Лабораторна робота № 2

Тема: Організація взаємодії між процесами та потоками.

Мета: Навчитися планувати процеси і потоки в середовищі операційної системи, розробляти програми планування процесів і потоків.

Обладнання та програмне забезпечення: ПК, операційна система Windows, середовище розробки Visual Studio.

Вказівки для самостійної підготовки

Під час підготовки необхідно повторити теоретичний матеріал:

1. Алгоритми планування.
2. Реалізація планування у Windows.
3. Функції Win API для роботи пріоритетами процесів і потоків.

Теоретичні відомості

1 Алгоритми планування

Виконання потоку – це цикл чергування періодів обчислень і періодів очікування введення-виведення. Інтервал часу, коли потік виконує інструкції процесора, називають *інтервалом використання процесора (CPU burst),* інтервал часу очікування введення-виведення — *інтервалом введення-виведення (I/O burst).*

1.1 Планування за принципом FIFO

Чергу готових потоків організовують за принципом FIFO. Коли в системі створюється новий потік, його керуючий блок додається у хвіст черги. Коли процесор звільняється, його надають потоку з голови черги.

1.2 Кругове планування

Кожному потокові виділяють інтервал часу *(квант часу)* і протягом якого цьому потокові дозволено виконуватися. Коли потік усе ще виконується після вичерпання кванта часу, його переривають і перемикають процесор на виконання інструкцій іншого потоку. Коли він блокується або закінчує своє виконання до вичерпання кванта часу, процесор теж передають іншому потокові. Довжина кванта часу для всієї системи однакова.

1.3 Планування із пріоритетами

Кожному потокові надають пріоритет, на виконання ставиться потік із найвищим пріоритетом із черги готових потоків. Пріоритети можуть надаватися потокам статично або динамічно.

Розподіл пріоритетів може призвести до того, що потоки процесів із низьким пріоритетом чекатимуть дуже довго. Таку ситуацію називають ***голодуванням***.

1.4 Планування на підставі характеристик подальшого виконання

Рішення про вибір потоку для виконання приймають на підставі знання або оцінки характеристик подальшого його виконання.

Алгоритм ***«перший - із найкоротшим часом виконання» (STCF).*** З кожним потоком пов'язують тривалість наступного інтервалу використання ним процесора і для виконання щоразу вибирають той потік, у якого цей інтервал найкоротший. У результаті потоки, що захоплюють процесор на коротший час, отримують під час планування перевагу і швидше виходять із системи.

Алгоритм «перший — із найкоротшим часом виконання, що залишився» (SRTCF). Його відмінність від SCTF полягає в тому, що, коли в чергу готових потоків додають новий, у якого наступний інтервал використання процесора коротший, ніж час, що залишився до завершення виконання поточного потоку, поточний потік переривається, і на його місце стає новий потік.

1.5 Багаторівневі черги зі зворотним зв'язком

Є кілька черг готових потоків із різним пріоритетом, потоки черги із нижчим пріоритетом виконуються, тільки коли всі черги верхнього рівня порожні.

Потокам дозволено переходити із черги в чергу. Потоки в одній черзі об'єднуються за довжиною інтервалу використання процесора. Потоки із коротшим інтервалом перебувають у черзі з більшим пріоритетом.

Усередині всіх черг використовують кругове планування (у найнижчій працює FIFO-алгоритм). Якщо потік вичерпав свій квант часу, він переміщається у хвіст черги із нижчим пріоритетом. У результаті потоки з коротшими інтервалами залишаються з високим пріоритетом, а потоки з довшими інтервалами продовжують свій квант часу. Можна автоматично переміщати потоки, які давно не отримували керування, із черги нижнього рівня на рівень вище.

1.6 Лотерейне планування

Ідея лотерейного планування:

1. потік отримує деяку кількість лотерейних квитків, кожен з яких дає право користуватися процесором упродовж часу T;
2. планувальник через проміжок часу T вибирає випадково один лотерейний квиток;
3. потік, «що виграв», дістає керування до наступного розіграшу.

2 Реалізація планування у Windows

2.1 Планування потоків у ядрі

Ядро Windows розв'язує під час планування дві задачі:

1. облік відносних пріоритетів, присвоєних кожному потокові;
2. мінімізацію часу відгуку інтерактивних застосувань.

Базовою одиницею планування є потік. Під час планування ядро не розрізняє потоки різних процесів, воно має справу з пріоритетами потоків, готових до виконання в певний момент часу.

Пріоритети потоків і процесів

Для визначення порядку виконання потоків диспетчер ядра використовує систему пріоритетів. Кожному потокові присвоюють пріоритет, заданий числом у діапазоні від 1 до 31 (що більше число, то вище пріоритет).

Пріоритети реального часу — 16-31; їх резервує система для дій, час виконання яких є критичним чинником. Динамічні пріоритети — 1-15; вони можуть бути присвоєні потокам застосувань користувача.

Спочатку процесу присвоюють ***клас пріоритету***, а потім потокам цього процесу — відносний пріоритет, який відраховують від класу пріоритету процесу (базовий пріоритет). Під час виконання відносний пріоритет може змінюватися.

Класи пріоритету процесів: реального часу (real-time, відповідає пріоритету потоку 24); високий (high, 13); нормальний (normal, 8); не використовуваний (idle, 4).

Відносні пріоритети потоку бувають такі: найвищий (+2 до базового); вище за нормальний (+1 до базового); нормальний (дорівнює базовому); нижче за нормальний (-1 від базового); найнижчий (-2 від базового).

Пошук потоку для виконання

Для виконання новий потік вибирається, коли:

1. минув квант часу для потоку (*алгоритму пошуку готового потоку);*
2. потік перейшов у стан очікування події (потік віддає квант часу і дає команду планувальникові запустити алгоритм пошуку готового потоку);
3. потік перейшов у стан готовності до виконання (*алгоритм розміщення готового потоку).*

Планувальник підтримує спеціальну структуру даних — *список готових потоків*. У списку зберігається 31 елемент — по одному для кожного рівня пріоритету. З кожним елементом пов'язана черга готових потоків, всі потоки з однаковим пріоритетом перебувають у черзі, яка відповідає їхньому рівню пріоритету.

Під час виконання алгоритму пошуку готового потоку планувальник переглядає всі черги потоків, починаючи з черги найвищого пріоритету (31). Як тільки під час перегляду трапляється потік, його вибирають для виконання.

Алгоритм розміщення готового потоку поміщає потік у список готових потоків. Спочатку перевіряють, чи не володіє потік вищим пріоритетом, ніж той, котрий виконується в цей момент. При цьому новий потік негайно починає виконуватися, а поточний поміщається у список готових потоків; у противному разі новий потік поміщається в чергу списку готових потоків, відповідну до його пріоритету. У початку кожної черги розташовані потоки, які були витиснені до того, як вони виконувалися впродовж хоча б одного кванта, всі інші потоки поміщаються в кінець черги.

Потік може бути витиснений коли:

1. потік перейшов у стан очікування;
2. минув квант часу потоку;
3. потік із вищим пріоритетом перейшов у стан готовності до виконання;
4. змінився пріоритет потоку або пріоритет іншого потоку.

Динамічна зміна пріоритету і кванта часу

Під час виконання потоків динамічний пріоритет і довжина кванта часу можуть бути скориговані ядром системи. Є два види такої динамічної зміни: *підтримка* і *ослаблення*.

Підтримка зводиться до тимчасового підвищення пріоритету потоків. Коли потік переходить у стан готовності до виконання внаслідок настання очікуваної події, виконують операцію підтримки.

Під час завершення операції введення-виведення підвищення пріоритету залежить від типу операції.

Під час зміни стану синхронізаційного об'єкта пріоритет потоку, який очікує цієї зміни, збільшують на одиницю.

Вихід з будь-якого стану очікування для потоків інтерактивних застосувань призводить до підвищення пріоритету на 2, таке саме підвищення відбувається під час переходу в стан готовності потоків, пов'язаних із відображенням інтерфейсу користувача.

Для запобігання голодуванню потоки, які не виконувалися упродовж досить тривалого часу, різко підвищують свій пріоритет.

Внаслідок операцій підтримки динамічний пріоритет потоку не може перевищити значення 15.

Після закінчення кожного кванта часу поточний пріоритет потоку зменшують на одиницю, поки він не дійде до базового, після чого пріоритет залишають на одному рівні до наступної операції підтримки.

Видом підтримки є зміна кванта часу для інтерактивних застосувань. Якщо задано використання квантів змінної довжини, для інтерактивних застосувань довжина

кванта буде збільшуватися. Якщо така підтримка задана, то коли інтерактивне застосування захоплює фокус, всі його потоки отримують квант часу, який дорівнює значенню підтримки (наприклад, 40 або 60 мс).

Значення кванта може й зменшуватися (слабшати). Під час виконання будь-якої функції очікування довжина кванта зменшується на одиницю.

Запобігання голодуванню

Якщо в системі постійно є потоки з високим пріоритетом, може виникати голодування потоків, пріоритет яких нижчий. Для уникнення голодування, спеціальний потік ядра один раз за секунду обходить чергу готових потоків у пошуках тих, які перебували у стані готовності досить довго (понад 3 с) і жодного разу не отримали шансу на виконання. Коли такий потік знайдено, то йому присвоюють пріоритет 15 (і він дістає змогу негайного виконання). Крім того, довжину його кванта часу подвоюють. Після того, як два кванти часу минають, пріоритет потоку і його квант повертаються до вихідних значень.

3 Програмний інтерфейс планування

3.1 Функції Win API для роботи пріоритетами процесів і потоків.

Зміна класу пріоритету процесу виконується функцією SetPriorityClass(), для визначення поточного класу пріоритету - функцію GetPriorityClass():

```
DWORD GetPriorityClass(
  [in] HANDLE hProcess
);
```

```
BOOL SetPriorityClass(
  [in] HANDLE hProcess,
  [in] DWORD  dwPriorityClass
);
```

`hProcess` - *дескриптор процесу;*

`dwPriorityClass` - *клас пріоритету.*

Завдання відносного пріоритету потоку виконується функцією (),а визначення пріоритету - GetThreadPriority().

```
int GetThreadPriority(
  [in] HANDLE hThread
);
```

```
BOOL SetThreadPriority(
  [in] HANDLE hThread,
  [in] int    nPriority
);
```

`hThread` - *дескриптор потоку;*
`nPriority` - *пріоритету потоку.*

3.2 Приклад реалізації пріоритетів процесу:

```
HANDLE curh = SetCurrentProcess();
//задати клас пріоритету для поточного процесу
SetPriorityClass(curh, IDLE_PRIORITY_CLASS);
// взнати поточне значення класу пріоритету
printf("Поточний клас пріоритету: %d\n", GetPriorityClass(curh));
```

3.3 Приклад реалізації пріоритетів потоку:

```
DWORD tid;
// створення потоку
HANDLE th = _beginthreadex(... CREATE_SUSPENDED, &tid);
//задання пріоритету
SetThreadPriority(th,THREAD_PRIORITY_IDLE);
// поновлення виконання потоку
ResumeThread(th);
// визначення пріоритету
printf("Поточний пріоритет потоку: %d\n", GetThreadPriority(th));
// закриття дескриптора потоку
CloseHandle(th);
```

4 Приклади програмної реалізації планування процесів та потоків

Приклад 1. Приклад роботи функцій *GetPriorityClass* та *GetPriorityClass.*

```
#include <windows.h>
#include
<conio.h>int
main()
{
```

```
    HANDLE
    hProcess;
    DWORD
    dwPriority;

    //отримуємо дескриптор поточного
    процесуhProcess = GetCurrentProcess();

    //визначаємо пріоритет поточного
    процесуdwPriority =
    GetPriorityClass(hProcess);

    //встановлюємо фоновий пріоритет поточного
    процесуif (!SetPriorityClass(hProcess,
    IDLE_PRIORITY_CLASS))
    {
        cputs("\n Set priority class failed.
        \n");cputs("\n Press any key to exit.
        \n"); getch();
        return GetLastError();
    }

    dwPriority = GetPriorityClass(hProcess);
    cprintf("\n The priority of the process = %d. \n", dwPriority);

    //встановлюємо високий пріоритет поточного
    процесуif (!SetPriorityClass(hProcess,
    HIGH_PRIORITY_CLASS))
    {
        cputs("\n Set priority class failed.
        \n");cputs("\n Press any key to exit.
        \n"); getch();
        return GetLastError();
    }
    dwPriority = GetPriorityClass(hProcess);
    cprintf("\n The priority of the process = %d. \n", dwPriority);

        cputs("\n Press any key to exit.
        \n");getch();

    return 0;
}
```

Приклад 2 Визначення та зміна пріоритетів потоків.

```
#include
<windows.h>
#include
<conio.h>
int main()
{
   HANDLE
   hThread;
   DWORD
   dwPriority;

   //отримуємо дескриптор поточного
   потокуhThread = GetCurrentThread();

   //понижуємо пріоритет поточного потоку
   if (!SetThreadPriority(hThread, THREAD_PRIORITY_LOWEST))
   {
         cputs("\n Set thread priority  failed.
         \n");cputs("\n Press any key to exit.
         \n"); getch();
         return GetLastError();
   }
   //визначаємо рівень пріоритету поточного потоку
   dwPriority = GetThreadPriority(hThread);
    cprintf("\n The priority level of the thread = %d. \n", dwPriority);
   //підвищуємо пріоритет поточного процесу
   if (!SetThreadPriority(hThread, THREAD_PRIORITY_HIGHEST))
   {
         cputs("\n Set priority class failed. \n");
         cputs("\n Press any key to exit. \n");
         getch();
         return GetLastError();
   }
   //визначаємо рівень пріоритету поточного потоку
   dwPriority = GetThreadPriority(hThread);
   cprintf("\n The priority level of the thread = %d. \n", dwPriority);
         cputs("\n Press any key to exit.
         \n");getch();
   return 0;
}
```

Приклад 3 Керування динамічною зміною пріоритетів потоків.

```
#include
<windows.h>
#include <conio.h>

int main()
{
        HANDLE hProcess,
        hThread;BOOL
        bPriorityBoost;

        //отримуємо дескриптор поточного
        процесуhProcess = GetCurrentProcess();

        //визначаємо режим динамічного підвищення пріоритетів для процесу
        if (!GetProcessPriorityBoost(hProcess, &bPriorityBoost))
        {
                cputs("\n Set process priority boost failed. \n");
                cputs("\n Press any key to exit. \n");
                getch();
                return GetLastError();
        }
        cprintf("\n The process priority boost = %d. \n", bPriorityBoost);
        //виключаємо режим динамічного підвищення пріоритетів для процесуif
        (!SetProcessPriorityBoost(hProcess, TRUE))
        {
                cputs("\n Set process priority boost failed. \n");
                cputs("\n Press any key to exit. \n");
                getch();
                return GetLastError();
        }

        //отримуємо дескриптор поточного
        потокуhThread = GetCurrentThread();

        //визначаємо режим динамічного підвищення пріоритетів для процесу
        if (!GetProcessPriorityBoost(hProcess, &bPriorityBoost))
        {
                cputs("\n Set process priority boost failed. \n");
                cputs("\n Press any key to exit. \n");
                getch();
                return GetLastError();
        }
```

```
        cprintf("\n The process priority boost = %d. \n", bPriorityBoost);
        //включаємо режим динамічного підвищення пріоритетів для потоку
        if (!SetProcessPriorityBoost(hProcess, FALSE))
        {
                cputs("\n Set process priority boost failed. \n");
                cputs("\n Press any key to exit. \n");
                getch();
                return GetLastError();
        }
                cputs("\n Press any key to exit.
                \n");getch();
        return 0;
}
```

Завдання для студентів

1. Розробити програми для роботи з процесами та потоками. Для програмної реалізації використовувати середовище програмування Microsoft Visual Studio, мова програмування – C/C++, інтерфейс консольний.
2. Скласти звіт про виконання лабораторної роботи.
   Зміст звіту:
   - ➢ опис функцій для роботи з процесами та потоками;
   - ➢ постановка задачі;
   - ➢ програмний код;
   - ➢ результати виконання програми;
   - ➢ висновок.
3. До захисту лабораторної роботи підготувати відповіді на контрольні питання.

Контрольні питання

1. Охарактеризувати алгоритми планування: планування за принципом FIFO, кругове планування, планування із пріоритетами.
2. Охарактеризувати алгоритми планування: планування на підставі характеристик подальшого виконання, багаторівневі черги зі зворотним зв'язком, лотерейне планування.
3. Як виконується планування у ядрі? Охарактеризувати пріоритети потоків і процесів.
4. Як виконується пошук потоку для виконання?
5. Як виконується динамічна зміна пріоритету і кванта часу?

6. Як виконується запобігання голодуванню?
7. Як виконується зміна класу пріоритету процесу? Дати характеристику функціям, які при цьому використовуються. Проаналізувати параметри функцій.
8. Як виконується визначення поточного класу пріоритету? Дати характеристику функції, яка при цьому використовуються. Проаналізувати параметри функції.
9. Як виконується завдання відносного пріоритету потоку? Дати характеристику функції, яка при цьому використовуються. Проаналізувати параметри функції.
10. Як виконується визначення пріоритету потоку? Дати характеристику функції, яка при цьому використовуються. Проаналізувати параметри функції.

Варіанти завдань:

1. 1. Розробити програму, яка обчислює число Фібоначчі за заданим номером n за формулою $F_n = F_{n-1} + F_{n-2}$, де $F_0 = F_1 = 1$. Обчислення числа Фібоначчі оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
2. Розробити програму, яка шукає найбільше число у динамічному масиві цілих чисел розміру n (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук найбільшого оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
3. Розробити програму, яка шукає найбільше число у динамічному двовимірному масиві цілих чисел розміру $n \times n$ (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук найбільшого оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
4. Розробити програму, яка сортує динамічний масив цілих чисел розміру n за зростанням (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Сортування масиву оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
5. Розробити програму, яка обчислює середнє арифметичне елементів динамічного двовимірного масиву цілих чисел розміру $n \times n$ (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук середнього арифметичного оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
6. Розробити програму, яка обчислює суму парних чисел від L до U. Обчислення суми оформити як функцію потоку. Запустити потік на виконання з декількома рівнями

пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.

7. Розробити програму, яка шукає кількість додатних чисел у динамічному масиві цілих чисел розміру *n* (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук кількісті додатних чисел оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
8. Розробити програму, яка шукає кількість повторень заданого числа *x* у динамічному масиві цілих чисел розміру *n* (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук кількість повторень заданого числа оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
9. Розробити програму, яка шукає кількість пробілів у текстовому файлі. Пошук кількісті пробілів оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
10. Розробити програму, яка шукає кількість цифр у текстовому файлі. Пошук кількісті цифр оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
11. Розробити програму, яка обчислює суму квадратів елементів динамічного масиву цілих чисел розміру *n (*масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук суми квадратів оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
12. Розробити програму, яка шукає кількість чисел, які більші за своїх сусідів, у динамічному масиві цілих чисел розміру *n* (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук кількісті чисел, які більші за своїх сусідів, оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
13. Розробити програму, яка шукає скільки раз повторюється найбільше число у динамічному масиві цілих чисел розміру *n* (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук кількості найбільшого числа оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes().

Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.

14. Розробити програму, яка обчислює кількість дільників заданого числа *n*. Обчислення кількості дільників оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
15. Розробити програму, яка шукає найменше число у динамічному масиві цілих чисел розміру *n* (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук найменшого оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
16. Розробити програму, яка шукає найменше число у динамічному двовимірному масиві цілих чисел розміру $n \times n$ (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук найменшого оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
17. Розробити програму, яка сортує динамічний масив цілих чисел розміру *n* за спаданням (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Сортування масиву оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
18. Розробити програму, яка обчислює суму непарних чисел від L до U. Обчислення суми оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
19. Розробити програму, яка шукає кількість від'ємних чисел у динамічному масиві цілих чисел розміру *n* (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук кількісті від'ємних чисел оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
20. Розробити програму, яка шукає кількість нулів у динамічному масиві цілих чисел розміру *n* (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук кількісті нулів оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
21. Розробити програму, яка шукає кількість малих латинських літер у текстовому файлі. Пошук кількісті малих латинських літер оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за

допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.

22. Розробити програму, яка шукає кількість великих латинських літер у текстовому файлі. Пошук кількісті великих латинських літер оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
23. Розробити програму, яка шукає кількість чисел, які менші за своїх сусідів, у динамічному масиві цілих чисел розміру n (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук кількісті чисел, які менші за своїх сусідів оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
24. Розробити програму, яка шукає порядковий номер найбільшого числа у динамічному масиві цілих чисел розміру n (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук порядкового номеру найбільшого числа оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
25. Розробити програму, яка шукає порядковий номер найменшого числа у динамічному масиві цілих чисел розміру n (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Пошук порядкового номеру найменшого числа оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
26. Розробити програму, яка шукає кількість рядків у текстовому файлі. Пошук кількісті рядків оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
27. Розробити програму, яка обчислює суму чисел у динамічному двовимірному масиві цілих чисел розміру $n \times n$ (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Обчислення суми чисел оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
28. Розробити програму, яка заміняє усі додатні числа на нуль у динамічному масиві цілих чисел розміру n (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Заміну усіх додатних числа на нуль оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.

29. Розробити програму, яка заміняє усі від’ємні числа на нуль у динамічному масиві цілих чисел розміру *n* (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Заміну усіх від’ємних чисел на нуль оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці.
30. Розробити програму, яка заміняє усі від’ємні числа на *-1*, а додатні на *1* у динамічному масиві цілих чисел розміру *n* (масив заповнюється значеннями, згенерованими за допомогою функцій генерування псевдовипадкових чисел). Заміну чисел оформити як функцію потоку. Запустити потік на виконання з декількома рівнями пріоритету, визначити час виконання потоку за допомогою функції GetThreadTimes(). Запустити програму декілька раз з різними вхідними даними, результати оформити у вигляді таблиці

Приклад виконання:

Пріоритети планування

Виконання потоків заплановано на основі їхнього пріоритету планування. Кожному потоку призначається пріоритет планування. Рівні пріоритету варіюються від нуля (найнижчий пріоритет) до 31 (найвищий пріоритет). Лише потік із нульовою сторінкою може мати нульовий пріоритет. (Потік нульової сторінки — це системний потік, відповідальний за обнулення будь-яких вільних сторінок, коли немає інших потоків, які потрібно виконувати.)

Система розглядає всі потоки з однаковим пріоритетом як рівні. Система призначає проміжки часу циклічно всім потокам із найвищим пріоритетом. Якщо жоден із цих потоків не готовий до виконання, система призначає проміжки часу циклічно всім потокам із наступним найвищим пріоритетом. Якщо потік з вищим пріоритетом стає доступним для виконання, система припиняє виконання потоку з нижчим пріоритетом (не дозволяючи йому завершити використання свого інтервалу часу) і призначає повний сегмент потоку з вищим пріоритетом. Пріоритет кожного потоку визначається наступними критеріями:

• Клас пріоритету процесу

• Рівень пріоритету потоку в межах класу пріоритету його процесу

Клас пріоритету та рівень пріоритету поєднуються, щоб сформувати базовий пріоритет потоку.

Клас пріоритету процесу

Кожен процес належить до одного з наступних класів пріоритету:

IDLE_PRIORITY_CLASS (код 0x00000040)
BELOW_NORMAL_PRIORITY_CLASS (код 0x00004000)
NORMAL_PRIORITY_CLASS (код 0x00000020)
ABOVE_NORMAL_PRIORITY_CLASS (код 0x00008000)
HIGH_PRIORITY_CLASS (код 0x00000080)
REALTIME_PRIORITY_CLASS (код 0x00000100)

За замовчуванням клас пріоритету процесу становить NORMAL_PRIORITY_CLASS. Використовуйте функцію GetPriorityClass(), щоб визначити поточний клас пріоритету процесу, і функцію SetPriorityClass(), щоб змінити клас пріоритету процесу.

Процеси, такі як заставки або програми, які періодично оновлюють дисплей, повинні використовувати IDLE_PRIORITY_CLASS. Це запобігає втручанню потоків цього процесу, які не мають високого пріоритету, у потоки з вищим пріоритетом.

Обережно використовуйте HIGH_PRIORITY_CLASS. Якщо потік працює з найвищим рівнем пріоритету протягом тривалого часу, інші потоки в системі не отримають процесорний час. Якщо кілька потоків мають високий пріоритет одночасно, потоки втрачають свою ефективність. Клас високого пріоритету має бути зарезервований для потоків, які повинні реагувати на критичні за часом події. Якщо ваша програма виконує одне завдання, для якого потрібен клас високого пріоритету, а решта її завдань мають звичайний пріоритет, використовуйте SetPriorityClass(), щоб тимчасово підвищити клас пріоритету програми; потім зменшіть його після виконання критичного за часом завдання. Інша стратегія полягає у створенні процесу з високим пріоритетом, у якому всі потоки блокуються більшу частину часу, пробуджуючи потоки лише тоді, коли потрібні критичні завдання. Важливим моментом є те, що потік з високим пріоритетом повинен виконуватися протягом короткого часу і лише тоді, коли йому потрібно виконати критичну за часом роботу.

Старайтеся не використовувати REALTIME_PRIORITY_CLASS, оскільки це перериває системні потоки, які керують введенням даних миші, введенням даних з клавіатури та фоновим очищенням диска. Цей клас може підійти для програм, які «спілкуються» безпосередньо з обладнанням або виконують короткі завдання, які повинні мати обмежені перерви.

Рівень пріоритету потоку

Нижче наведено рівні пріоритету в кожному класі пріоритету:

THREAD_PRIORITY_IDLE
THREAD_PRIORITY_LOWEST
THREAD_PRIORITY_BELOW_NORMAL
THREAD_PRIORITY_NORMAL
THREAD_PRIORITY_ABOVE_NORMAL
THREAD_PRIORITY_HIGHEST
THREAD_PRIORITY_TIME_CRITICAL

Усі потоки створюються з рівнем пріоритету THREAD_PRIORITY_NORMAL. Це означає, що

пріоритет потоку такий самий, як і клас пріоритету процесу. Після створення потоку скористайтеся функцією SetThreadPriority(), щоб налаштувати його пріоритет відносно інших потоків у процесі.

Щоб визначити поточний рівень пріоритету потоку, використовуйте функцію GetThreadPriority().

Базовий пріоритет потоку

Клас пріоритету процесу та рівень пріоритету потоку поєднуються, щоб сформувати базовий пріоритет кожного потоку.

У наведеній нижче таблиці показано базовий пріоритет для комбінацій класу пріоритету процесу та значення пріоритету потоку.

Клас пріоритету процесу	**Рівень пріоритету потоку**	**Базовий пріоритет**
IDLE_PRIORITY_CLASS	THREAD_PRIORITY_IDLE	1
	THREAD_PRIORITY_LOWEST	2
	THREAD_PRIORITY_BELOW_NORMAL	3
	THREAD_PRIORITY_NORMAL	4
	THREAD_PRIORITY_ABOVE_NORMAL	5
	THREAD_PRIORITY_HIGHEST	6
	THREAD_PRIORITY_TIME_CRITICAL	15
BELOW_NORMAL_PRIORITY_CLASS	THREAD_PRIORITY_IDLE	1
	THREAD_PRIORITY_LOWEST	4
	THREAD_PRIORITY_BELOW_NORMAL	5
	THREAD_PRIORITY_NORMAL	6
	THREAD_PRIORITY_ABOVE_NORMAL	7
	THREAD_PRIORITY_HIGHEST	8
	THREAD_PRIORITY_TIME_CRITICAL	15
NORMAL_PRIORITY_CLASS	THREAD_PRIORITY_IDLE	1
	THREAD_PRIORITY_LOWEST	6
	THREAD_PRIORITY_BELOW_NORMAL	7
	THREAD_PRIORITY_NORMAL	8
	THREAD_PRIORITY_ABOVE_NORMAL	9
	THREAD_PRIORITY_HIGHEST	10
	THREAD_PRIORITY_TIME_CRITICAL	15
ABOVE_NORMAL_PRIORITY_CLASS	THREAD_PRIORITY_IDLE	1
	THREAD_PRIORITY_LOWEST	8
	THREAD_PRIORITY_BELOW_NORMAL	9
	THREAD_PRIORITY_NORMAL	10
	THREAD_PRIORITY_ABOVE_NORMAL	11
	THREAD_PRIORITY_HIGHEST	12
	THREAD_PRIORITY_TIME_CRITICAL	15
HIGH_PRIORITY_CLASS	THREAD_PRIORITY_IDLE	1
	THREAD_PRIORITY_LOWEST	11
	THREAD_PRIORITY_BELOW_NORMAL	12
	THREAD_PRIORITY_NORMAL	13
	THREAD_PRIORITY_ABOVE_NORMAL	14
	THREAD_PRIORITY_HIGHEST	15
	THREAD_PRIORITY_TIME_CRITICAL	15
REALTIME_PRIORITY_CLASS	THREAD_PRIORITY_IDLE	16
	THREAD_PRIORITY_LOWEST	22
	THREAD_PRIORITY_BELOW_NORMAL	23
	THREAD_PRIORITY_NORMAL	24

	THREAD_PRIORITY_ABOVE_NORMAL	25
	THREAD_PRIORITY_HIGHEST	26
	THREAD_PRIORITY_TIME_CRITICAL	31

```c
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
// Програма перемножує між собою дві матриці розміру n
// Множення матриць оформлено у вигляді функції потоку.
// Потік запускається на виконання декілька раз з різними рівнями пріоритету
//
// Структура для передачі аргументів у функцію потоку
struct ThreadArgs
{
	int array_size; // Розмір масиву
};

// Функція потоку
DWORD WINAPI multMatrix(LPVOID lpParam)
{
	struct ThreadArgs* threadArgs = (struct ThreadArgs*)lpParam;
	int size = threadArgs->array_size;
	int i, j, l;

	// Генерування динамічного масиву з випадковими значеннями
	char** a = (char**)malloc(size * sizeof(char*));
	for (i = 0; i < size; i++)
		a[i] = (char*)malloc(size * sizeof(char));

	for (i = 0; i < size; i++)
		for (j = 0; j < size; j++)
		{
			a[i][j] = rand() % 10;
		}
	// Генерування динамічного масиву для зберігання результатів
	char** c = (char**)malloc(size * sizeof(char*));
	for (i = 0; i < size; i++)
		c[i] = (char*)malloc(size * sizeof(char));

	// Множення матриць
	for (i = 0; i < size; i++)
		for (j = 0; j < size; j++)
		{
			c[i][j] = 0;
			for (l = 0; l < size; l++)
				c[i][j] = c[i][j] + a[i][l] * a[l][j];
		}


	for (int i = 0; i < size; i++)
		free(a[i]);
	free(a);

	for (int i = 0; i < size; i++)
		free(c[i]);
	free(c);

	return 0;
}

// Функція для друку результатів
void printRes(HANDLE* hThread)
{
	printf("-------------------------------------------------------------------------
-----------------------------------------\n");
	printf("|  Thread  |  Kernel Time,  |    User Time,    |  Execution Time,  |
Thread     |      Thread    |  Thread Cycle |\n");
	printf("| Priority |  milliseconds  |   milliseconds   |   milliseconds    |
creation time  |     end time   |   Time, tacts |\n");
```

```
	printf("-------------------------------------------------------------------------
-----------------------------------------\n");

	for (int i = 0; i < 7; i++)
	{

		int p = GetThreadPriority(hThread[i]);
		printf("|     %3d  |", p);

		// Визначення часу виконання потоку за допомогою GetThreadTimes()
		FILETIME creationTime, exitTime, kernelTime, userTime;
		FILETIME creationLocalTime, exitLocalTime;
		if (GetThreadTimes(hThread[i], &creationTime, &exitTime, &kernelTime,
&userTime))
		{
			ULARGE_INTEGER kernelTimeInt, userTimeInt;
			kernelTimeInt.LowPart = kernelTime.dwLowDateTime;
			kernelTimeInt.HighPart = kernelTime.dwHighDateTime;
			userTimeInt.LowPart = userTime.dwLowDateTime;
			userTimeInt.HighPart = userTime.dwHighDateTime;
			printf(" %13.5f  |", kernelTimeInt.QuadPart * 1e-4);
			printf(" %15.5f  |", userTimeInt.QuadPart * 1e-4);

			ULARGE_INTEGER creationTimeInt, exitTimeInt;
			creationTimeInt.LowPart = creationTime.dwLowDateTime;
			creationTimeInt.HighPart = creationTime.dwHighDateTime;
			exitTimeInt.LowPart = exitTime.dwLowDateTime;
			exitTimeInt.HighPart = exitTime.dwHighDateTime;
			double exeTime = (exitTimeInt.QuadPart - creationTimeInt.QuadPart) *
1e-4;
			printf(" %16.5f  |", exeTime);

			SYSTEMTIME systemTime;
			FileTimeToLocalFileTime(&creationTime, &creationLocalTime);
			FileTimeToLocalFileTime(&exitTime, &exitLocalTime);
			FileTimeToSystemTime(&creationLocalTime, &systemTime); //
Перетворення часу у формат SYSTEMTIME
			printf("   %02u:%02u:%02u:%03u  |", systemTime.wHour,
systemTime.wMinute, systemTime.wSecond, systemTime.wMilliseconds);
			FileTimeToSystemTime(&exitLocalTime, &systemTime); // Перетворення
часу у формат	SYSTEMTIME
			printf("  %02u:%02u:%02u:%03u  |", systemTime.wHour,
systemTime.wMinute, systemTime.wSecond, systemTime.wMilliseconds);

			ULONGLONG ThreadCycleTime;
			QueryThreadCycleTime(hThread[i], &ThreadCycleTime);
			printf("%14llu |\n", ThreadCycleTime);

			printf("-------------------------------------------------------------
-------------------------------------------------------\n");
		}
	}
}

void ParallelThreadExecution(int n)
{
	struct ThreadArgs threadArgs;
	threadArgs.array_size = n;

	HANDLE hThread[7];
	DWORD dwThreadId[7];

	int i;

	// Створення потоків
```

```
	for (i = 0; i < 7; i++)
	{
		hThread[i] = CreateThread(
			NULL,                   // default security attributes
			0,                      // use default stack size
			multMatrix,             // thread function
			&threadArgs,        // argument to thread function
			CREATE_SUSPENDED,   // create suspended initially
			&dwThreadId[0]);    // returns the thread identifier

		if (hThread[0] == NULL)
		{
			fprintf(stderr, "Error creating thread (%lu).\n", GetLastError());
			exit(1);
		}
	}

	// Задання рівня пріоритету потоків
	if (!SetThreadPriority(hThread[0], THREAD_PRIORITY_IDLE))
		printf("\nSet Thread[0] Priority %d is failed !!!\n",
THREAD_PRIORITY_IDLE);

	for (int i = 1; i < 6; i++)
	{
		if (!SetThreadPriority(hThread[i], i - 3))
			printf("\nSet Thread[%d] Priority %d is failed !!!\n", i, i - 3);
	}

	if (!SetThreadPriority(hThread[6], THREAD_PRIORITY_TIME_CRITICAL))
		printf("\nSet Thread[6] Priority %d is failed !!!\n",
THREAD_PRIORITY_TIME_CRITICAL);

	for (i = 0; i < 7; i++)
		SetThreadPriorityBoost(hThread[i], TRUE);


	// Запускаємо потоки у виконання
	for (i = 0; i < 7; i++)
	{
		ResumeThread(hThread[i]);
	}

	// Чекаємо поки потоки завершаться
	WaitForMultipleObjects(7, hThread, TRUE, INFINITE);

	// Виводимо результати
	printRes(hThread);

	// Закриття дескрипторів потоків
	for (int i = 0; i < 7; i++)
		CloseHandle(hThread[i]);
}

void SequentialThreadExecution(int n)
{
	struct ThreadArgs threadArgs;
	threadArgs.array_size = n;

	int i;

	HANDLE hThread[7];
	DWORD dwThreadId[7];

	// Створюємо, запускаємо і очікуємо завершення 0-го потоку
	hThread[0] = CreateThread(
```

```
		NULL,                    // default security attributes
		0,                              // use default stack size
		multMatrix,              // thread function
		&threadArgs,        // argument to thread function
		CREATE_SUSPENDED,   // create suspended initially
		&dwThreadId[0]);    // returns the thread identifier

	if (hThread[0] == NULL)
	{
		fprintf(stderr, "Error creating thread (%lu).\n", GetLastError());
		exit(1);
	}

	BOOL success = SetThreadPriority(hThread[0], THREAD_PRIORITY_IDLE);
	if (!success)
		printf("\nSet Thread[0] Priority %d is failed !!!\n",
THREAD_PRIORITY_IDLE);

	SetThreadPriorityBoost(hThread[0], TRUE);

	ResumeThread(hThread[0]);
	WaitForSingleObject(hThread[0], INFINITE);

	// Створюємо, запускаємо і очікуємо завершення потоків від 1 до 5
	for (i = 1; i < 6; i++)
	{
		hThread[i] = CreateThread(
			NULL,                    // default security attributes
			0,                              // use default stack size
			multMatrix,              // thread function
			&threadArgs,        // argument to thread function
			CREATE_SUSPENDED,   // create suspended initially
			&dwThreadId[i]);    // returns the thread identifier

		if (hThread[i] == NULL)
		{
			fprintf(stderr, "Error creating thread (%lu).\n", GetLastError());
			exit(1);
		}

		success = SetThreadPriority(hThread[i], i - 3);
		if (!success)
			printf("\nSet Thread[%d] Priority %d is failed !!!\n", i, i - 3);

		SetThreadPriorityBoost(hThread[i], TRUE);

		ResumeThread(hThread[i]);
		WaitForSingleObject(hThread[i], INFINITE);
	}

	// Створюємо, запускаємо і очікуємо завершення 6-го потоку
	hThread[6] = CreateThread(
		NULL,                    // default security attributes
		0,                              // use default stack size
		multMatrix,              // thread function
		&threadArgs,        // argument to thread function
		CREATE_SUSPENDED,   // create suspended initially
		&dwThreadId[6]);    // returns the thread identifier

	if (hThread[6] == NULL)
	{
		fprintf(stderr, "Error creating thread (%lu).\n", GetLastError());
		exit(1);
	}
```

```
	success = SetThreadPriority(hThread[6], THREAD_PRIORITY_TIME_CRITICAL);
	if (!success)
		printf("\nSet Thread[7] Priority %d is failed !!!\n",
THREAD_PRIORITY_TIME_CRITICAL);

	SetThreadPriorityBoost(hThread[6], TRUE);

	ResumeThread(hThread[6]);

	WaitForSingleObject(hThread[6], INFINITE);

	printRes(hThread);

	// Закриття дескрипторів
	for (int i = 0; i < 7; i++)
		CloseHandle(hThread[i]);
}

int main()
{

	// Отримання дескриптора поточного процесу
	HANDLE hProcess = GetCurrentProcess();

	// Отримання пріоритету класу для поточного процесу
	int priorityClass = GetPriorityClass(hProcess);
	printf("-------------------------------------------------------\n");
	printf("\nPriority class for the current process is : %#010x\n", priorityClass);

	int k;
	printf("\nIf you want to set the priority class to IDLE_PRIORITY_CLASS press 1,
to HIGH_PRIORITY_CLASS  press 2\n");
	printf("\else, press any other key : ");
	scanf_s("%d", &k);

	if( k == 1)
	{
		if (SetPriorityClass(hProcess, IDLE_PRIORITY_CLASS))
		{
			printf("\nSet Priority Class is success !!!\n");
			priorityClass = GetPriorityClass(hProcess);
			printf("Priority class for the current process is : %#010x\n",
priorityClass);
		}
		else
			printf("\nSet Priority Class is failed !!!\n");
	}
	else
		if (k == 2)
		{
			if (SetPriorityClass(hProcess, HIGH_PRIORITY_CLASS))
			{
				printf("\nSet Priority Class is success !!!\n");
				priorityClass = GetPriorityClass(hProcess);
				printf("Priority class for the current process is : %#010x\n",
priorityClass);
			}
			else
				printf("\nSet Priority Class is failed !!!\n");
		}


	int n;
	printf("\n-------------------------------------------------------\n");
	printf("Enter the size of the array: ");
```

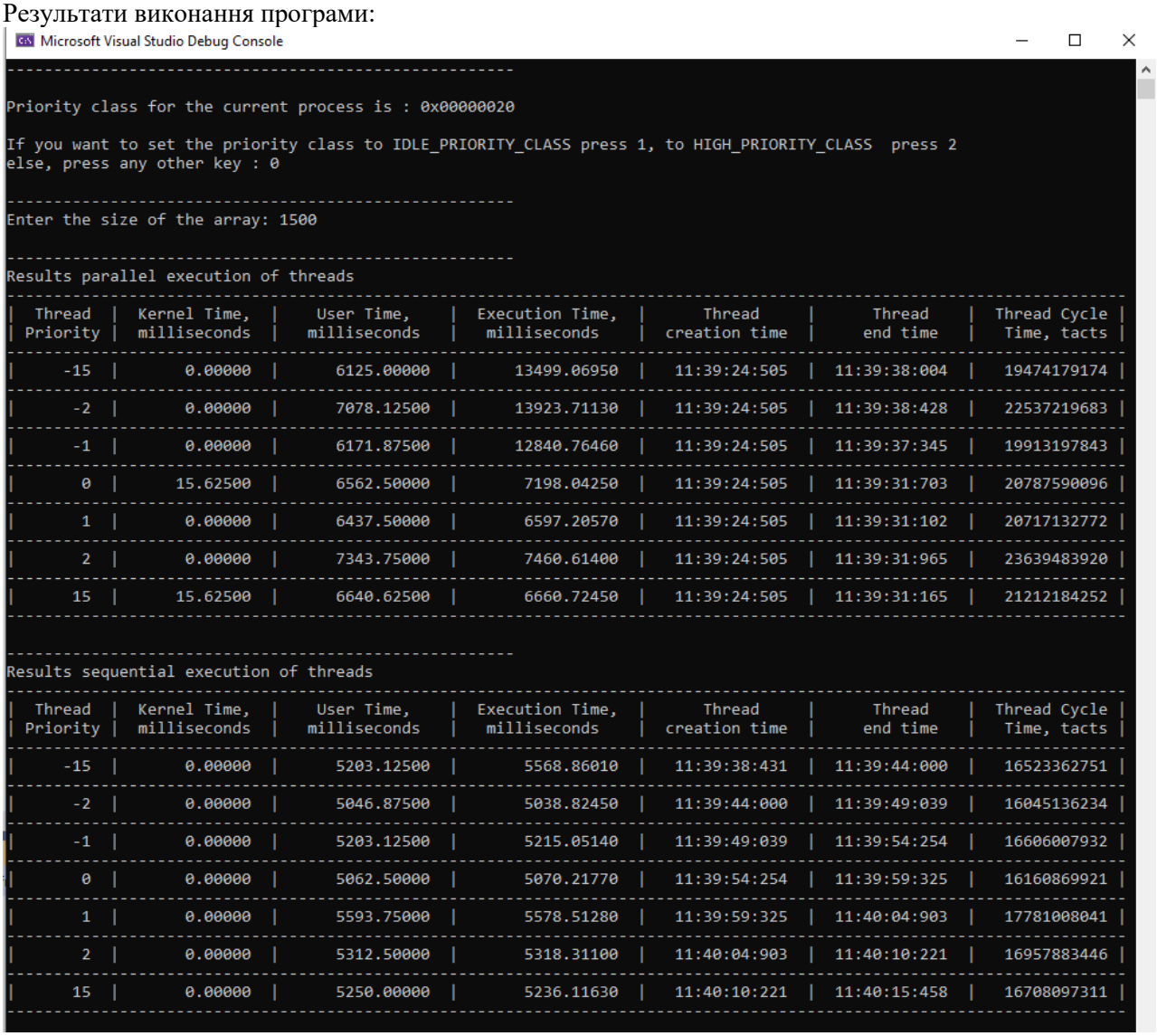
```
        scanf_s("%d", &n);

        printf("\n-----------------------------------------------------\n");
        printf("Results parallel execution of threads\n");
        ParallelThreadExecution(n);

        printf("\n-----------------------------------------------------\n");
        printf("Results sequential execution of threads\n");
        SequentialThreadExecution(n);

        return 0;
}
```

Результати виконання програми:



Microsoft Visual Studio Debug Console

```
-------------------------------------------------------

Priority class for the current process is : 0x00000020

If you want to set the priority class to IDLE_PRIORITY_CLASS press 1, to HIGH_PRIORITY_CLASS  press 2
else, press any other key : 0

-------------------------------------------------------
Enter the size of the array: 1500

-------------------------------------------------------
Results parallel execution of threads
-------------------------------------------------------------------------------------------------------------------------
|  Thread   |  Kernel Time,  |    User Time,    |  Execution Time,  |      Thread      |      Thread      | Thread Cycle |
| Priority  |  milliseconds  |   milliseconds   |    milliseconds   |  creation time   |     end time     |   Time, tacts |
-------------------------------------------------------------------------------------------------------------------------
|     -15   |       0.00000  |      6125.00000  |      13499.06950  |   11:39:24:505   |  11:39:38:004    |   19474179174 |
-------------------------------------------------------------------------------------------------------------------------
|      -2   |       0.00000  |      7078.12500  |      13923.71130  |   11:39:24:505   |  11:39:38:428    |   22537219683 |
-------------------------------------------------------------------------------------------------------------------------
|      -1   |       0.00000  |      6171.87500  |      12840.76460  |   11:39:24:505   |  11:39:37:345    |   19913197843 |
-------------------------------------------------------------------------------------------------------------------------
|       0   |      15.62500  |      6562.50000  |       7198.04250  |   11:39:24:505   |  11:39:31:703    |   20787590096 |
-------------------------------------------------------------------------------------------------------------------------
|       1   |       0.00000  |      6437.50000  |       6597.20570  |   11:39:24:505   |  11:39:31:102    |   20717132772 |
-------------------------------------------------------------------------------------------------------------------------
|       2   |       0.00000  |      7343.75000  |       7460.61400  |   11:39:24:505   |  11:39:31:965    |   23639483920 |
-------------------------------------------------------------------------------------------------------------------------
|      15   |      15.62500  |      6640.62500  |       6660.72450  |   11:39:24:505   |  11:39:31:165    |   21212184252 |
-------------------------------------------------------------------------------------------------------------------------

-------------------------------------------------------
Results sequential execution of threads
-------------------------------------------------------------------------------------------------------------------------
|  Thread   |  Kernel Time,  |    User Time,    |  Execution Time,  |      Thread      |      Thread      | Thread Cycle |
| Priority  |  milliseconds  |   milliseconds   |    milliseconds   |  creation time   |     end time     |   Time, tacts |
-------------------------------------------------------------------------------------------------------------------------
|     -15   |       0.00000  |      5203.12500  |       5568.86010  |   11:39:38:431   |  11:39:44:000    |   16523362751 |
-------------------------------------------------------------------------------------------------------------------------
|      -2   |       0.00000  |      5046.87500  |       5038.82450  |   11:39:44:000   |  11:39:49:039    |   16045136234 |
-------------------------------------------------------------------------------------------------------------------------
|      -1   |       0.00000  |      5203.12500  |       5215.05140  |   11:39:49:039   |  11:39:54:254    |   16606007932 |
-------------------------------------------------------------------------------------------------------------------------
|       0   |       0.00000  |      5062.50000  |       5070.21770  |   11:39:54:254   |  11:39:59:325    |   16160869921 |
-------------------------------------------------------------------------------------------------------------------------
|       1   |       0.00000  |      5593.75000  |       5578.51280  |   11:39:59:325   |  11:40:04:903    |   17781008041 |
-------------------------------------------------------------------------------------------------------------------------
|       2   |       0.00000  |      5312.50000  |       5318.31100  |   11:40:04:903   |  11:40:10:221    |   16957883446 |
-------------------------------------------------------------------------------------------------------------------------
|      15   |       0.00000  |      5250.00000  |       5236.11630  |   11:40:10:221   |  11:40:15:458    |   16708097311 |
-------------------------------------------------------------------------------------------------------------------------
```