# МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Національний університет “Львівська політехніка”**



**Інститут післядипломної освіти**

**ЗВІТ**

**Про виконання лабораторної роботи №2**

**«Робота з потоками в ОС Windows»**

**з дисципліни «Операційні системи»**

Виконав:

слухач групи ПЗС-11

Гринчук Тарас

Прийняв:

доц. Яковина В.С.

« »\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2014 р.

∑ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

ЛЬВІВ – 2014

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

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

## 1. Теоретичні відомості

Розглянемо основні функції WinAPI призначені для роботи з процесами та потоками:

***CreateProcess()*** - створює новий процес і його первинний потік. Новий процес виконує вказаний виконуваний файл.

***CreateThread( )*** - створює новий потік в адресному просторі процесу. Формат функції:

*HANDLE CreateThread*

*( LPSECURITY\_ATTRIBUTES lpThreadAttributes ,*

*/ / Атрибути захисту потоку*

*DWORD dwStackSize , / / ​​розмір стека в байтах*

*LPTHREAD\_START\_ROUTINE lpStartAddress ,*

*/ / покажчик на функцію потоку*

*LPVOID lpParameter , / / ​​аргумент , переданий у функцію потоку*

*BDWORD dwCreationFlags , / / ​​прапори управління створенням потоку*

*LPDWORD lpThreadId ,*

*/ / Область пам'яті для повернення ідентифікатора потоку ); ...*

Функція повертає описувач породженого потоку . параметри:

*lpThreadAttributes* - покажчик на структуру , що описує параметри захисту потоку ; якщо параметру присвоєно значення NULL, то встановлюються атрибути «за замовчуванням»;

*dwStackSize* - встановлює розмір стека, який відводиться потоку; Якщо параметр дорівнює нулю, то встановлюється стек, рівний стеку первинного потоку;

*lpStartAddress* - адреса функції, яку виконуватиме потік; функція має один 32-бітний аргумент і повертає 32-бітове значення;

*lpParameter* - параметр, переданий у функцію, яку буде використовувати потік;

*dwCreationFlags* - додатковий прапор, який управляє створенням потоку; якщо цей параметр дорівнює *CREATE\_SUSPENDED*, то потік після створення запускається на виконання до виклику функції *ResumeThread*;

*lpThreadId* - покажчик на 32-бітну змінну, якій буде присвоєно значення унікального ідентифікатора потоку.

**Зміна пріоритету процесу.**

Потоки , що володіють високим пріоритетом , займають більшу частину часу центрального процесора, раніше завершують свою роботу і здатні швидше реагувати на дії користувача. Якщо всім потокам буде присвоєнно однаково високий пріоритет, нічого доброго не вийде. Справа в тому, що якщо кільком потокам буде присвоєно один і той же пріоритет (НЕ має значення, високий або низький), то програма-планувальник виділить їм

однаковий час роботи центрального процесора, і сама ідея пріоритетів втратить сенс. Один потік зможе швидше реагувати сигналами на сигнали тільки в тому випадку, якщо будуть сповільнені інші потоки. Це ж правило в рівній мірі можна застосувати і до процесів.

Наведені нижче функції перевіряють або змінюють базовий пріоритет потоку:

*BOOL SetThreadPriority*

*( HANDLE hThread / / дескриптор потоку*

*int iPriority ) ;/ / новий рівень пріоритету*

*int GetThreadPriority ( HANDLE hThread ); ...*

Функція ***SetThreadPriority()*** повертає значення TRUE у разі успішного завершення потоку, а значення FALSE - і при виникненні помилки. Функція ***GetThreadPriority()*** повертає значення, що визначає пріоритет.

Для позначення можливих значень пріоритету в обох функціях використовують набір констант:

* THREAD\_PRIORITY\_LOWESTCor - на два рівні нижче пріоритету процесу;
* THREAD\_PRIORITY\_BELOW\_NORMAL - на один рівень нижче пріоритету процесу;
* THREAD\_PRIORITY\_NORMAL - той же рівень пріоритету, що і у процесу;
* THREAD\_PRIORITY\_ABOVE\_NORMAL - на один рівень вище пріоритету процесу;
* THREAD\_PRIORITY\_HIGHEST - На два рівні вище пріоритету процесу;
* THREAD\_PRIORITY\_TIME\_CRITICAL - Рівень 15 (для звичайних користувацьких процесів) ;
* THREAD\_PRIORITY\_IDLE - Рівень 1 (для звичайних користувацьких процесів).

**Переривання і відновлення виконання потоків.**

Перерваний потік призупиняє своє виконання і не враховується при розподілі часу центрального процесора. Потік залишається в такому стані до тих пір, поки інший потік не відновить його виконання . Зупинку потоку можна призвести, зокрема, в тому випадку, якщо користувач перериває виконання певного завдання. До тих пір, поки завдання не буде скасоване, потік можна перевести в стан очікування. Якщо користувач вирішить продовжити роботу, потік відновить виконання з тієї точки, де він був зупинений . Щоб тимчасово припинити виконання потоків служать функції:

*DWORD SuspendThread ( HANDLE hThread );*

*DWORD ResumeThread ( HANDLE hThread ) .*

Один і той же потік можна послідовно зупинити кілька разів, не відновлюючи його виконання, однак кожної послідовної команді ***SuspendThread()*** повинна відповідати відповідна команда ***ResumeThread ().***

Система відраховує кількість скасованих команд за допомогою лічильника переривань. Кожна команда ***SuspendThread()*** інкременує значення лічильника, а кожна команда ***ResumeThread()*** декркменує його. Обидві функції повертають попереднє значення лічильника у вигляді параметра типу DWORD. Потік відновить своє виконання тільки в тому випадку, якщо лічильник приймезначення 0.

**Припинення виконання потоку.**

За аналогією з тим , як Windows-програма завершується по досягненні кінця функції main( ), потік зазвичай припиняє своє існування при досягненні кінця функції, в якій був початий. Коли він досягає кінця стартовою функції, система автоматично викликає команду ExitThread(), що має синтаксис:

*VOID ExitThread ( DWORD dwExitCode );*

Незалежно від того, як - явно або неявно (в результаті виконання оператора return ) - викликається функція ExitThread(), вона видаляє потік з черги програми-планувальника і знищує його стек. Однак сам об'єкт при цьому зберігаються. Тому навіть після припинення виконання потоку ви можете запросити його код завершення. По можливості дескриптор потоку слід закритиь явно (за допомогою функцій CloseHandle()), з тим щоб потік не займав лишній обсяг пам'яті. При закритті останнього дескриптора система автоматично знищує потік.

## 2. Хід роботи

## Завдання (варіант №5).

1. Створити окремий потік, і здійснити в ньому табулювання функції, задану розкладом в ряд Тейлора, в області її визначення на відрізку від А до В (кількість кроків не менше 100 000). Функція:



1. Реалізувати табулювання функцій у 2-ох, 4-ох, 8-ох потоках. Виміряти час роботи потоків та час роботи процесу за допомогою функцій WinAPI. Порівняти результати роботи в одному потоці і в багатьох.
2. Для кожного потоку реалізувати можливість його запуску, призупинення, продовження та зупинку.
3. Реалізувати можливість зміни пріоритету виконання процесу та потоків.
4. Продемонструвати результати виконання роботи, а також кількість створених потоків у “Диспетчері задач”, або подібних утилітах (н-д, ProcessExplorer).
5. Створимо потік в функції *main()* нашої програми наступним чином:

*int main() {*

*......................*

*HANDLE hThread[N]; //масив дескрипторів потоків*

*DWORD ThreadId[N]; //унікальні ідентифікатори потоків*

*//створюємо потоки*

*hThread[0] = CreateThread(NULL, 0, Output0, NULL, 0, &ThreadId[0]);*

*......................*

*return 0;*

*}*

Функція, в якій буде виконуватись потік *Output0(…)* матиме вигляд:

*//функція опрацювання 1-го потоку*

*DWORD WINAPI Output0(LPVOID Param) {*

*tabulation(0);*

*return 0;*

*}*

У функції *tabulation(…) -* буде виконуватись алгоритм табулювання, що має вигляд:

*//функція табулювання*

*void tabulation(int threadNum) { //threadNum - номер потоку*

*const double a = -1, b = 1.0; //(a;b) - проміжок табулювання*

*const int n = 100000; //кількість точок табулювання*

*const double eps = 0.0001; //точність обчислення*

*int i;*

*double x, s, si, y;*

*......................*

*for(int j = 1; j <= n; j++){*

*//отримуємо наступне значення x на проміжку табуляції*

*x = a+j\*(b-a)/(double) n;*

*//перший елемент ряду*

*si = x; s = si;*

*i = 2;*

*//виконуємо, поки не досягнуто бажаної точності*

*while(fabs(si)>eps) {*

*//обчислимо i-ий елемент ряду Тейлора*

*si \*= -x\*(i-1)/i;*

*//збільшимо суму ряду*

*s += si;*

*//збільшимо лічильник*

*i++;*

*}*

*//отримаємо значення ф-ї штатними методами*

*y=log(1+x);*

*printf("Потiк %d Крок %d: x=%f y=%f s=%f\n", threadNum+1, j, x, y, s);*

*}*

*......................*

*}*

1. Для реалізації табулювання у декількох потоків, використаємо наступний код:

const int N = 8; //к-сть потоків

*......................*

*if(N > 1) hThread[1] = CreateThread(NULL, 0, Output1, NULL, CREATE\_SUSPENDED, &ThreadId[1]);*

*if(N > 2) hThread[2] = CreateThread(NULL, 0, Output2, NULL, CREATE\_SUSPENDED, &ThreadId[2]);*

*......................*

*if(N > 7) hThread[7] = CreateThread(NULL, 0, Output7, NULL, CREATE\_SUSPENDED, &ThreadId[7]);*

Опрацювання цих потоків будемо здійснювати у функціях *Output1… Output7*, аналогічно як у п.1. Для обчислення часу роботи процесу та всіх потоків, створимо окремий клас:

*//клас для розрахунку часу роботи потоків*

*class Timing2 {*

*public:*

*//запуск таймера*

*void StartTiming();*

*//стоп таймера*

*void StopTiming();*

*//отримання значення таймера*

*double GetUserSeconds() const {*

*return value;*

*}*

*private:*

*DWORD\_PTR oldmask;*

*double value;*

*LARGE\_INTEGER time1;*

*};*

1. Запуск та відновлення роботи потоків, після призупинення, реалізуємо так:

*for(int i = 0; i < N; i++) ResumeThread(hThread[i]);*

Призупинення роботи потоків:

*for(int i = 0; i < N; i++) SuspendThread(hThread[i]);*

Завершуємо всі потоки наступним чином:

*DWORD ExitCode[N];*

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

*//отримуємо код завершення потоків*

*GetExitCodeThread(hThread[i], &ExitCode[i]);*

*//завершення потоку*

*TerminateThread(hThread[i], ExitCode[i]);*

*//закриття дексриптора потоку*

*CloseHandle(hThread[i]);*

*}*

1. Механізм встановлення пріоритету процесу та потоків, має вигляд:

*//пріоритет з яким виконується процес*

*const DWORD PROCESS\_PRIORITY = HIGH\_PRIORITY\_CLASS;*

*/\* Варіанти пріоритетів процесу:*

*REALTIME\_PRIORITY\_CLASS - реальний час*

*HIGH\_PRIORITY\_CLASS - високий*

*NORMAL\_PRIORITY\_CLASS - середнiй*

*IDLE\_PRIORITY\_CLASS - низький \*/*

*......................*

*int main() {*

*......................*

*//встановлюємо пріотитет процесу*

*SetPriorityClass(GetCurrentProcess(), PROCESS\_PRIORITY);*

*......................*

*/\**

*Встановлюємо пріоритет потоків:*

*Стандартні значення констант пріоритету потоків:";*

*THREAD\_PRIORITY\_TT.ME\_CRITT.CAL = 15*

*THREAD\_PRIORITY\_HIGHEST = 2*

*THREAD\_PRIORITY\_ABOVE\_NORMAL = 1*

*THREAD\_PRIORITY\_NORMAL = 0*

*THREAD\_PRIORITY\_BELOW\_NORMAL = -1*

*THREAD\_PRIORITY\_LOWEST = -2*

*THREAD\_PRIORITY\_IDLE = -15 \*/*

*for(int i = 0; i < N; i++)*

*//не парним - серелнiй пріоритет, парним - низький*

*SetThreadPriority(hThread[i], i%2 ? THREAD\_PRIORITY\_NORMAL : THREAD\_PRIORITY\_LOWEST);*

*......................*

*return 0;*

*}*

1. Кінцевий текст програми міститься в розділі 3, а результати виконання див. розділ 4.

## Текст програми

#include <iostream>

#include <stdio.h>

#include <conio.h>

#include <math.h>

#include <windows.h>

using namespace std;

//==============================================================

//глобальні константи

const int N = 8; //к-сть потоків

//масив тривалості роботи потоків

double ThreadTime[N];

//пріоритет з яким виконується процес

const DWORD PROCESS\_PRIORITY = HIGH\_PRIORITY\_CLASS;

/\* Варіанти пріоритетів процесу:

REALTIME\_PRIORITY\_CLASS - реальний час

HIGH\_PRIORITY\_CLASS - високий

NORMAL\_PRIORITY\_CLASS - середнiй

IDLE\_PRIORITY\_CLASS - низький \*/

//==============================================================

//клас для розрахунку часу роботи потоків

class Timing2 {

public:

//запуск таймера

void StartTiming();

//стоп таймера

void StopTiming();

//отримання значення таймера

double GetUserSeconds() const {

return value;

}

private:

DWORD\_PTR oldmask;

double value;

LARGE\_INTEGER time1;

};

void Timing2::StartTiming() {

//команда компілятору -

//не оптимізовувати код отримання значення даної змінної

volatile int warmingUp = 1;

//виконуємо паралельно в декількох потоках

#pragma omp parallel for

/\*

Даний цикл виводить процесор на максимальну швидкість роботи і тим самим трохи

підвищує точність вимірювання швидкості.

При цьому ми "прогріваємо" всі доступні ядра. \*/

for (int i=1; i<10000000; i++) {

#pragma omp atomic

warmingUp \*= i;

}

//виконання поточного потоку на першому процесорі

oldmask = SetThreadAffinityMask(::GetCurrentThread(), 1);

//засікаємо к-сть імпульсів процесора

//до початку роботи таймера

QueryPerformanceCounter(&time1);

}

void Timing2::StopTiming() {

LARGE\_INTEGER performance\_frequency, time2;

//тактова частота

QueryPerformanceFrequency(&performance\_frequency);

//засікаємо к-сть імпульсів процесора

//при завершенні роботи таймера

QueryPerformanceCounter(&time2);

//виконуємо потік на першому процесорі

SetThreadAffinityMask(::GetCurrentThread(), oldmask);

//різниця в тактах між завершенням та початком

value = (double)(time2.QuadPart - time1.QuadPart);

//поділимо на к-сть тактів в секуду -

//отримаємо значення в секундах

value /= performance\_frequency.QuadPart;

}

//==============================================================

//функція табулювання

void tabulation(int threadNum) { //threadNum - номер потоку

const double a = -1, b = 1.0; //(a;b) - проміжок табулювання

const int n = 100; //кількість точок табулювання

const double eps = 0.0001; //точність обчислення

int i;

double x, s, si, y;

Timing2 t;

//засікаєм час початку роботи потоку

t.StartTiming();

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

//отримуємо наступне значення x на проміжку табуляції

x = a+j\*(b-a)/(double) n;

//перший елемент ряду

si = x; s = si;

i = 2;

//виконуємо, поки не досягнуто бажаної точності

while(fabs(si)>eps) {

//обчислимо i-ий елемент ряду Тейлора

si \*= -x\*(i-1)/i;

//збільшимо суму ряду

s += si;

//збільшимо лічильник

i++;

}

//отримаємо значення ф-ї штатними методами

y=log(1+x);

printf("Потiк %d Крок %d: x=%f y=%f s=%f\n", threadNum+1, j, x, y, s);

}

//засікаєм час завершення потоку

t.StopTiming();

//...і зберігаємо його в глобальний масив

ThreadTime[threadNum] = t.GetUserSeconds();

}

//==============================================================

//функція опрацювання 1-го потоку

DWORD WINAPI Output0(LPVOID Param) {

tabulation(0);

return 0;

}

//==============================================================

//функція опрацювання 2-го потоку

DWORD WINAPI Output1(LPVOID Param) {

tabulation(1);

return 0;

}

//==============================================================

//функція опрацювання 3-го потоку

DWORD WINAPI Output2(LPVOID Param) {

tabulation(2);

return 0;

}

//==============================================================

//функція опрацювання 4-го потоку

DWORD WINAPI Output3(LPVOID Param) {

tabulation(3);

return 0;

}

//==============================================================

//функція опрацювання 5-го потоку

DWORD WINAPI Output4(LPVOID Param) {

tabulation(4);

return 0;

}

//==============================================================

//функція опрацювання 6-го потоку

DWORD WINAPI Output5(LPVOID Param) {

tabulation(5);

return 0;

}

//==============================================================

//функція опрацювання 7-го потоку

DWORD WINAPI Output6(LPVOID Param) {

tabulation(6);

return 0;

}

//==============================================================

//функція опрацювання 8-го потоку

DWORD WINAPI Output7(LPVOID Param) {

tabulation(7);

return 0;

}

//==============================================================

int main() {

setlocale(LC\_ALL,"Ukrainian");

HANDLE hThread[N]; //масив дескрипторів потоків

DWORD ThreadId[N]; //унікальні ідентифікатори потоків

//встановлюємо пріотитет процесу

SetPriorityClass(GetCurrentProcess(), PROCESS\_PRIORITY);

Timing2 t;

//засікаємо час початку процесу

t.StartTiming();

//створюємо потоки

hThread[0] = CreateThread(NULL, 0, Output0, NULL, 0, &ThreadId[0]);

if(N > 1) hThread[1] = CreateThread(NULL, 0, Output1, NULL, CREATE\_SUSPENDED, &ThreadId[1]);

if(N > 2) hThread[2] = CreateThread(NULL, 0, Output2, NULL, CREATE\_SUSPENDED, &ThreadId[2]);

if(N > 3) hThread[3] = CreateThread(NULL, 0, Output3, NULL, CREATE\_SUSPENDED, &ThreadId[3]);

if(N > 4) hThread[4] = CreateThread(NULL, 0, Output4, NULL, CREATE\_SUSPENDED, &ThreadId[4]);

if(N > 5) hThread[5] = CreateThread(NULL, 0, Output5, NULL, CREATE\_SUSPENDED, &ThreadId[5]);

if(N > 6) hThread[6] = CreateThread(NULL, 0, Output6, NULL, CREATE\_SUSPENDED, &ThreadId[6]);

if(N > 7) hThread[7] = CreateThread(NULL, 0, Output7, NULL, CREATE\_SUSPENDED, &ThreadId[7]);

/\*

Встановлюємо пріоритет потоків:

Стандартні значення констант пріоритету потоків:";

THREAD\_PRIORITY\_TT.ME\_CRITT.CAL = 15

THREAD\_PRIORITY\_HIGHEST = 2

THREAD\_PRIORITY\_ABOVE\_NORMAL = 1

THREAD\_PRIORITY\_NORMAL = 0

THREAD\_PRIORITY\_BELOW\_NORMAL = -1

THREAD\_PRIORITY\_LOWEST = -2

THREAD\_PRIORITY\_IDLE = -15 \*/

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

//не парним - серелнiй пріоритет, парним - низький

SetThreadPriority(hThread[i], i%2 ? THREAD\_PRIORITY\_NORMAL : THREAD\_PRIORITY\_LOWEST);

//запуск потоків

for(int i = 0; i < N; i++) ResumeThread(hThread[i]);

//після натиску на будь-яку клавішу -

//призупиняємо роботу всіх потоків

\_getch();

for(int i = 0; i < N; i++) SuspendThread(hThread[i]);

//продовжуємо роботу всіх потоків

//після наступного натиску на клавіатурі

\_getch();

for(int i = 0; i < N; i++) ResumeThread(hThread[i]);

//чекаємо на завершення всіх потоків

WaitForMultipleObjects(N, hThread, TRUE,INFINITE);

//засікаємо час завершення роботи процесу

t.StopTiming();

printf("\n");

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

printf("Час роботи потоку %d: %.3G секунд. Прiоритет потоку: %d\n", i+1, ThreadTime[i], GetThreadPriority(hThread[i]));

printf("(значення прiоритетiв потоку: HIGHEST = 2, NORMAL = 0, LOWEST = -2)\n");

printf("\nЗагальний час роботи процесу: %.3G секунд. Прiоритет процесу: %d\n", t.GetUserSeconds(), GetPriorityClass(GetCurrentProcess()));

printf("(значення прiоритетiв процесу: HIGH = %d, NORMAL = %d, IDLE = %d)\n", HIGH\_PRIORITY\_CLASS, NORMAL\_PRIORITY\_CLASS, IDLE\_PRIORITY\_CLASS);

printf("\nДля завершення роботи програми - натиснiть на будь-яку клавiшу...\n");

//завершуємо всі потоки

DWORD ExitCode[N];

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

//отримуємо код завершення потоків

GetExitCodeThread(hThread[i], &ExitCode[i]);

//завершення потоку

TerminateThread(hThread[i], ExitCode[i]);

//закриття дексриптора потоку

CloseHandle(hThread[i]);

}

//після натиску на будь-яку клавішу - вихід з програми

\_getch();

return 0;

}

## Результат виконання програми

Запустимо програму на виконання з ***одним потоком*** та ***високим пріоритетом виконанання процесу*** (рис. 4.1). Також запустимо диспетчер задач і подивимось на кількість потоків під час виконання алгоритму (рис. 4.2), на кількість потоків після завершення потоку, в якому виконується алгоритм і закриття цього потоку (рис. 4.3), а також на пріоритет з яким виконується процес (рис. 4.4).

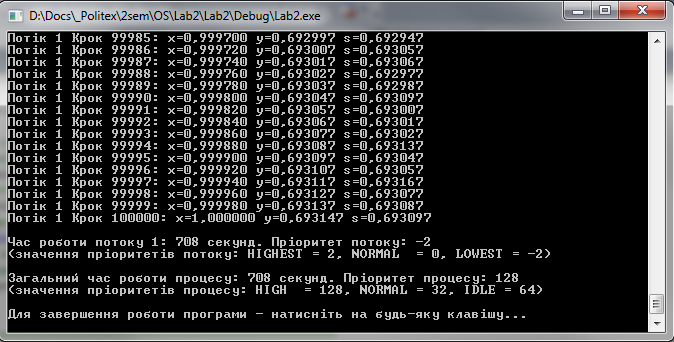


Рис. 4.1. Один потік / високий пріоритет процесу

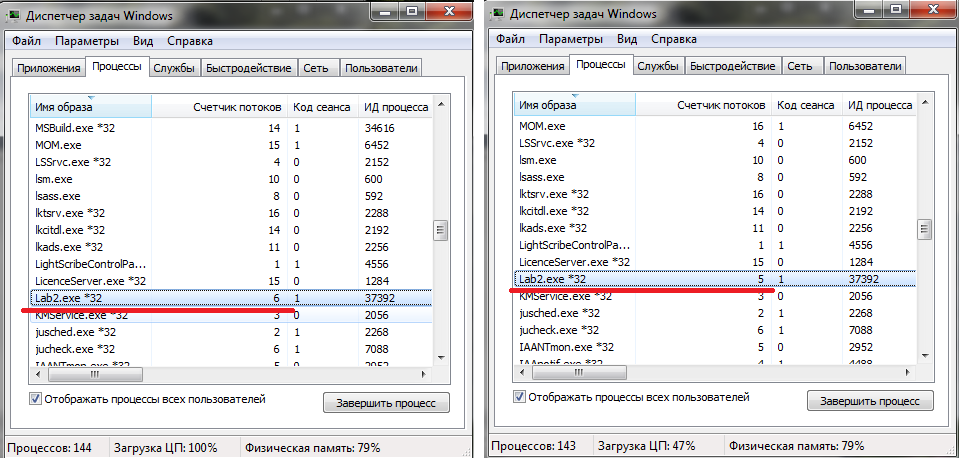


Рис. 4.2. Один потік / під час виконання Рис. 4.3. Один потік / після завершення

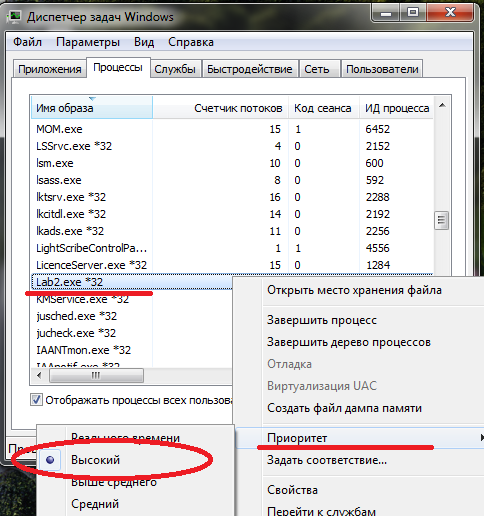


Рис. 4.4. Пріоритет процесу – «високий»

Як бачимо, час виконання процесу, як і єдиного потоку, становить 708 секунд. Пріоритет потоку = 128 (що відповідає значенню «Високий»), пріоритет потоку = -2 (нижчий за пріоритет процесу). Звернемо увагу, що в диспетчері задач під час виконання програми, кількість потоків змінювалась, але найдовше трималося значення – 6, після закриття потоку – 5. З цього можна зробити висновок, що 5-ть потоків є службовими, а 1 – користувацький, створений нами.

Збільшимо ***кількість потоків до двох*** та запустимо програму (рис. 4.5). Запустимо диспетчер задач (рис. 4.6 - 4.7).

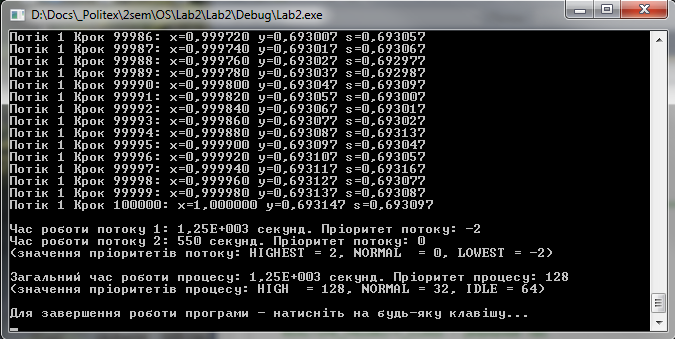


Рис. 4.5. Два потоки

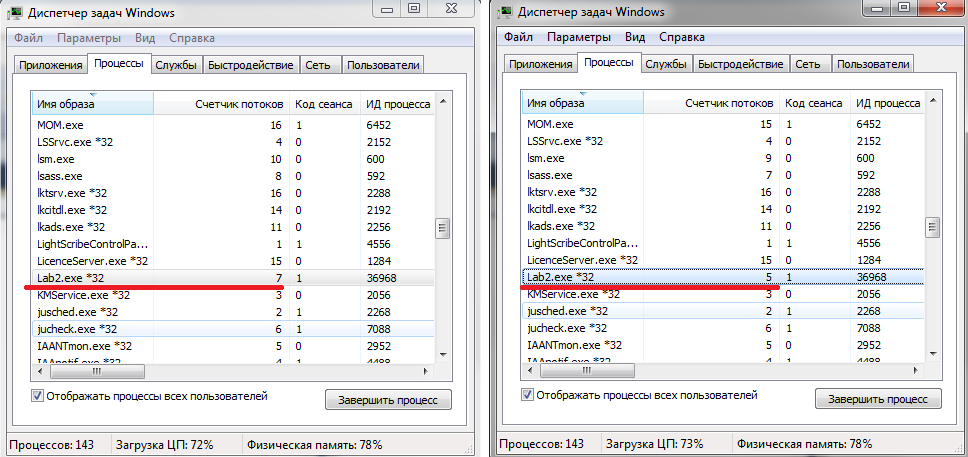


Рис. 4.6. Два потоки / під час виконання Рис. 4.7. Два потоки / після завершення

Як бачимо, час виконання процесу, як першого потоку, становить 1,25×103 = 1250 секунд, що є більшим за час виконання процесу в попередньому випадку. Другий потік виконався швидше (за 550 секунд), оскільки має вищий пріоритет. В диспетчері задач під час виконання програми, кількість потоків – 7, після закриття потоку – 5 (два користувацькі).

Збільшимо ***кількість потоків до чотирьох*** (рис. 4.8 – 4.10).

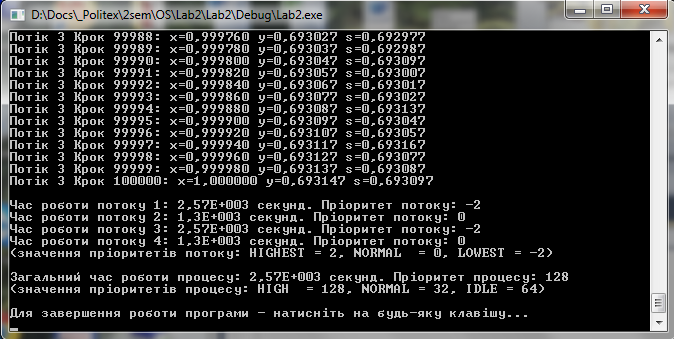


Рис. 4.8. Чотири потоки

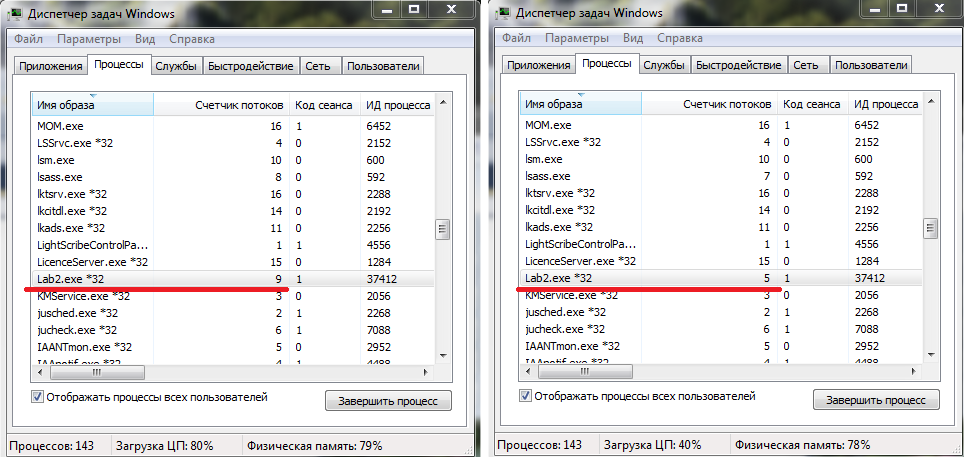


Рис. 4.9. Чотири потоки / під час Рис. 4.10. Чотири потоки / після

Як бачимо, час виконання процесу, як 1-го та 3-го потоку, становить 2,57×103 секунд, що є більшим за час виконання процесу в попередньому випадку. 2-ий та 4-ий потоки виконалися швидше (за 1,3×103 секунд), оскільки має вищий пріоритет. В диспетчері задач під час виконання програми, кількість потоків – 9, після закриття потоку – 5 (чотири користувацькі).

Збільшимо ***кількість потоків до шістьох*** (рис. 4.11 – 4.13).

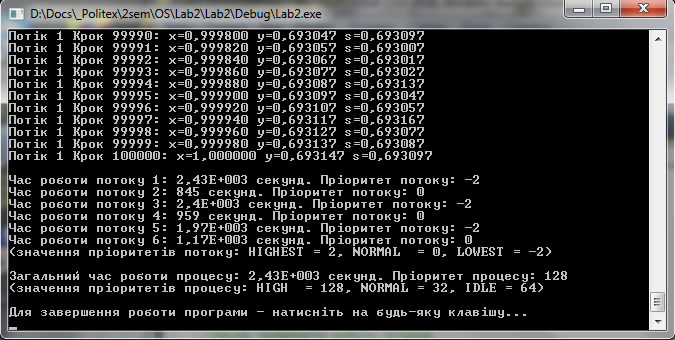


Рис. 4.11. Шість потоків

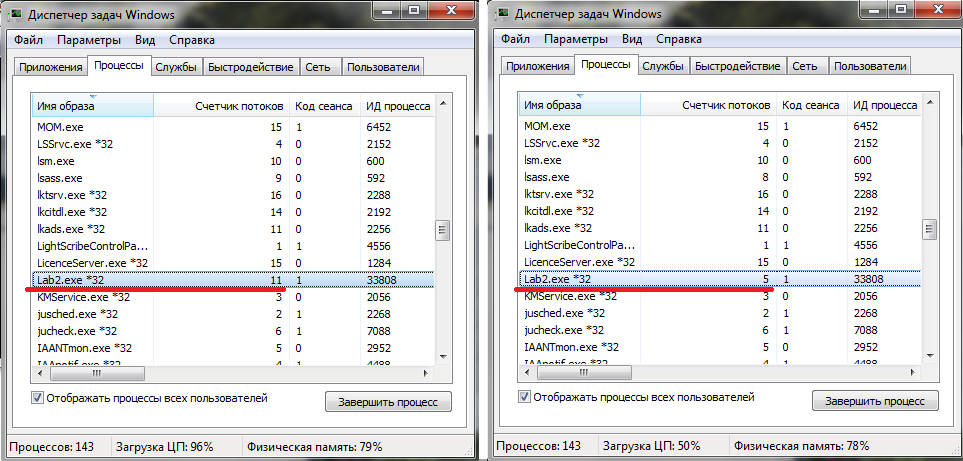


Рис. 4.12. Шість потоків / під час Рис. 4.13. Шість потоків / після

Як бачимо, час виконання процесу, як 1-го потоку, становить 2,43×103 секунд, що є меншим за час виконання процесу в попередньому випадку. Причиною цього, можливо є, те що комп’ютер мав менше завантаження всіма процесами ОС, ніж в попередні рази. Всі парні потоки виконалися швидше, оскільки мають вищий пріоритет. В диспетчері задач під час виконання програми, кількість потоків – 11, після закриття потоку – 5 (шість користувацьких).

Збільшимо ***кількість потоків до вісьмох*** (рис. 4.14 – 4.16).

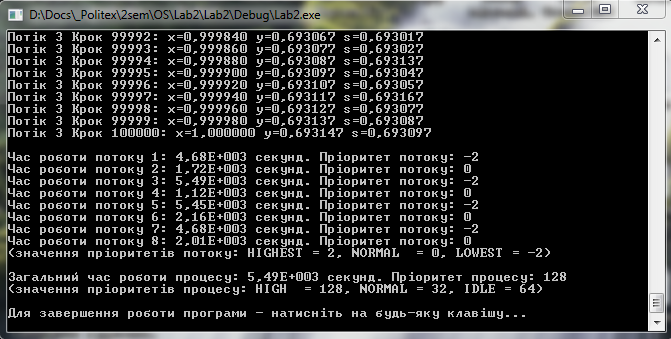


Рис. 4.14. Вісім потоків

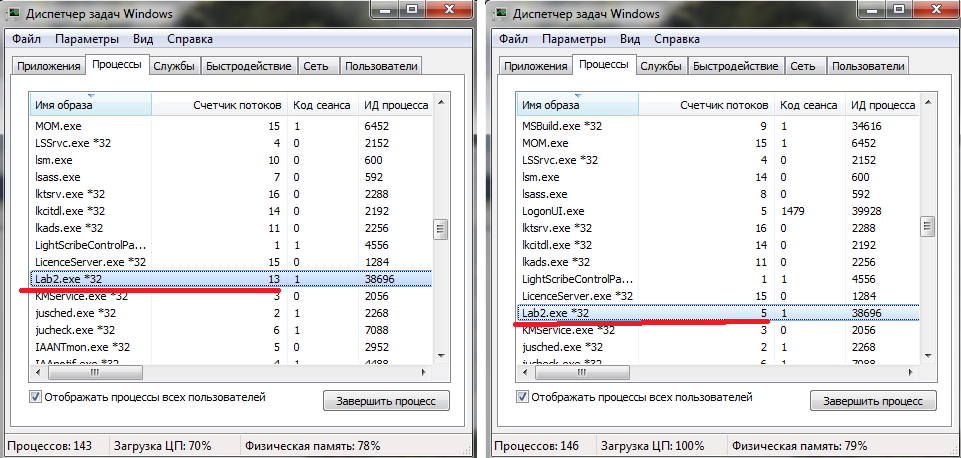


Рис. 4.15. Вісім потоків / під час Рис. 4.16. Вісім потоків / після

Як бачимо, час виконання процесу, як 3-го потоку, становить 5,49×103 секунд, що є більшим за час виконання у всіх попередніх випадках.. Всі парні потоки виконалися швидше, оскільки мають вищий пріоритет. В диспетчері задач під час виконання програми, кількість потоків – 13, після закриття потоку – 5 (вісім користувацьких).

Запустимо програму на виконання з ***одним потоком*** та з ***пріоритетом виконанання процесу в реальному часі*** (рис. 4.17). Також запустимо диспетчер задач і перевіримо чи процес має дійсно даний пріоритет (рис. 4.18).

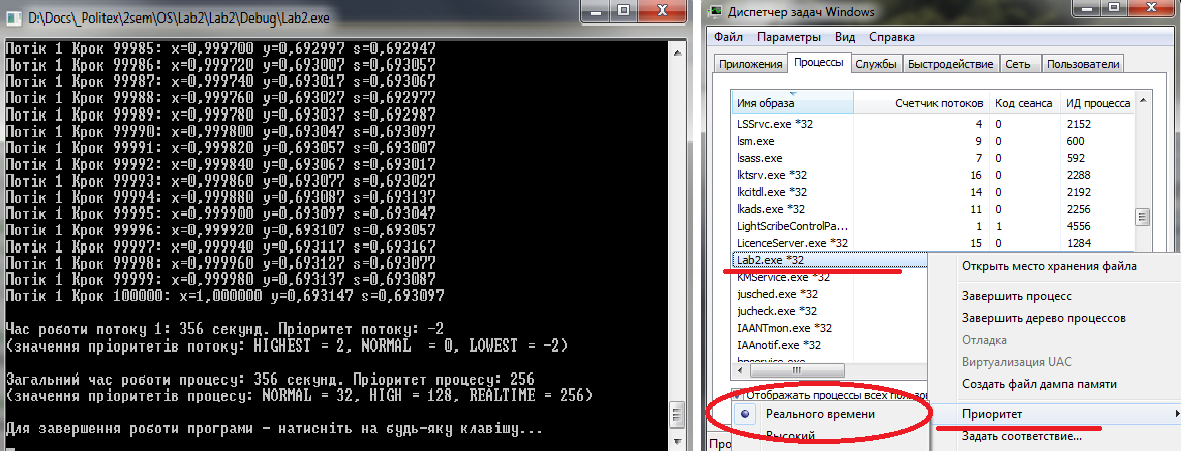


Рис. 4.17. Один потік / Пріоритет – реальний час Рис. 4.18. Вигляд в диспетчері задач

Як бачимо, час виконання процесу, становить 356 секунд, що є швидшим (практично в два рази), ніж при запуску програми з з високим пріоритетом процесу (708 секунд), та при інших рівних умовах.

## ВИСНОВКИ

На даній лабораторній роботі я ознайомитися з багатопоточністю в ОС Windows. Навчився працювати з потоками, використовуючи WinAPI-функції, зокрема запускати декілька потоків, призупиняти, відновлювати їхню роботу та закривати їх, також змінювати пріоритети виконання потоків та процесів, розраховувати час роботи в потоці та в процесі.