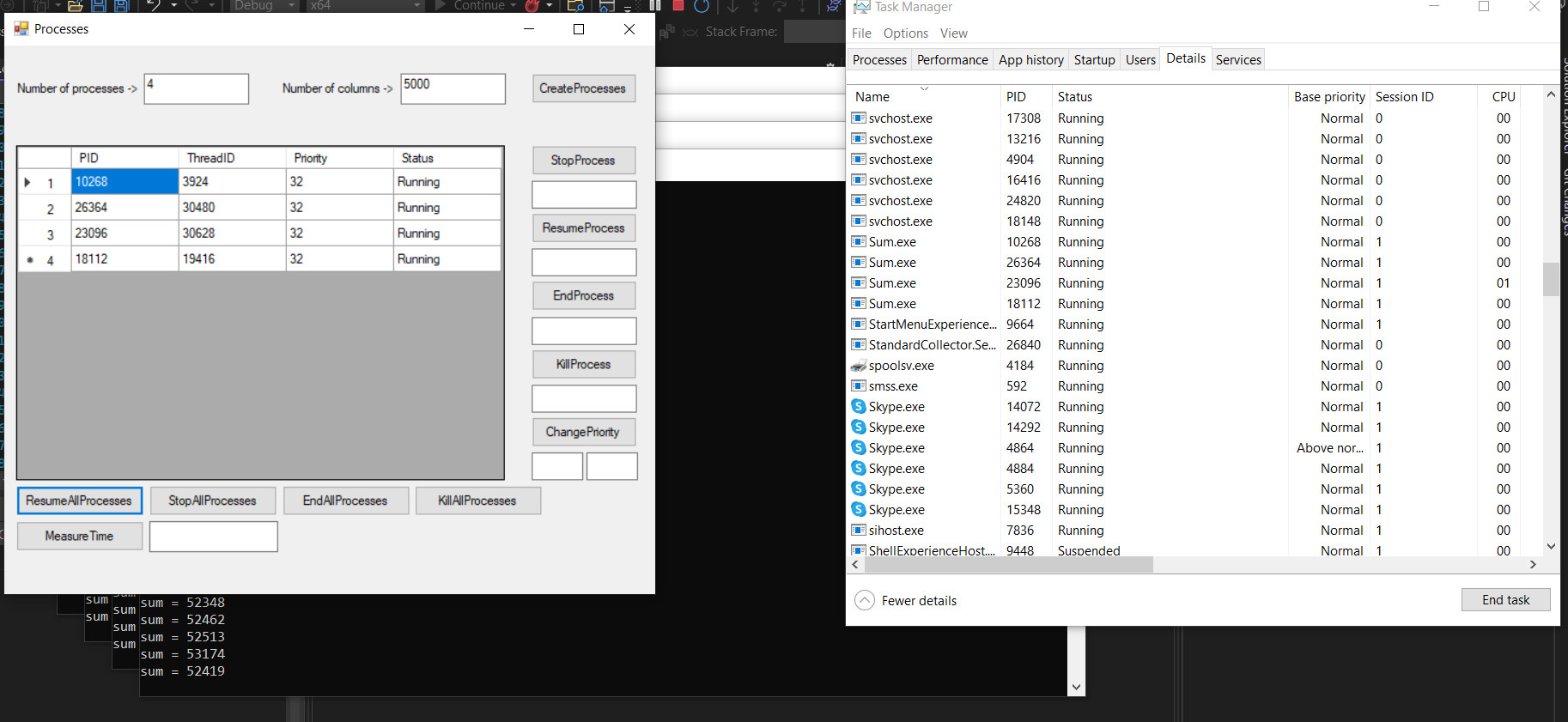


Рис. 4 Створення та запуск 4 процесів за допомогою CreateProcess() i ResumeAllProcesses().

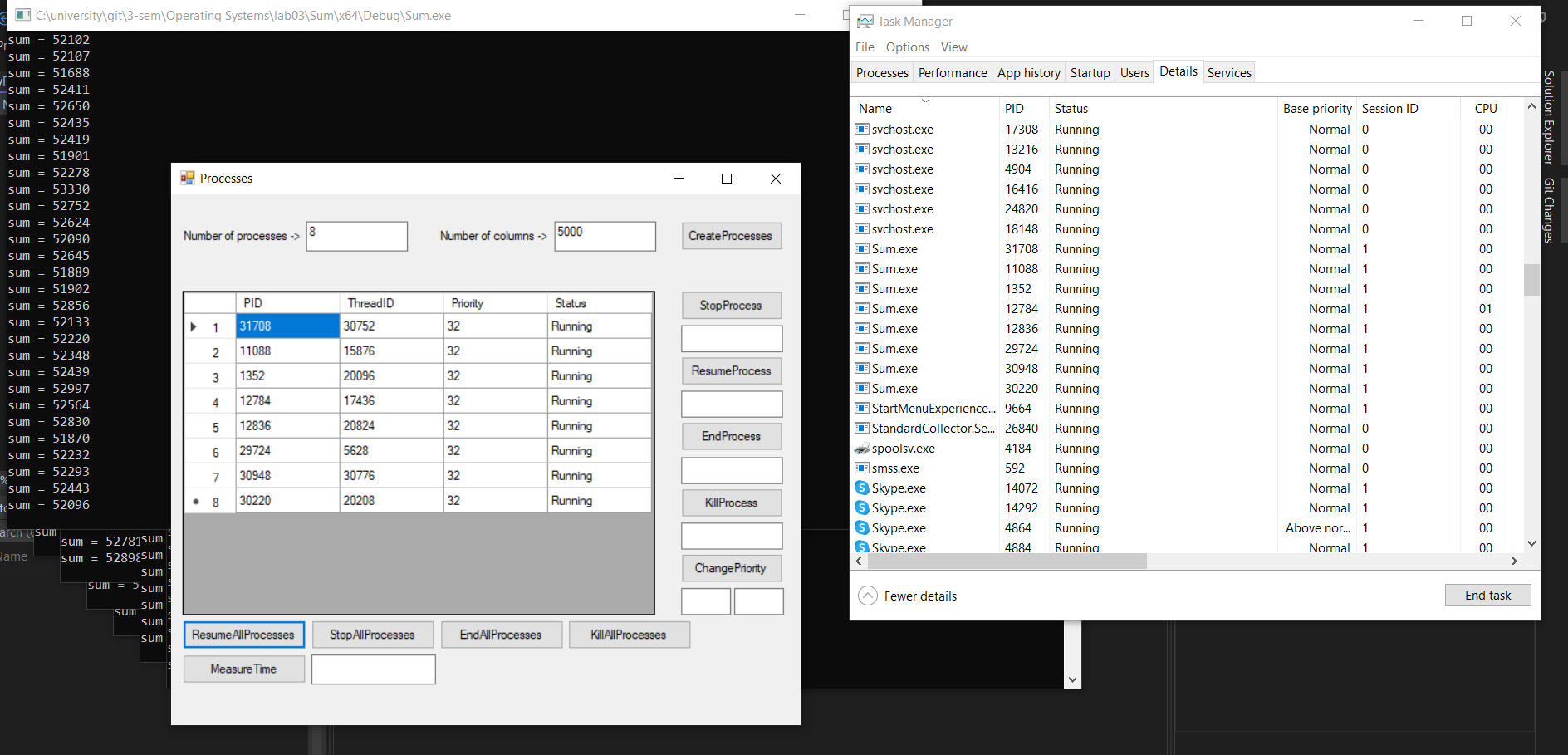


Рис. 5 Створення та запуск 8 процесів за допомогою CreateProcess() i ResumeAllProcesses().

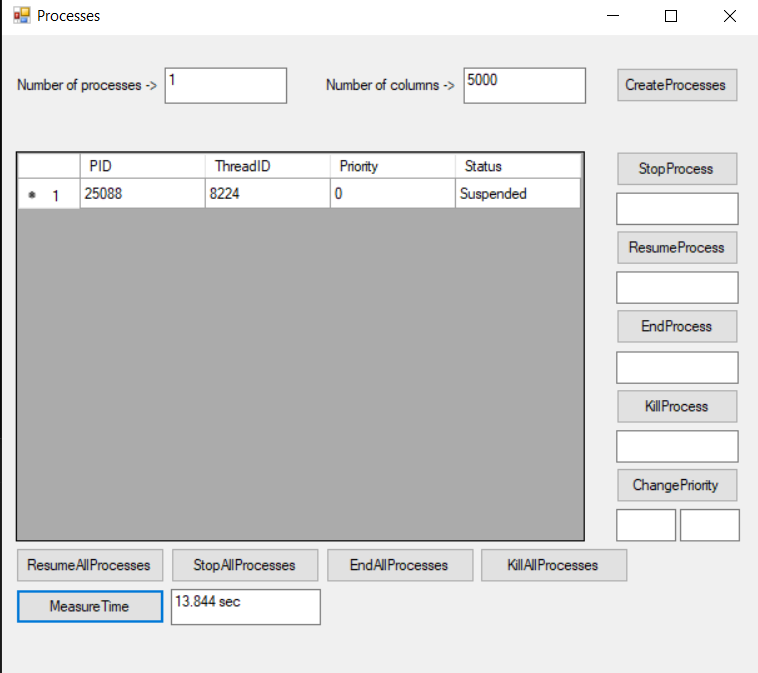


Рис. 6 Вимірювання часу виконання 1 процесу за допомогою MeasureTime().

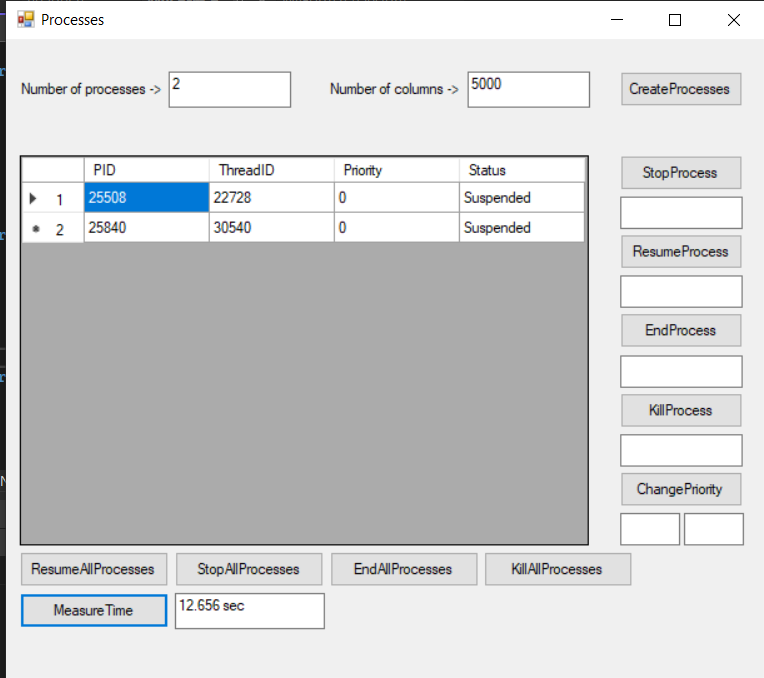


Рис. 7 Вимірювання часу виконання 2 процесів за допомогою MeasureTime().

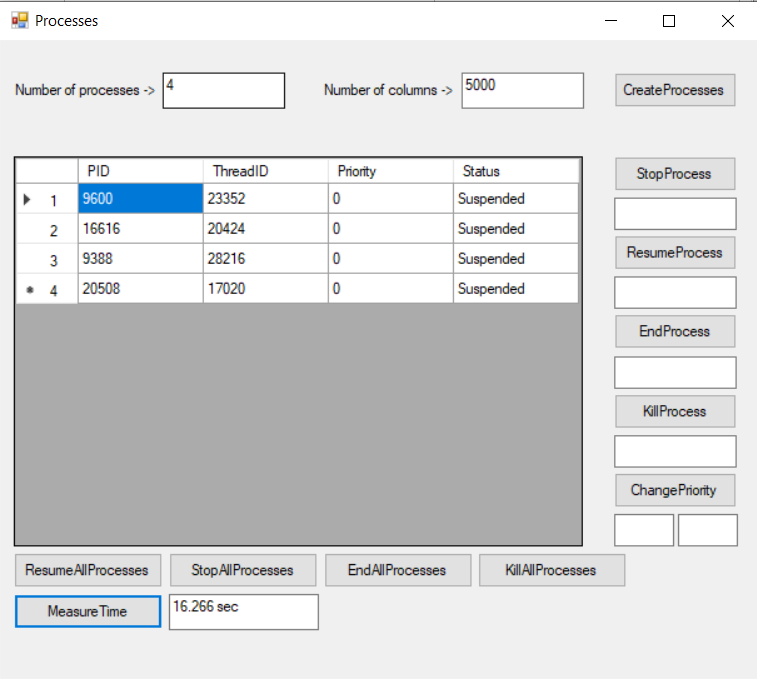


Рис. 8 Вимірювання часу виконання 4 процесів за допомогою MeasureTime().

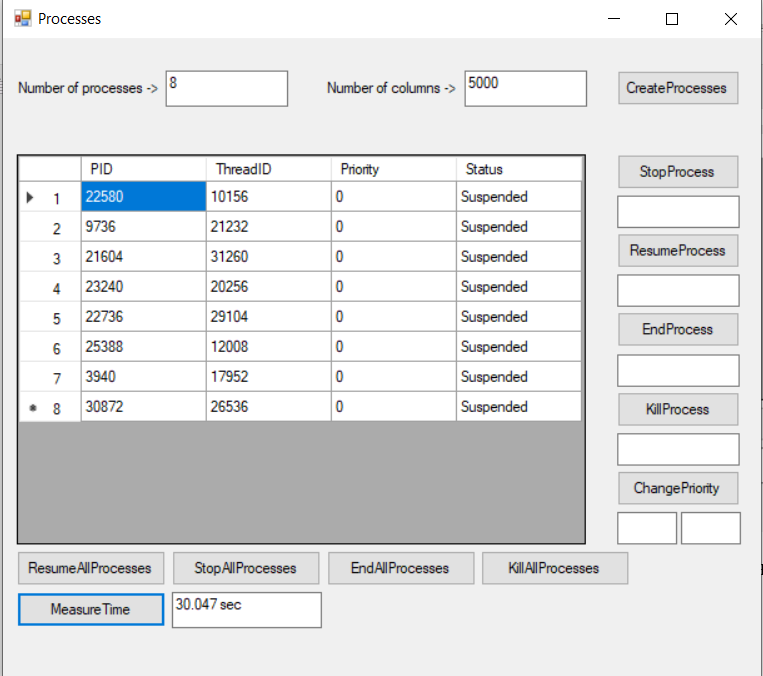


Рис. 9 Вимірювання часу виконання 8 процесів за допомогою MeasureTime().

|  |  |
| --- | --- |
| Кількість процесів | Час виконання |
| 1 процес | 13.844 sec |
| 2 процеси | 12.656 sec |
| 4 процеси | 16.266 sec |
| 8 процесів | 30.047 sec |

Таблиця порівняння кількості процесів і часу їхнього виконання в секундах. З результатів можна зробити висновок, що 1-2 процеси мають приблизно однаковий час виконання, 4 процеси лише на трохи збільшують час, в той час як час виконання 8 процесів суттєво збільшується.

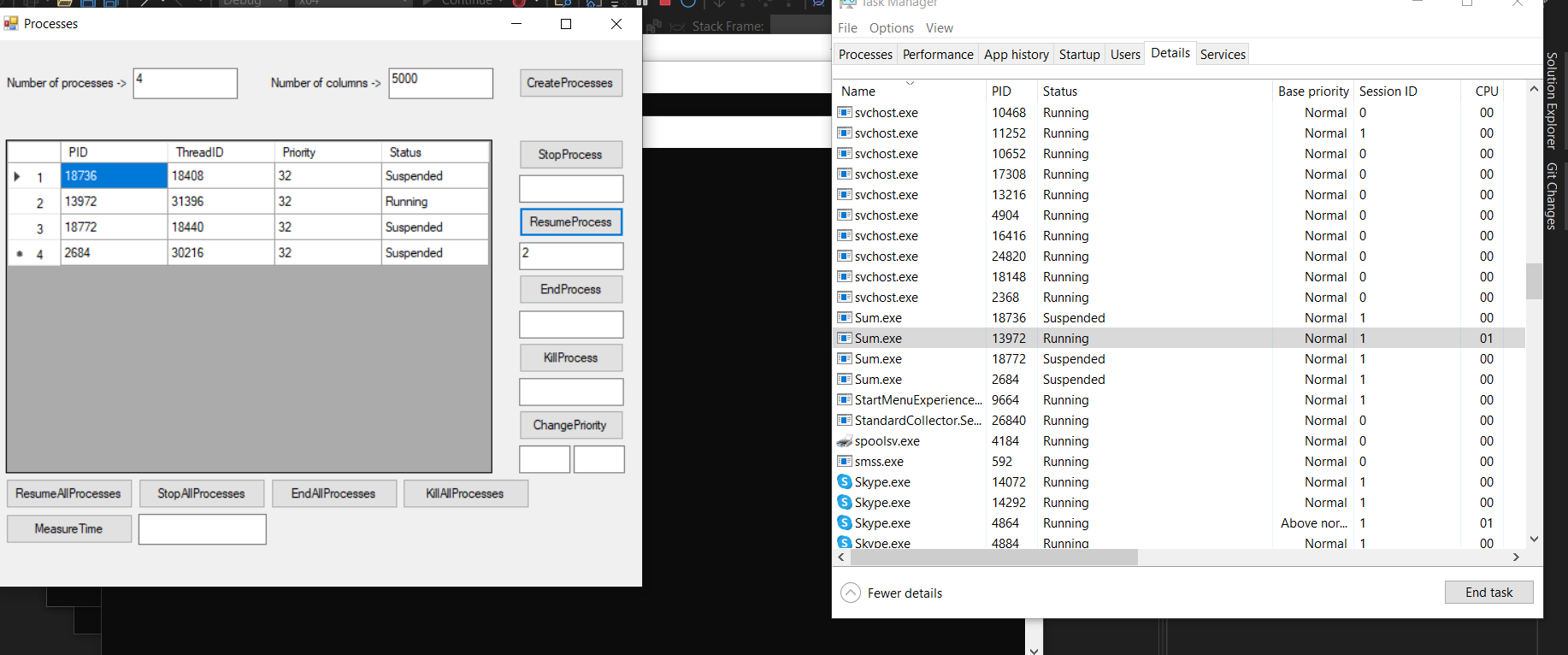


Рис. 10 Відновлення призупиненого процесу за допомогою ResumeProcess().

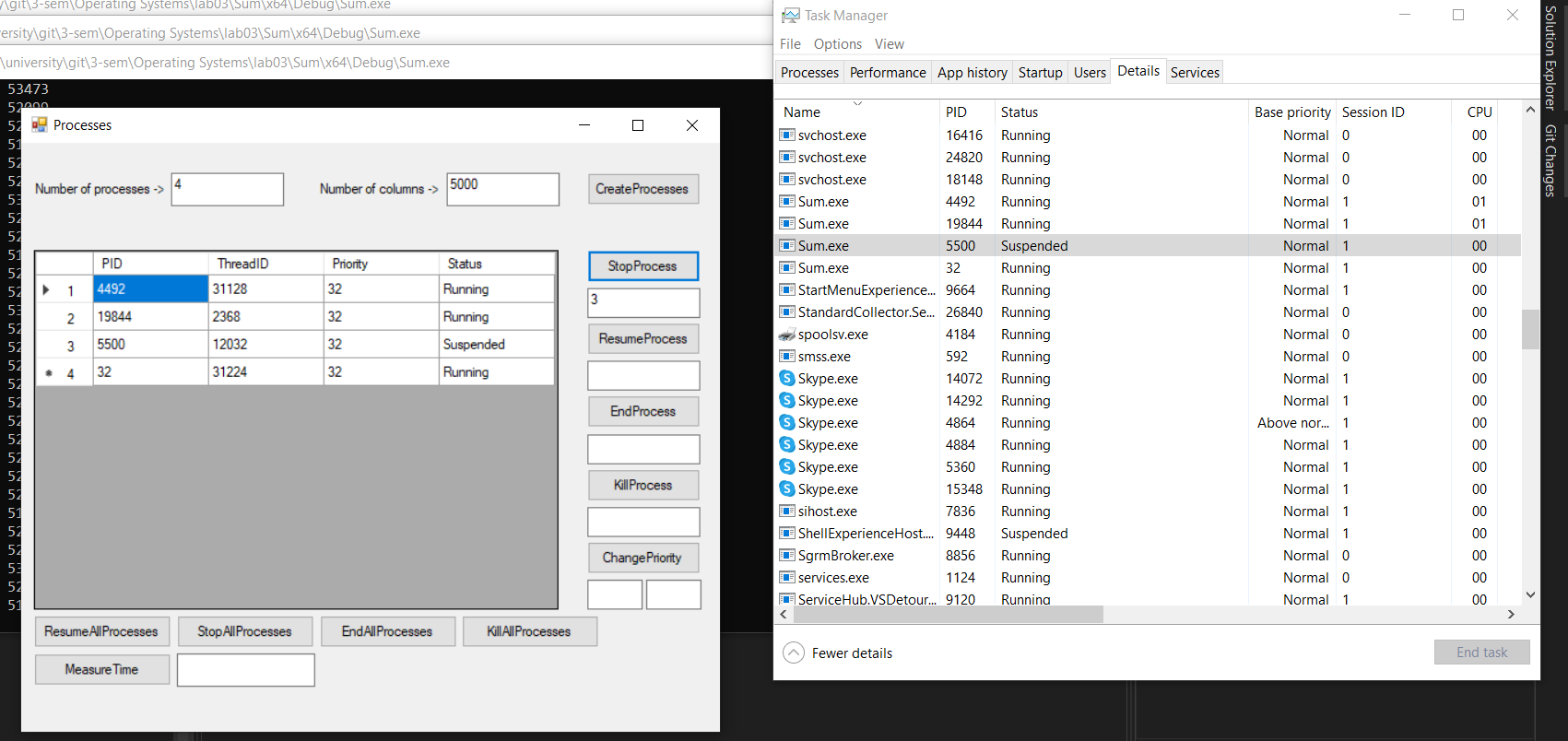


Рис. 11 Призупинення процесу.

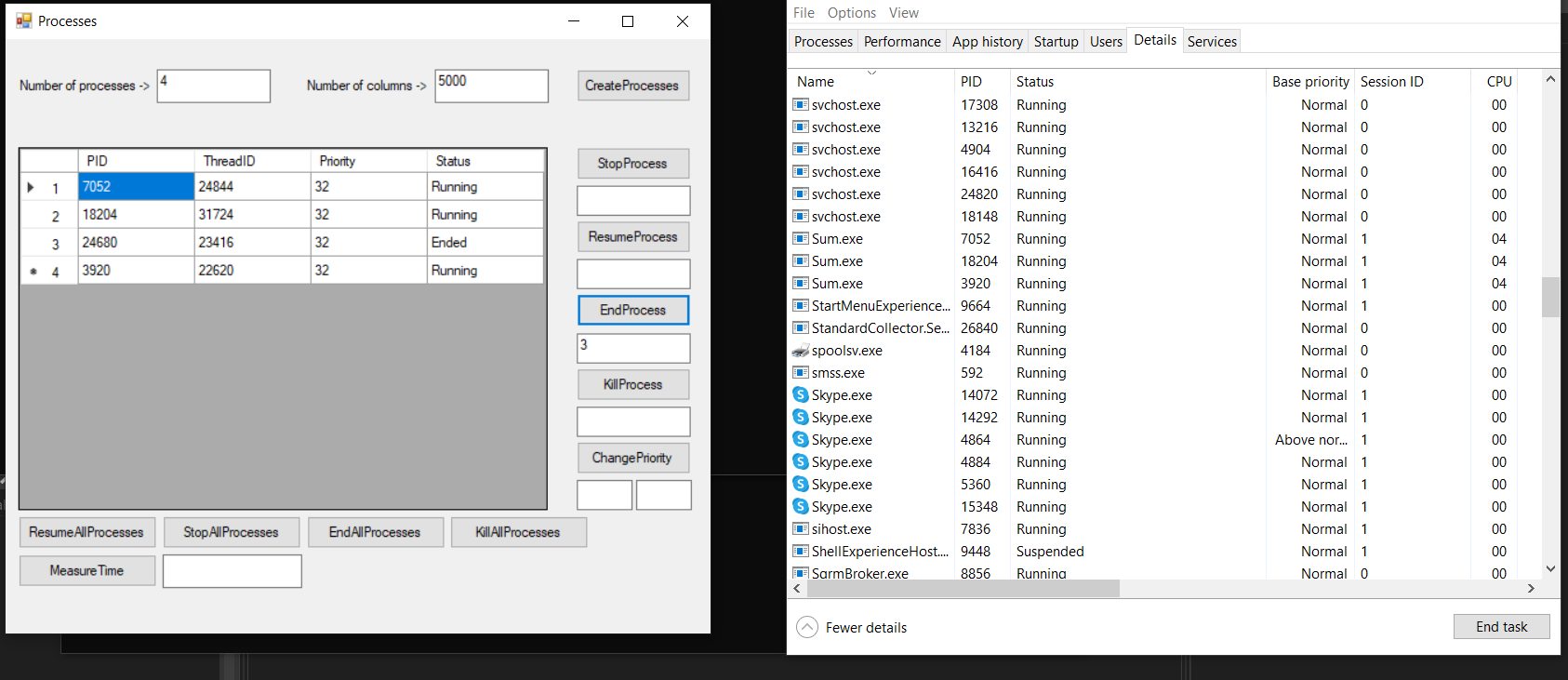


Рис. 12 Завершення процесу за допомогою EndProcess().

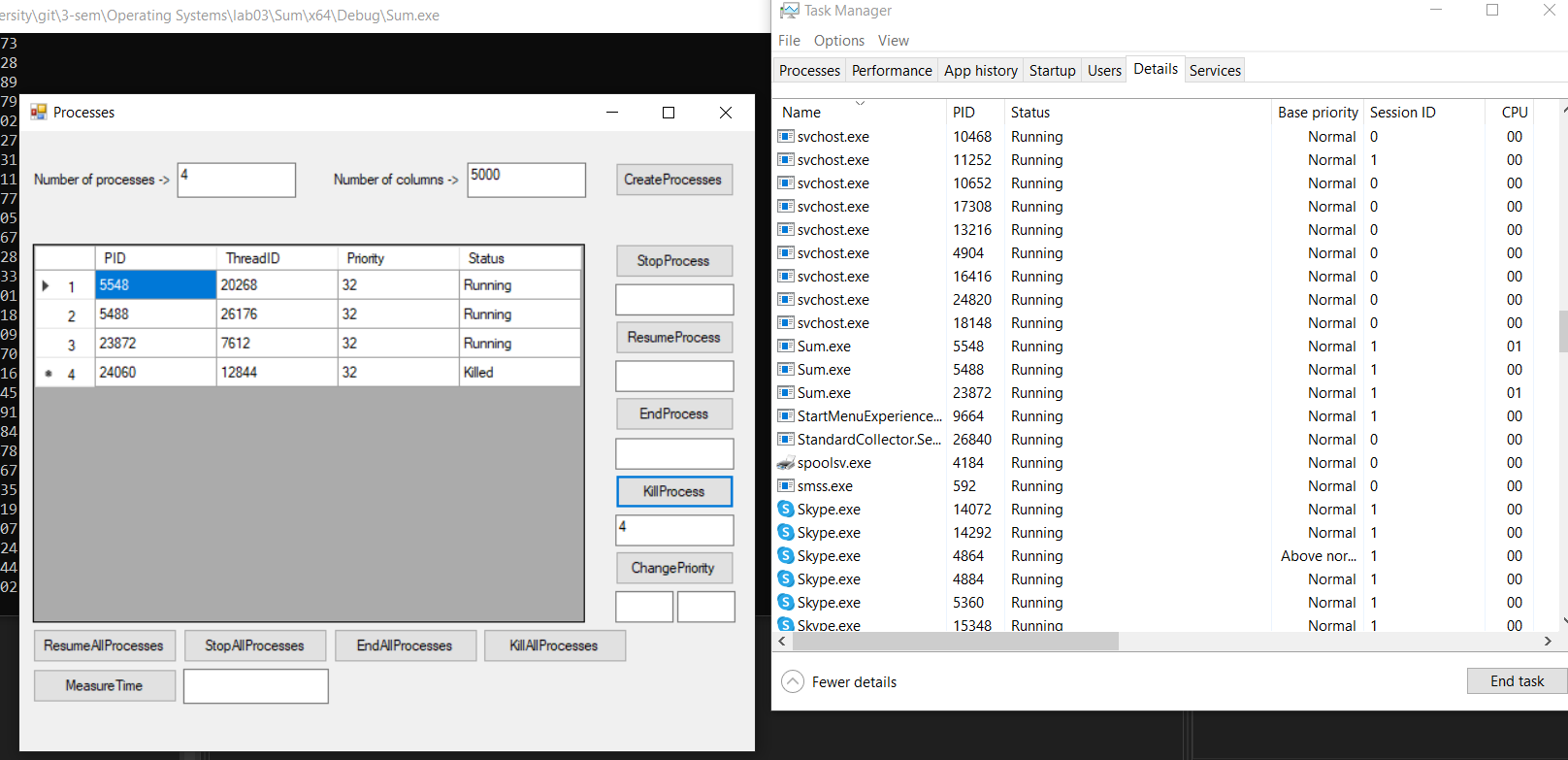


Рис. 13 Вбиття процесу за допомогою KillProcess().

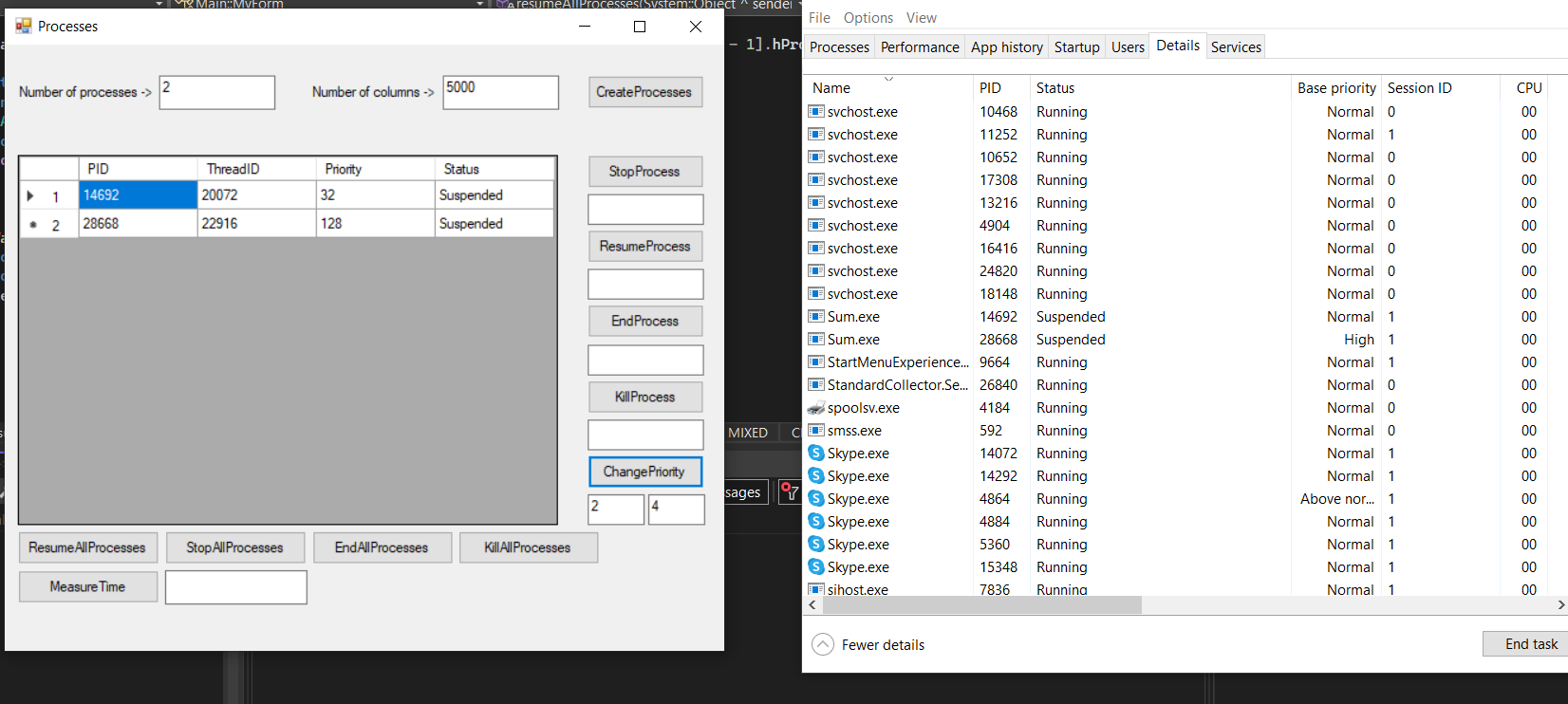
****

Рис. 14 Зміна пріоритету за допомогою ChangePriority() з Normal до High.

**Висновок**

На цій лабораторній роботі я навчилась працювати з WinApi-функціями, а саме навчилась створювати процеси, їх зупиняти, відновлювати, закінчувати, вбивати та змінювати пріоритет виконання процесів. Також виміряла час виконання 1,2,4,8 процесів та порівняна результати у таблиці. Реалізувала форму у Visual Studio 2022 і наглядно показала там результати роботи програми.