**МІНІСТЕРСТВО ОСВІТИ І НАУКИ  УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"**

Інститут **ІКНІ**

Кафедра **ПЗ**

**ЗВІТ**

до лабораторної роботи № 1

**На тему:** *“ Ознайомлення та керування процесами в операційних системах для персонального комп’ютера. 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:** Ознайомлення та керування процесами в операційних системах для персонального комп’ютера. Windows.

**Мета роботи:** Ознайомитися з процесами та потоками в ОС Windows. Навчитися працювати із системними утилітами, що дають можливість отримувати інформацію про процеси, потоки, використовувану ними пам’ять, та іншу необхідну інформацію.

**Індивідуальне завдання**

**Варіант 1**

1. За допомогою утиліти «Диспетчер задач» та Process Explorer отримати повну інформацію про процеси: ідентифікатор процесу, завантаження ЦП (центрального процесора), час ЦП, базовий пріоритет, стан процесу, пам'ять-використання, пам'ять-зміни, пам'ять-максимум, помилок сторінки, об’єкти USER, код сеансу, об’єм віртуальної пам’яті, лічильник дескрипторів, лічильник потоків.

2. За допомогою утиліти Process Explorer отримати додаткову інформацію про процеси та їхні потоки.

3. Використовуючи «Диспетчер задач» та Process Explorer змінити пріоритет будь-якого процесу, від низького до «реального часу»; задати відповідність виконання процесів на окремих ядрах центрального процесора; виконати завершення процесу.

4. Використовуючи Process Explorer призупинити процес і відновити його роботу.

5. Скомпілювати файл main.cpp представлений нижче і запустити виконуваний файл на різній кількості активних процесорів (ядер). Знайти для даної програми величини A , S , p при різних вхідних значеннях величини n .

#include <iostream>

#include <thread>

#include <mutex>

#include <vector>

#include <algorithm>

#include <cmath>

#define DEF\_THREAD\_COUNT 4

const int N = 50000000;

namespace MCConcurrency

{

inline int threadsCount(int iterationsCount = -1)

{

int numThreads = static\_cast<int>(std::thread::hardware\_concurrency());

if (numThreads == 0)

numThreads = DEF\_THREAD\_COUNT;

if (iterationsCount == 0)

return 1;

if (iterationsCount > -1)

numThreads = std::min(static\_cast<int>(numThreads), iterationsCount);

return numThreads;

}

template<typename Function>

void parallel\_for(int first, int last, int threadsNumbers, const Function& l)

{

int numThreads = threadsNumbers;

int elementsCount = last - first;

//auto numThreads = threadsCount(last - first);

if (numThreads <= 0)

return;

int threadNumber = 0;

std::mutex tasksMutex;

std::vector<std::thread> threads;

const auto chunk = std::max(static\_cast<int>(std::ceil(elementsCount /

static\_cast<double>(numThreads))), 1);

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

{

threads.push\_back(std::thread([&]()

{

int thread\_number;

tasksMutex.lock();

thread\_number = threadNumber;

++threadNumber;

tasksMutex.unlock();

int beg = first + thread\_number \* chunk;

int end = std::min(first + (thread\_number + 1) \* chunk, elementsCount);

for (int ind = beg; ind < end; ind++)

{

l(ind, thread\_number);

}

}));

}

std::for\_each(begin(threads), end(threads), [](std::thread& th)

{

th.join();

});

}

}

int main()

{

typedef std::chrono::high\_resolution\_clock Time;

typedef std::chrono::milliseconds ms;

typedef std::chrono::duration<float> fsec;

std::vector<float> arr(N);

auto numThreads = MCConcurrency::threadsCount(N);

std::cout << "Set process af inity (cores count) and press <Enter>";

getchar();

auto t0 = Time::now();

MCConcurrency::parallel\_for(0, N, numThreads, [&](int nIndex, int threadNumber) {

arr[nIndex] = sin(threadNumber) \* cos(threadNumber);

});

auto t1 = Time::now();

fsec fs = t1 - t0;

ms d = std::chrono::duration\_cast<ms>(fs);

std::cout << "Duration: " << d.count() << "ms\n";

return 0;

}

6. Дослідити вплив зміни відповідності ядру на швидкодію процесу. Виконати завдання згідно варіанту, що відповідає порядковому номеру у списку підгрупи.

1. Пошук файлів

**Теоретичні відомості**

Операційна система - це набір програм, які підтримують роботу програм користувачів, забезпечують їх взаємодію з апаратними засобами та здійснюють загальне керування роботою ЕОМ.

Процес — одне з найважливіших понять у архітектурі [операційних систем](https://uk.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) та [програмуванні](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F). Процес — об'єкт операційної системи, контейнер системних ресурсів, призначених для підтримки виконання [програми](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B0). Коли в середовищі [операційної системи](https://uk.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) запускається [прикладна програма](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B0), система створює спеціальний об’єкт – процес, – який призначений для підтримки її виконання. Хоча може здатися, що програма й процес – поняття схожі, вони фундаментально відрізняються. Програма – це статичний набір команд, а процес – контейнер для ресурсів, які використовуються при виконанні екземпляра програми. Системна реалізація та функції процесу в різних [операційних системах](https://uk.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) дещо відмінні.

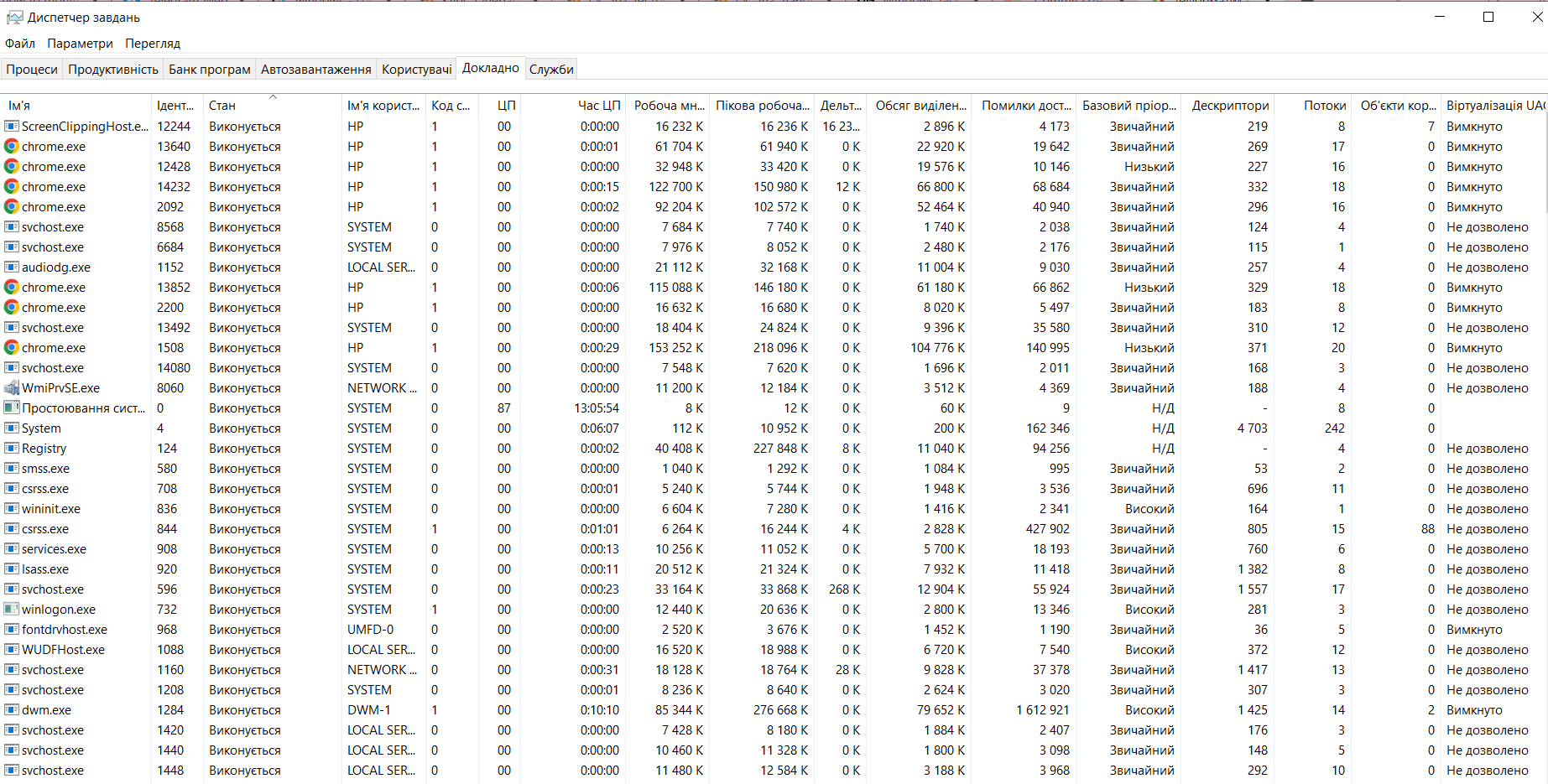
На найвищому рівні абстракції процес у [Windows](https://uk.wikipedia.org/wiki/Windows) містить:

* закритий [віртуальний адресний простір](https://uk.wikipedia.org/wiki/%D0%92%D1%96%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0_%D0%BF%D0%B0%D0%BC%27%D1%8F%D1%82%D1%8C) – діапазон адрес [віртуальної пам'яті](https://uk.wikipedia.org/wiki/%D0%92%D1%96%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0_%D0%BF%D0%B0%D0%BC%27%D1%8F%D1%82%D1%8C), яким може користуватися процес;
* програму, що виконується – початковий код і спроектовані на віртуальний адресний простір процесу дані;
* список відкритих дескрипторів (handles) різних системних ресурсів –комунікаційних портів, файлів та інших об'єктів, доступних усім потокам даного процесу;
* контекст захисту (security context), який називають маркером доступу (access token), – він ідентифікує користувача та групи безпеки й привілеї, зіставлені з процесом;
* унікальний ідентифікатор процесу;
* мінімум один [потік](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D1%96%D0%BA_(Windows)).

Найважливішою складовою частиною процесу є [потік](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D1%96%D0%BA_(Windows)). Процес як системний об'єкт призначений для "обслуговування" потоків. Сам по собі процес не виконує ніяких дій, для цього призначений [потік](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D1%96%D0%BA_(Windows)). Крім закритого адресного простору й одного або кількох потоків, у кожного процесу є ідентифікатор захисту і список відкритих дескрипторів об'єктів (файли, розділи загальної пам'яті). Кожен процес має контекст захисту, який зберігається в об'єкті – [маркері доступу](https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%80%D0%BA%D0%B5%D1%80_%D0%B4%D0%BE%D1%81%D1%82%D1%83%D0%BF%D1%83). [Маркер доступу](https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%80%D0%BA%D0%B5%D1%80_%D0%B4%D0%BE%D1%81%D1%82%D1%83%D0%BF%D1%83) містить ідентифікацію захисту й визначає повноваження даного процесу. Дескриптори віртуальних адрес (virtual address descriptors, VAD) – це структури даних, які використовуються диспетчером пам'яті для обліку віртуальних адрес, задіяних процесом.

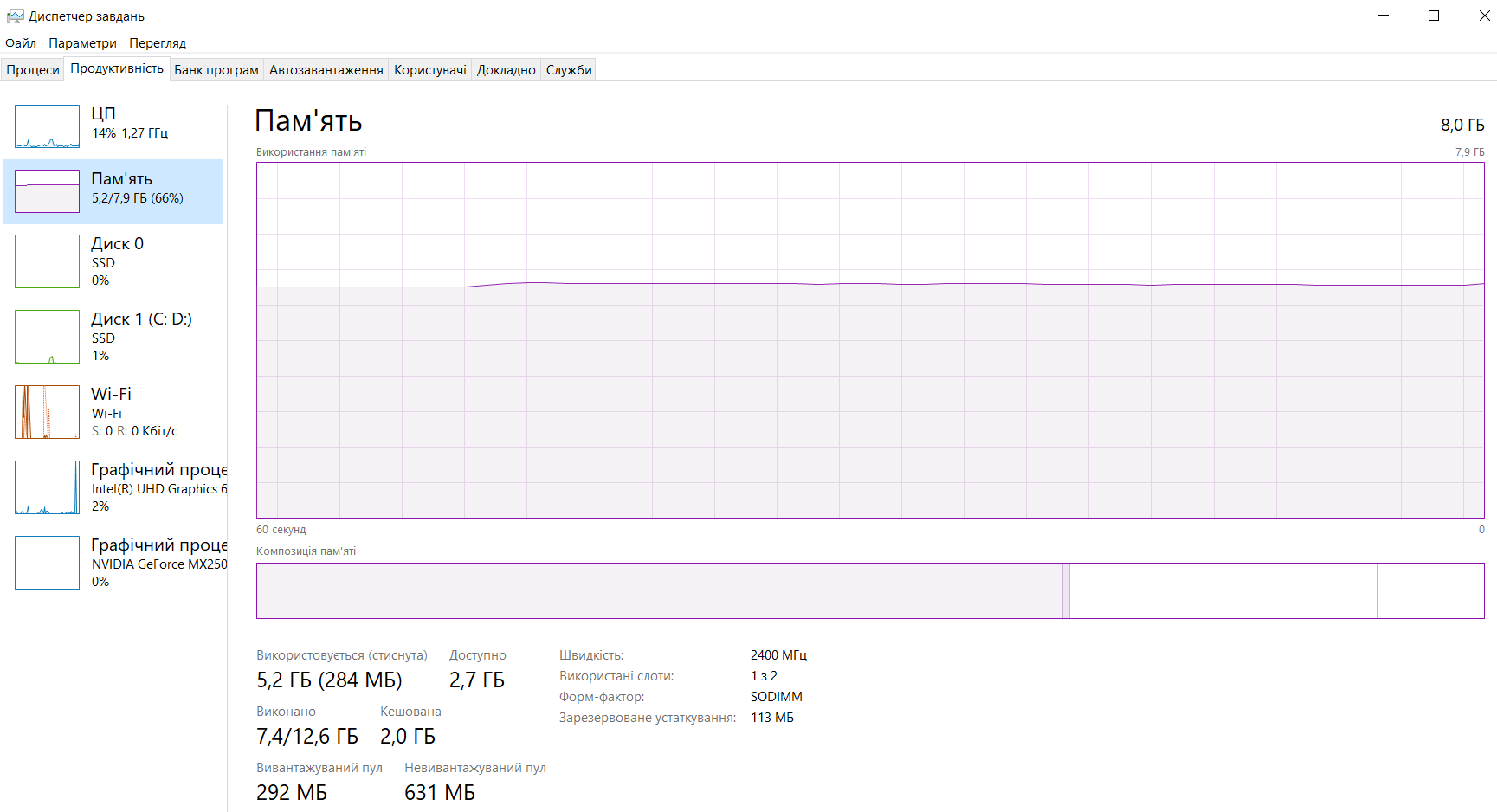
**Хід роботи**

1. За допомогою Task Manager, зокрема вкладки “Докладно” та Process Explorer я отримала повну інформацію про процеси у своїй операційній системі.

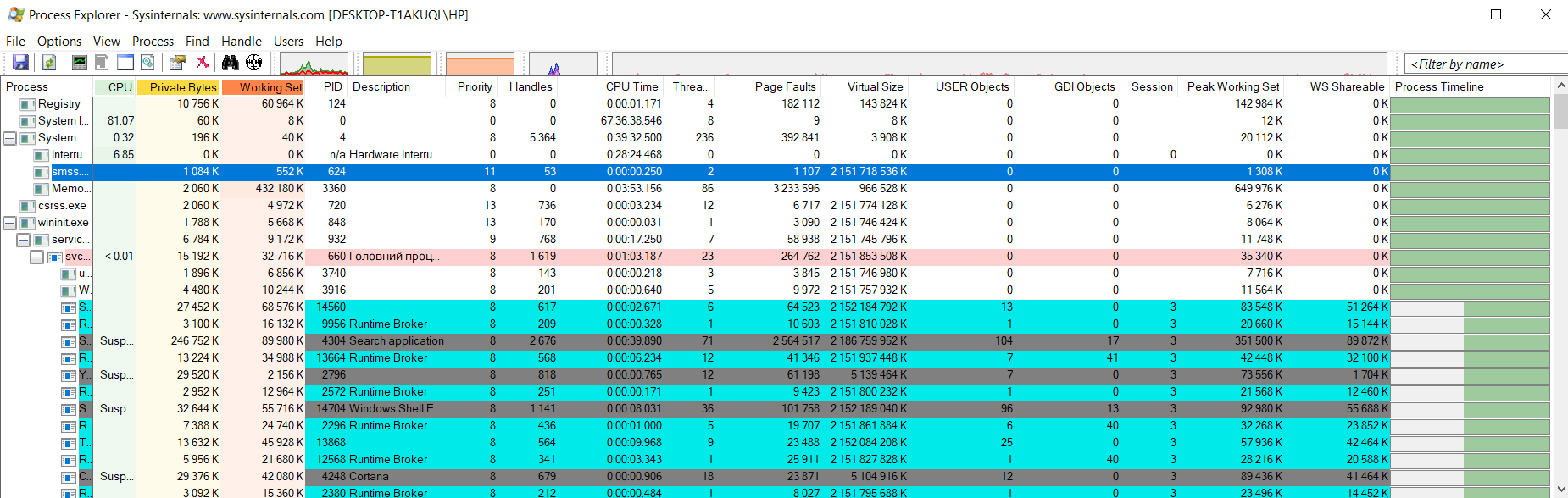


*Рис. 1 Процеси та їх властивості у Task Manager*

Обсяг віртуальної пам’яті доступний у такому вигляді:

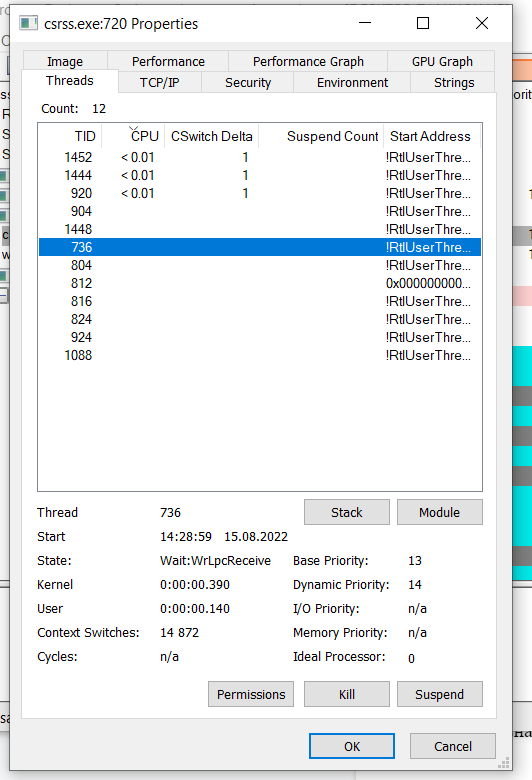


*Рис. 2 Обсяг віртуальної пам’яті у Task Manager*



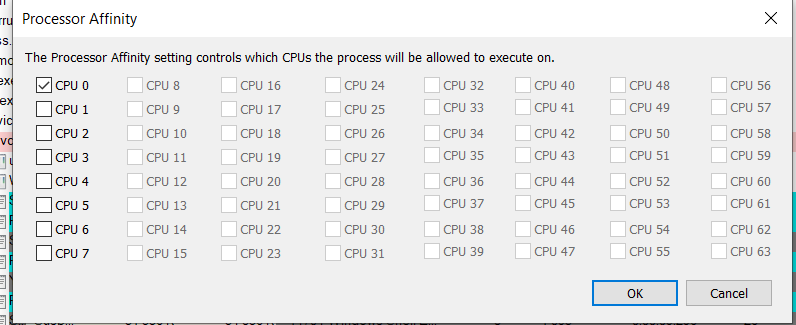
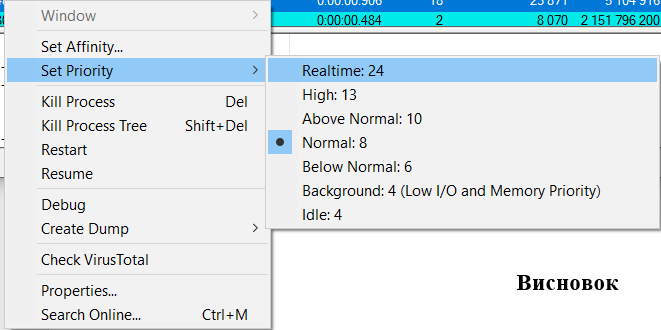
*Рис. 3 Процеси та їх властивості у Process Explorer*

1. За допомогою Process Explorer я отримала додаткову інформацію про потоки процесів, зокрема: Thread ID, час початку роботи потоку та стартову адресу тощо.

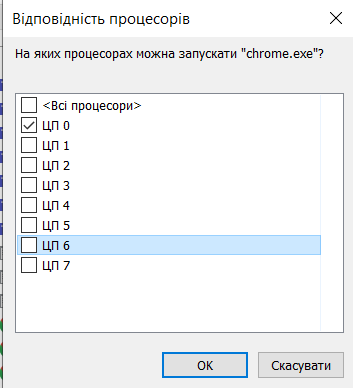
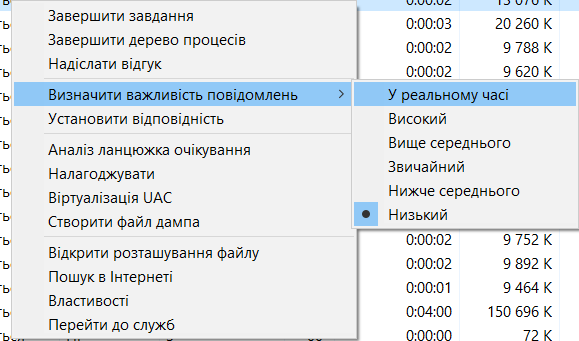


*Рис. 4 Інформація про потоки у Process Explorer*

1. На цьому етапі я змінювала пріоритет процесів від низького до реального часу, задала відповідність виконання процесів на окремих ядрах центрального процесора і виконала завершення процесу.

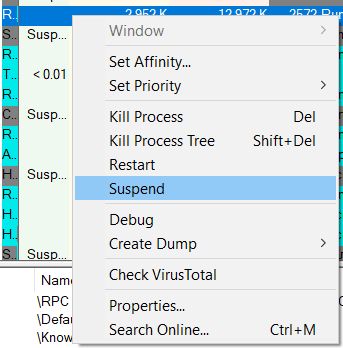
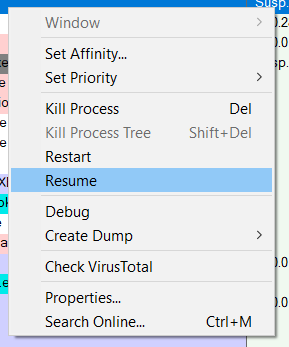


*Рис. 5 Операції над процесами у Process Explorer*



*Рис. 6 Операції над процесами у Task Manager*

4. На цьому кроці я призупиняла та відновлювала роботу процесу у Process Explorer.

*Рис. 7 Операції над процесами у Task Manager*

1. Я компілювала файл тричі разів, застосовуючи 6 різних значень n і запускала виконуваний вайл на n активних процесорах.

При запуску з CPU 0, T1 = 1974 ms. Для розрахунків використовувала ці формули:

A = T1/Tn; A = S, p + (1-p)/n = 1/A => p = (n/A - 1) / (n-1);

Спочатку запускала файл на CPU 1, CPU 2, CPU 3. При n = 3, T3 = 910 ms , A = 2.17, p =0.19;

Запуск на CPU 1, CPU 2, CPU 3, CPU 4. При n = 4, T = 692 ms, A = 2.85, p = 0.14;

Запуск на CPU 1, CPU 2, CPU 3, CPU 4, CPU 5. При n = 5, T = 600 ms, A = 3.29 , p = 0.104;

Запуск на CPU 1, CPU 2, CPU 3, CPU 4, CPU 5, CPU 6. При n = 6, T = 591 ms, A = 3.34, p = 0.1588;

Коли відбувається на ядрах від CPU 0 до CPU 7, n = 8, T =192 ms, A = 10.2, p = -0.03;

1. Для виконання цього завдання я відкрила Провідник файлів та Process Explorer.

Налаштувала моніторинг процесу explorer.exe у режимі користувача та слідкувала за часом виконання пошуку файлу у папці. Досліджуючи залежність між відповідністю ядру і швидкодією, я отримала такі значення часу, що відповідають вказаним процесорам.

CPU 0 – 453 мс

CPU 1 – 187 мс

CPU 2 – 219 мс

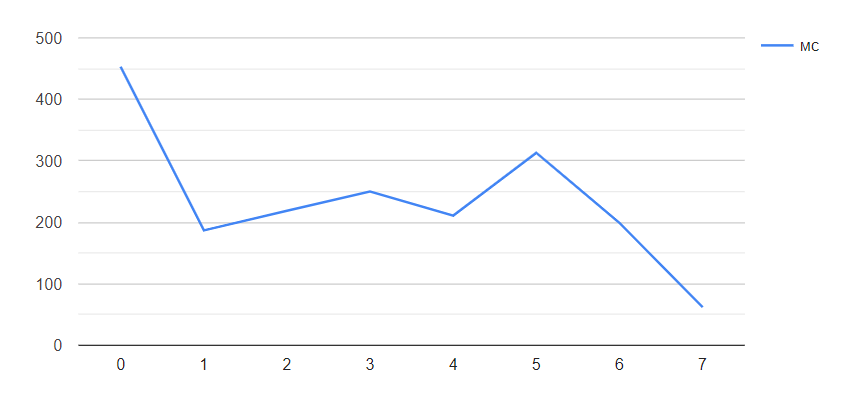
CPU 3 – 250 мс

CPU 4 – 211 мс

CPU 5 – 313 мс

CPU 6 – 199 мс

CPU 7 – 62 мс



*Рис. 8 Відношення між відповідністю ядру та CPU Time*

1. Результат моніторингу процесів, проведений у обидвох утилітах, поданий у висновку.

**Висновок**

На цій лабораторній роботі я глибше познайомилася з процесами, які відбуваються у моїй ОС, за допомогою Диспетчера Задач та Process Explorer. Провела моніторинг процесів та операції над цими процесами, зокрема зміна пріоритету, зупинка, відновлення тощо. Отримувала широкий спектр даних про процеси за допомогою раніше згаданих утиліт. Внаслідок цього моніторингу я побачила, що Process Explorer надає набагато більше характеристик процесу, так само дає мені як користувачу значно більший функціонал як з потоками, так і процесами. Task Manager також має багато переваг в залежності від того, скільки інформації про процеси потрібно користувачу. У Task Manager легко доступитись до віртуальної пам’яті, графіків, що стосуються роботи CPU в реальному часі.

Результати дослідження відношення між відповідністю ядру та CPU Time:найбільшу швидкодію було помічено при роботі CPU 7, найменшу - CPU 0.