**МІНІСТЕРСТВО ОСВІТИ І НАУКИ  УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"**

**ІКНІ**

Кафедра **ПЗ**

**ЗВІТ**

до лабораторної роботи № 3

**На тему:** *“ Робота з процесами в ОС Windows”*

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

**Лектор:**

ст. викладач каф. ПЗ

Грицай О.Д.

**Виконала:**

ст. гр. ПЗ-26

Матолінець Л.А.

**Прийняв:**

доцент каф. ПЗ

Горечко О.М.

« \_\_\_\_ » \_\_\_\_\_\_\_\_  2022 р.

∑= \_\_\_\_                              .

Львів – 2022

**Тема роботи:** Робота з процесами в ОС Windows.

**Мета роботи:** Ознайомитися з багатопоточністю в ОС Windows. Навчитися працювати з процесами, використовуючи WinAPI-функції.

**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)

**Теоретичні відомості**

Процес - завантажена в пам'ять і готова до виконання програма.

Кожен процес має свій власний віртуальний адресний простір (4Gb). Процес складається з коду, даних і інших системних ресурсів, таких як відкриті файли, канали (pipes), що синхронізують об'єкти.

Потік (thread) - базовий об'єкт, якому операційна система розподіляє час центрального процесора.

Виконання процесу починається зі стартового потоку. Надалі він може породжувати інші потоки. Ресурси процесу доступні всім його потокам. Кожен потік використовує структуру даних, для збереження*контексту виконання*, у той час, коли в нього віднімається процесор. У контекст входять регістри процесора, перемінні оточення, стеки ядра і користувача. Усі потоки одного процесу спільно використовують його віртуальний адресний простір.

У Win32® API визначені наступні функції роботи з процесами і потоками

|  |  |
| --- | --- |
| **Назва функції** | **Виконувана дія** |
| AttachThreadInput | Переключення механізмів уведення з однієї нитки на іншу |
| CommandLineToArgv | Робить розбір командного рядка в Unicode |
| CreateProcess | Створює процес |
| CreateRemoteThread | Створює потік в адресному просторі іншого процесу |
| CreateThread | Створює потік |
| ExitProcess | Завершує процес і весь його потоки |
| ExitThread | Завершує потік |
| FreeEnvironmentStrings | Звільняє пам'ять області перемінні середовища |
| GetCommandLine | Повертає покажчик на командний рядок |
| GetCurrentProcess | Повертає описувач (handle) поточного процесу |
| GetCurrentProcessId | Повертає ідентифікатор поточного процесу |
| GetCurrentThread | Повертає описувач (handle) поточного потоку |
| GetCurrentThreadId | Повертає ідентифікатор поточного потоку |
| GetEnvironmentStrings | Повертає рядок перемінні середовища |
| GetEnvironmentVariable | Повертає значення зазначеного перемінного середовища |
| GetExitCodeProcess | Повертає код завершення процесу |
| GetExitCodeThread | Повертає код завершення потоку |
| GetPriorityClass | Повертає клас пріоритету процесу |
| GetProcessAffinityMask | Повідомляє, на яких процесорах дозволене виконання процесу |
| GetProcessShutdownParameters | Повідомляє параметри поводження процесу при завершенні роботи системи |
| GetProcessTimes | Повертає тимчасові характеристики зазначеного процесу |
| GetProcessVersion | Повідомляє версію Windows, для якої призначений процес |
| GetProcessWorkingSetSize | Повертає характеристики доступного процесу адресного простору |
| GetStartupInfo | Повертає параметри процесу, отримані їм при створенні |
| GetThreadPriority | Повідомляє пріоритет зазначеного потоку |
| GetThreadTimes | Повертає тимчасові характеристики зазначеного потоку |
| OpenProcess | Повертає описувач (handle) зазначеного процесу |
| ResumeThread | Зменшує лічильник затримок потоку (чи запускає його) |
| SetEnvironmentVariable | Установлює значення зазначеного перемінного середовища |
| SetPriorityClass | Установлює клас пріоритету процесу |
| SetProcessShutdownParameters | Установлює параметри поводження процесу при завершенні роботи системи |
| SetThreadAffinityMask | Установлює, на яких процесорах дозволене виконання потоку |
| SetThreadPriority | Установлює пріоритет зазначеного потоку |
| Sleep | Затримує виконання потоку на зазначену кількість мілісекунд |
| SleepEx | Затримує виконання до настання події введення/ чивисновку на час |
| SetProcessWorkingSetSize | Установлює характеристики доступного процесу адресного простору |
| SuspendThread | Припиняє виконання зазначеного потоку |
| TerminateProcess | Завершує зазначений процес |
| TerminateThread | Завершує зазначений потік |
| TlsAlloc | Розподіляє індекс локальної пам'яті потоку (thread local storage TLS) |
| TlsFree | Звільняє індекс TLS |
| TlsGetValue | Повертає дані, розміщені в TLS із зазначеним індексом |
| TlsSetValue | Поміщає дані в TLS із зазначеним індексом |
| WaitForInputIdle | Чекає, поки не почнеться введення для зазначеного процесу |
| WinExec | Виконує зазначений додаток |

**Хід роботи**

1. Створюю віконне застосування для керування процесами:

#pragma endregion

PROCESS\_INFORMATION\* pi = new PROCESS\_INFORMATION[8];

int processesAmount;

String^ getPriority(HANDLE handle)

{

int priority = GetPriorityClass(handle);

String^ normal = "Normal";

String^ above\_normal = "Above Normal";

String^ below\_normal = "Below Normal";

String^ idle = "Idle";

String^ real\_time = "Real Time";

String^ high = "High";

String^ unknown = "Unknown";

switch (priority)

{

case NORMAL\_PRIORITY\_CLASS: return normal;

case ABOVE\_NORMAL\_PRIORITY\_CLASS: return above\_normal;

case HIGH\_PRIORITY\_CLASS: return high;

case BELOW\_NORMAL\_PRIORITY\_CLASS: return below\_normal;

case IDLE\_PRIORITY\_CLASS: return idle;

case REALTIME\_PRIORITY\_CLASS: return real\_time;

default: return unknown;

}

}

String^ getStatus(HANDLE handle)

{

DWORD status = WaitForSingleObject(handle, 0);

switch (status)

{

case WAIT\_OBJECT\_0: return "Terminated";

default:

int value = SuspendThread(handle);

ResumeThread(handle);

if (value == 0) {

return "Running";

}

return "Suspended";

}

}

private: System::Void button11\_Click(System::Object^ sender, System::EventArgs^ e) {

processesAmount = System::Convert::ToUInt32(textBox1->Text);

std::fstream fclear("fileOC.txt", std::fstream::trunc | std::fstream::out);

fclear.close();

std::fstream fEnd("End.txt", std::fstream::out | std::fstream::trunc);

fEnd << "0 1";

fEnd.close();

TCHAR szCmdLine[] = TEXT("C:\\Users\\HP\\source\\repos\\Tailor\\x64\\Debug\\Tailor.exe");

int n = System::Convert::ToUInt32(textBox1->Text);

long long int A = System::Convert::ToUInt32(textBox1->Text);

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

STARTUPINFO si;

PROCESS\_INFORMATION pinfo;

ZeroMemory(&si, sizeof(STARTUPINFO));

ZeroMemory(&pinfo, sizeof(PROCESS\_INFORMATION));

if (auto hr = CreateProcess(

NULL,

szCmdLine,

nullptr,

nullptr,

FALSE,

CREATE\_NEW\_CONSOLE,

nullptr,

nullptr,

&si,

&pinfo)) {

pi[i] = pinfo;

dataGridView1->RowCount += 1;

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

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

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

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

}

}

}

private: System::Void button7\_Click(System::Object^ sender, System::EventArgs^ e) {

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

CloseHandle((pi[i]).hProcess);

}

}

private: System::Void button4\_Click(System::Object^ sender, System::EventArgs^ e) {

int processNumber = System::Convert::ToInt32(textBox3->Text);

SetPriorityClass(pi[processNumber].hProcess, NORMAL\_PRIORITY\_CLASS);

dataGridView1->Rows[processNumber]->Cells[2]->Value = getPriority(pi[processNumber].hProcess);

}

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

int processNumber = System::Convert::ToInt32(textBox3->Text);

SetPriorityClass(pi[processNumber].hProcess, ABOVE\_NORMAL\_PRIORITY\_CLASS);

dataGridView1->Rows[processNumber]->Cells[2]->Value = getPriority(pi[processNumber].hProcess);

}

private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e) {

int processNumber = System::Convert::ToInt32(textBox3->Text);

SetPriorityClass(pi[processNumber].hProcess, BELOW\_NORMAL\_PRIORITY\_CLASS);

dataGridView1->Rows[processNumber]->Cells[2]->Value = getPriority(pi[processNumber].hProcess);

}

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

int processNumber = System::Convert::ToInt32(textBox3->Text);

SetPriorityClass(pi[processNumber].hProcess, HIGH\_PRIORITY\_CLASS);

dataGridView1->Rows[processNumber]->Cells[2]->Value = getPriority(pi[processNumber].hProcess);

}

private: System::Void button6\_Click(System::Object^ sender, System::EventArgs^ e) {

int processNumber = System::Convert::ToInt32(textBox3->Text);

SetPriorityClass(pi[processNumber].hProcess, IDLE\_PRIORITY\_CLASS);

dataGridView1->Rows[processNumber]->Cells[2]->Value = getPriority(pi[processNumber].hProcess);

}

private: System::Void button5\_Click(System::Object^ sender, System::EventArgs^ e) {

int processNumber = System::Convert::ToInt32(textBox3->Text);

SetPriorityClass(pi[processNumber].hProcess, REALTIME\_PRIORITY\_CLASS);

dataGridView1->Rows[processNumber]->Cells[2]->Value = getPriority(pi[processNumber].hProcess);

}

private: System::Void button9\_Click(System::Object^ sender, System::EventArgs^ e) {

int processNumber = System::Convert::ToInt32(textBox3->Text);

ResumeThread(pi[processNumber].hThread);

dataGridView1->Rows[processNumber]->Cells[1]->Value = getStatus(pi[processNumber].hProcess);

dataGridView1->Rows[processNumber]->Cells[2]->Value = getPriority(pi[processNumber].hProcess);

}

private: System::Void button14\_Click(System::Object^ sender, System::EventArgs^ e) {

int processNumber = System::Convert::ToInt32(textBox3->Text);

SuspendThread(pi[processNumber].hThread);

dataGridView1->Rows[processNumber]->Cells[1]->Value = getStatus(pi[processNumber].hProcess);

dataGridView1->Rows[processNumber]->Cells[2]->Value = getPriority(pi[processNumber].hProcess);

}

private: System::Void button8\_Click(System::Object^ sender, System::EventArgs^ e) {

int processNumber = System::Convert::ToInt32(textBox3->Text);

TerminateProcess(pi[processNumber].hProcess, 0);

CloseHandle(pi[processNumber].hThread);

CloseHandle(pi[processNumber].hProcess);

dataGridView1->Rows[processNumber]->Cells[1]->Value = getStatus(pi[processNumber].hProcess);

dataGridView1->Rows[processNumber]->Cells[2]->Value = getPriority(pi[processNumber].hProcess);

}

private: System::Void button10\_Click(System::Object^ sender, System::EventArgs^ e) {

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

ResumeThread(pi[i].hThread);

dataGridView1->Rows[i]->Cells[1]->Value = getStatus(pi[i].hProcess);

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

}

}

private: System::Void button15\_Click(System::Object^ sender, System::EventArgs^ e) {

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

SuspendThread(pi[i].hThread);

dataGridView1->Rows[i]->Cells[1]->Value = getStatus(pi[i].hProcess);

}

}

private: System::Void button12\_Click(System::Object^ sender, System::EventArgs^ e) {

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

TerminateProcess(pi[i].hProcess, 0);

dataGridView1->Rows[i]->Cells[1]->Value = getStatus(pi[i].hProcess);

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

}

}

private: System::Void textBox3\_TextChanged(System::Object^ sender, System::EventArgs^ e) {

}

private: System::Void button16\_Click(System::Object^ sender, System::EventArgs^ e) {

int processNumber = System::Convert::ToInt32(textBox3->Text);

ExitProcess(100);

dataGridView1->Rows[processNumber]->Cells[1]->Value = getStatus(pi[processNumber].hProcess);

dataGridView1->Rows[processNumber]->Cells[2]->Value = getPriority(pi[processNumber].hProcess);

}

private: System::Void button17\_Click(System::Object^ sender, System::EventArgs^ e) {

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

{

FILETIME creationTime;

FILETIME exitTime;

FILETIME kernelTime;

FILETIME userTime;

String^ status = getStatus(pi[i].hProcess);

GetProcessTimes(pi[i].hProcess,

(FILETIME\*)&creationTime,

(FILETIME\*)&exitTime,

(FILETIME\*)&kernelTime,

(FILETIME\*)&userTime);

unsigned long long cr = (((unsigned long long) creationTime.dwHighDateTime) << 32) + creationTime.dwLowDateTime;

unsigned long long ex = (((unsigned long long) exitTime.dwHighDateTime) << 32) + exitTime.dwLowDateTime;

unsigned long long ke = (((unsigned long long) kernelTime.dwHighDateTime) << 32) + kernelTime.dwLowDateTime;

unsigned long long us = (((unsigned long long) userTime.dwHighDateTime) << 32) + userTime.dwLowDateTime;

if (status != "Terminated") {

dataGridView1->Rows[i]->Cells[3]->Value = "Not Terminated";

dataGridView1->Rows[i]->Cells[4]->Value = (ke \* pow(10.0, -7)) + "с.";

dataGridView1->Rows[i]->Cells[4]->Value = (us \* pow(10.0, -7)) + "с.";

}

else {

dataGridView1->Rows[i]->Cells[3]->Value = ((ex - cr) \* pow(10.0, -7)) + "с.";

dataGridView1->Rows[i]->Cells[4]->Value = (ke \* pow(10.0, -7)) + "с.";

dataGridView1->Rows[i]->Cells[5]->Value = (us \* pow(10.0, -7)) + "с.";

}

}

}

private: System::Void button13\_Click(System::Object^ sender, System::EventArgs^ e) {

}

};

}

1. Створюю консольний застосунок для табуляції функції f = cos(x) в ряд Тейлора у проміжку від 0.3 до 0.7

Tailor.cpp:

#include <iostream>

#include <windows.h>

#include <string>

#include <chrono>

#include <iostream>

#include <fstream>

#include <stdlib.h>

#include <stdio.h>

#include <math.h>

const int nMaxTerms = 20;

double factorials[nMaxTerms];

double factorial(int n)

{

if (n == 1)

return 1;

else

return (double)n \* factorial(n - 1.0);

}

void precalcFactorials()

{

for (int i = 1; i < nMaxTerms + 1; i++)

{

factorials[i - 1] = factorial(i);

}

}

double taylorCos(double rads)

{

double result = 1.0;

for (int curTerm = 1; curTerm <= (nMaxTerms / 2) - 1; curTerm++)

{

double curTermValue = pow(rads, (curTerm \* 2));

curTermValue /= factorials[(curTerm \* 2) - 1];

if (curTerm & 0x01)

result -= curTermValue;

else

result += curTermValue;

}

return result;

}

int main()

{

double a = 0.3;

double b = 0.7;

int parts;

precalcFactorials();

std::cout << "Enter parts";

std::cin >> parts;

double step = (b - a) / parts;

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

std::cout << "x ="<< a<<" ";

std::cout << "taylorCos =" << taylorCos(a) << "\n";

a += step;

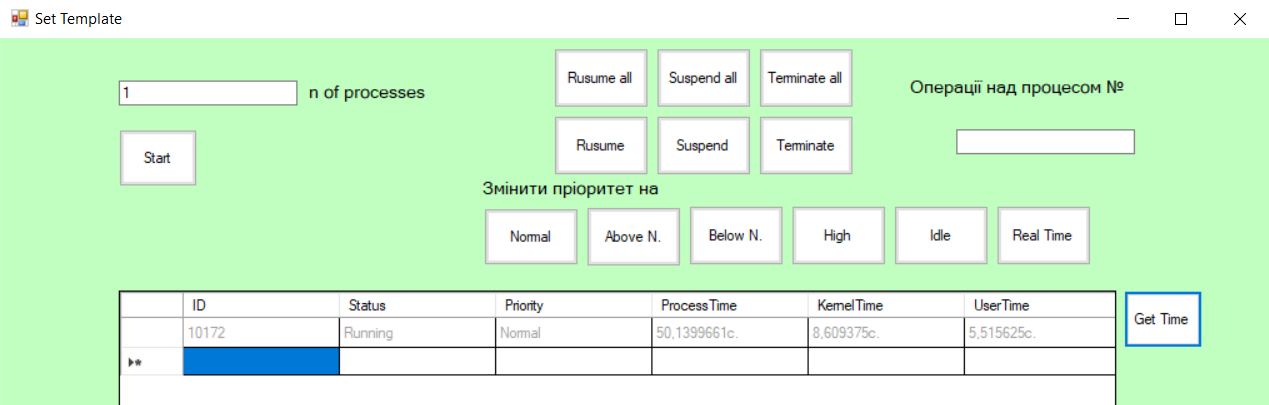
}

return 1;

}

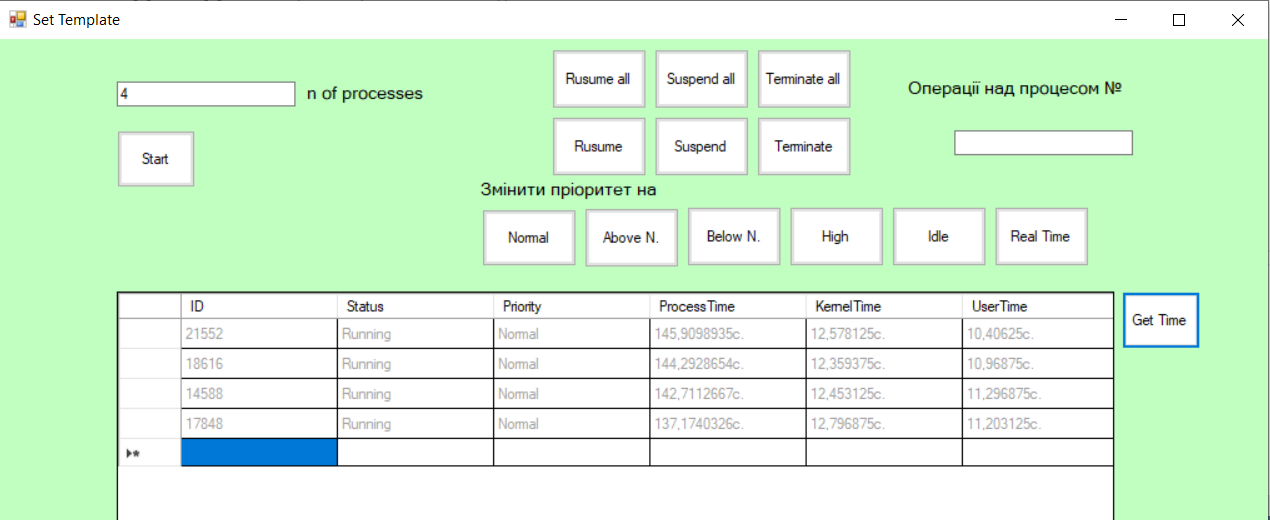
**Результати виконання**

1. Реалізовую створення 1 процесу і в ньому здійснюю розв’язок індивідуального завдання, кількість кроків - 100 000.



*Рис.1 Результат виконання програми*

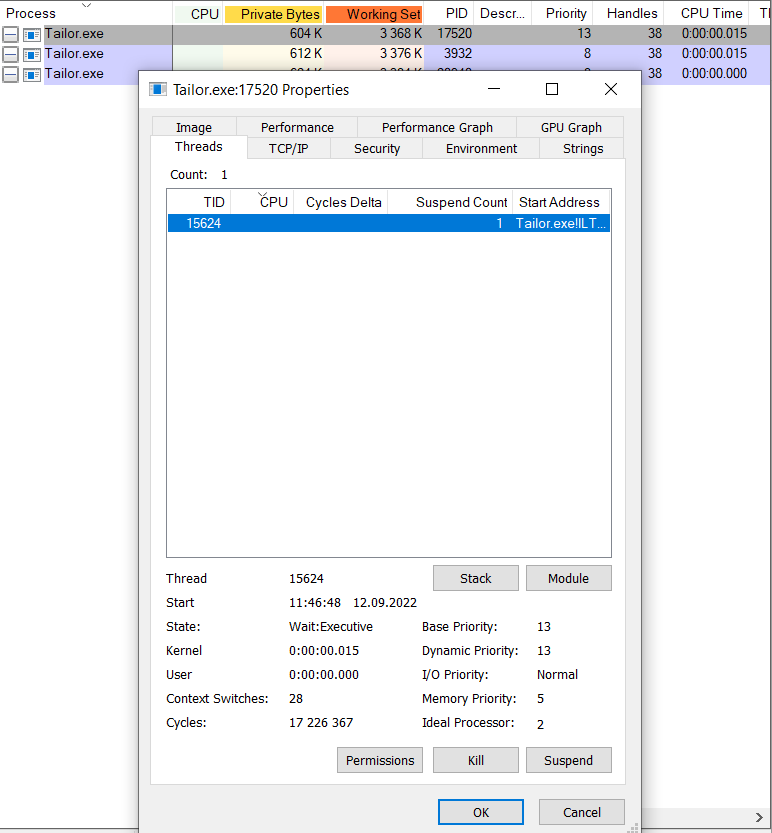
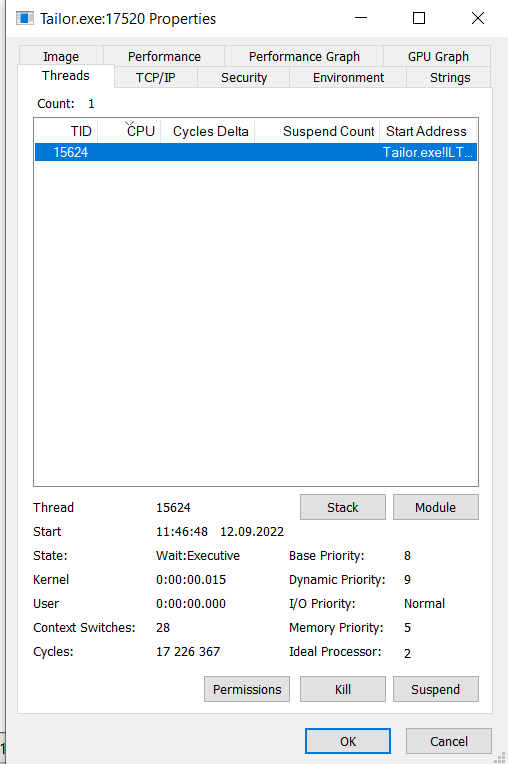
1. Запускаю 4 однакові процеси (кількість кроків – 100 000). За допомогою функції WinAPI вимірюю час виконання процесів.



*Рис.2 Результат виконання програми*

З рисунків видно, що час збільшився майже втричі: від 50с до 144с.

1. Для демонстрації наступних функцій варто прикріпити початковий стан одного з процесів та змінений варіант: я змінила пріоритет та призупинила процес.



*Рис.3 Результат виконання програми*

**Висновок**

На цій лабораторній роботі я познайомилася з функціями WinAPI, які дають можливість керувати процесами в ОС Windows. За допомогою них я реалізувала віконний застосунок, де продемонструвала можливості зупинки, відновлення та примусового завершення процесів. Весь моніторинг роботи програми я проводила у Process Explorer, що й навела у рисунках у ході роботи.

Результатом запуску Tailor.exe на декількох процесах можна вважати те, що зі збільшенням кількості процесів швидкодія програми зменшується. Часові показники наведені у ході роботи.