In this laboratory assignment you will learn methods for controlling thread scheduling by setting thread priorities. You will study the effects of the thread priority levels on the relative run times of competing threads in a multi-threaded process.

When a thread is created in a multi-threaded process it is given a normal priority level by default. Sometimes we would like to give one thread a higher or lower priority than another. C# supports five programmer-defined priority levels (Highest, AboveNormal, Normal, BelowNormal, and Lowest). The priority level of a thread can be set any time after the thread has been created as shown below.

```csharp
Threads myThread = new Threads("T");
myThread.thread.Priority = ThreadPriority.AboveNormal;
```

ThreadPriority is an enumeration defined in the System.Threading namespace. The scheduling of threads with different priorities is controlled in a multi-level feedback queue. Threads at the highest priority level will share the CPU (through time slicing) with other threads of the same priority until they are done or they are removed from the ready list by some event or state transition (e.g. Sleep, Suspend, Terminate, or Wait) Only when no higher priority threads are ready for the CPU will a lower priority thread receive a time quantum. For example, threads 0,1 and 2 above will take turns until they complete or transition to a non-ready state. When there are no threads in the Highest priority round-robin queue, threads 3 and 4 will get a turn with the CPU. This hierarchical scheduling proceeds from the Highest to the Lowest priority threads.

We have already learned that putting a thread to sleep for 0 time units is equivalent to placing it at the back of the queue. This is also true for priority scheduling except that the thread is placed at the back of the queue that matches the thread's priority setting. At first it would seem that this scheduling scheme would result in a completely predictable ordering of thread executions. However, due to the fact that the operating system may be running other (kernel or application) processes, thread scheduling can be somewhat unpredictable.

Due to the overhead created by priority scheduling in a multithreaded application, the effective use of the CPU can actually go down as the number of concurrently executing processes increases. The chart on the right shows the CPU Usage History for a single processor PC. In this example, when a process intensive application is running, CPU usage is at 100%. This is compared with a interactive graphics application that consumes only about 50% of the CPU.
While the graphics program was still running, a multi-threaded application using priority scheduling was launched. As shown in the chart, with both programs running the CPU usage dropped to around 17%. In addition, the efficiency of the interactive graphics program dropped significantly (from 20 fps down to 3 fps) while the multi-threaded application was running.

In order to make good design decisions, software developers need a detailed understanding of the behavior of thread scheduling in multi-threaded programs. We will build a multi-threaded application to experiment with priority scheduling using the source code provided below.

```csharp
//Thread Priority Scheduling
using System;
using System.Threading;
class Threads
{
    public static int total = 0;
    public int count;
    public Thread thread;

    public Threads(string name)
    {
        count = 0;
        thread = new Thread(new ThreadStart(this.run));
        thread.Name = name;
        thread.Start();
    }

    void run()
    {
        Console.WriteLine(thread.Name + " starting at =" + total);
        do
        {
            Thread.Sleep(0); // vary the sleep delay time e.g. 0,1,10,100
            Console.WriteLine(thread.Name + "=" + count + " total=" + total);
            count++;
            total++;
        } while (count < 40); // vary max count to change simulation run time
        Console.WriteLine(thread.Name + " terminating at = " + total);
    }
}
class CreateMoreThreads
{
    public static void Main()
    {
        Console.WriteLine("Main thread starting.");

        Threads myThread1 = new Threads("T1");
        Threads myThread2 = new Threads("T2");
        Threads myThread3 = new Threads("T3");
        Threads myThread4 = new Threads("T4");
        Threads myThread5 = new Threads("T5");
    }
}
myThread1.thread.Priority = ThreadPriority.BelowNormal;
myThread2.thread.Priority = ThreadPriority.AboveNormal;
myThread3.thread.Priority = ThreadPriority.Normal;
myThread4.thread.Priority = ThreadPriority.Lowest;
myThread5.thread.Priority = ThreadPriority.Highest;

myThread1.thread.Join();
myThread2.thread.Join();
myThread3.thread.Join();
myThread4.thread.Join();
myThread5.thread.Join();

Console.WriteLine("Main thread ending.");
Console.Read();
}
}

Step 1: Build a Console Application using the source code above. This program creates five threads and gives each of them a different priority from Highest to Lowest. Each thread performs the same operation, which is to accumulate a count value (currently set to 40). A total count is also accumulated in a global integer named total. For each pass through the while-loop the threads are put to sleep for some period of time. Initially the time is set to 0 time units (milliseconds).

Each time a thread gets a turn at the CPU it will display a message in the console window, increment the counters and return to its respective round-robin queue. Implement and test your program, to verify its proper operation before proceeding.

Question 1: Does the operation of this program follow the multi-level feedback queue scheduling algorithm described above and in the textbook? ____________ Discuss your observations of the output to support your answer.

______________________________________________________________________________
______________________________________________________________________________

Step 2: Run the program with the sleep time set to Sleep(1) rather than Sleep(0).

Question 2: Does the program still follow the scheduling algorithm? ____________ Discuss your observations.

______________________________________________________________________________
______________________________________________________________________________

Question 3: Do the threads execute in the same order each time the program is run? _________

Why or why not?

______________________________________________________________________________
______________________________________________________________________________
**Step 3:** Run the program with the sleep time set to Sleep(10), and then Sleep(100).

**Question 4:** Describe the behavior of the program as the amount of sleep time is increased.

______________________________________________________________________________
______________________________________________________________________________
______________________________________________________________________________

**Question 5:** Explain this effect.

______________________________________________________________________________
______________________________________________________________________________
______________________________________________________________________________

**Step 4:** Modify the program to run for a longer time. This can be done by setting the maximum count to some large value (e.g. 10000000) or by replacing `while(count < 40)` with `while(true)`. Comment out the `Console.WriteLine()` command and run the program for various sleep times. **Caution:** You may not want to run the program with `while(true)` and `Sleep(0)`. Open the Windows Task Manager (i.e. press <CTRL> <ALT> <DEL>) and monitor the CPU Usage under the **Performance** tab.

**Question 6:** With the `Console.WriteLine()` commented out....

   a. What is the CPU usage level for Sleep(0)? __________

   b. What is the CPU usage level for Sleep(1)? __________

   c. What is the CPU usage level for Sleep(100)? __________

**Question 7:** Explain the difference in CPU usage level between 6.a and 6.b above.

______________________________________________________________________________
______________________________________________________________________________
______________________________________________________________________________

**Question 8:** Explain the difference in CPU usage level between 6.b and 6.c above.

______________________________________________________________________________
______________________________________________________________________________
______________________________________________________________________________
The chart below shows the progress of the five threads for a typical run. The Loop Count is the value of the count variable for each thread, while the process time is the cumulative count for all threads (i.e. total). Compare these results with your own experience running the program as you answer the following questions.

**Question 9:** What is the actual difference in scheduling priority between $Highest$ and $AboveNormal$ priorities?

______________________________________________________________________________
______________________________________________________________________________

**Question 10:** What is the difference in rate of completion between the $Highest$ priority thread and the $Normal$ priority thread?

______________________________________________________________________________
______________________________________________________________________________

**Question 11:** What happens to the rate of completion of the $Normal$ priority thread after the higher priority threads have completed?

______________________________________________________________________________
______________________________________________________________________________

**Question 12:** Under what conditions in your experiments did a thread with $BelowNormal$ or $Lowest$ priority start execution before higher priority threads had completed? Discuss your answer.

______________________________________________________________________________
______________________________________________________________________________