**Assignment 7 (Problem 1.2)**

**Rasika Mohod**

rmohod@gmu.edu

G01044774

b)

The code logic here is written to perform the task of finding the nth Fibonacci series number and not the Factorial of a given number. This task is performed with different threads and also the time taken by each thread from its start to its completion is calculated.

The number of threads to be used is taken as input from user and also the number, n: the term to be found in Fibonacci series, is obtained as input from user.

The code creates an array of threads by using the number of threads as given input by user. Each thread is set with its priority among all the threads. The priority of each thread is calculated by using a mod function, t[i].setPriority((i % 10) + 1). Hence the threads are given priority within the range of 1-10. The priority is decided according to the position of the thread in the threads array. The thread at position/index 1 and index 11 will have same priority as 2.

The class ThreadTest creates an object of Test whose constructor initializes the variables. The run() method of Test class gives the output as the timestamp when each thread has started its execution and when the thread ends. This method also provides the total time in milliseconds taken by that particular thread to complete the task.

Lastly the ThreadTest class calls join() method on each thread to ensure all threads that started from main must end in order in which they started and also that main should end in last.

c)

**Output Analysis of different runtime executions:**

On the basis of the priority, the CPU gives time to the threads. Higher the priority, faster the threads are executed. A thread of **higher priority** **preempts** a thread of lower priority. As we look at the outputs, though it may appear random at the start, a pattern that is observed though is that, the threads with higher values are getting executed prior to threads having lower priority.

With less values of threads, the processor hardly takes any time to compute the logic. Refer output 1.

But with larger no. of threads, we do observe a pattern that higher priority threads getting started as well as executed first. One this is definite, that the thread that executes the first, has a priority more than the last thread to get executed.

From the study of the highlighted cases in output 2 – Execution 1, we realize that if a thread with higher priority started after the thread with lower priority it finishes before, even though started later.

a)

**Output 1:**

Input Number of Threads = 2

Input factorial (n) = 3

a) **Execution 1:**

Input Number of Threads = 2

Input factorial (n) = 3

Starting task Task 1 at 11: 16: 45: 312

Starting task Task 0 at 11: 16: 45: 312

Ending task Task 1 at 11: 16: 45: 312 after 0 milliseconds

Ending task Task 0 at 11: 16: 45: 312 after 0 milliseconds

**Execution 2:**

Input Number of Threads = 2

Input factorial (n) = 3

Starting task Task 1 at 12: 10: 33: 675

Starting task Task 0 at 12: 10: 33: 675

Ending task Task 1 at 12: 10: 33: 675 after 0 milliseconds

Ending task Task 0 at 12: 10: 33: 675 after 0 milliseconds

**Execution 3:**

Input Number of Threads = 2

Input factorial (n) = 3

Starting task Task 0 at 12: 19: 07: 080

Starting task Task 1 at 12: 19: 07: 080

Ending task Task 0 at 12: 19: 07: 080 after 0 milliseconds

Ending task Task 1 at 12: 19: 07: 080 after 0 milliseconds

b)

Here as we can see the number of threads used are 2, thus the computation does not take much time as there are enough cores to be allocated to each thread for its execution.

The three different runtime executions of same input data are different form each other. The third execution starts the thread 0 first and then thread 1 is started. The same order is maintained while ending the thread. However, by our priority principle, the task for thread 0 is performed before the task of thread 1.

c)

It can be observed from three different executions that with only two threads the number of tasks do not take much time to complete also there is no racing for the resources.

**Output 2:**

Input Number of Threads = 15

Input factorial (n) = 5

a) **Execution 1:**

Input Number of Threads = 15

Input factorial (n) = 5

Starting task Task 7 at 14: 00: 43: 953

Ending task Task 7 at 14: 00: 43: 953 after 0 milliseconds

Starting task Task 2 at 14: 00: 43: 953

Ending task Task 2 at 14: 00: 43: 953 after 0 milliseconds

Starting task Task 11 at 14: 00: 43: 953

Starting task Task 13 at 14: 00: 43: 953

Starting task Task 10 at 14: 00: 43: 953

Ending task Task 13 at 14: 00: 43: 953 after 0 milliseconds

Starting task Task 1 at 14: 00: 43: 953

Starting task Task 3 at 14: 00: 43: 953

Starting task Task 12 at 14: 00: 43: 953

Ending task Task 3 at 14: 00: 43: 953 after 0 milliseconds

Starting task Task 4 at 14: 00: 43: 953

Starting task Task 14 at 14: 00: 43: 953

Ending task Task 4 at 14: 00: 43: 953 after 0 milliseconds

Starting task Task 6 at 14: 00: 43: 953

Starting task Task 8 at 14: 00: 43: 953

Ending task Task 6 at 14: 00: 43: 953 after 0 milliseconds

Starting task Task 9 at 14: 00: 43: 953

Ending task Task 8 at 14: 00: 43: 953 after 0 milliseconds

Ending task Task 9 at 14: 00: 43: 953 after 0 milliseconds

Ending task Task 14 at 14: 00: 43: 953 after 0 milliseconds

Ending task Task 1 at 14: 00: 43: 953 after 0 milliseconds

Ending task Task 12 at 14: 00: 43: 953 after 0 milliseconds

Ending task Task 10 at 14: 00: 43: 953 after 0 milliseconds

Ending task Task 11 at 14: 00: 43: 953 after 0 milliseconds

Starting task Task 0 at 14: 00: 43: 953

Starting task Task 5 at 14: 00: 43: 953

Ending task Task 5 at 14: 00: 43: 968 after 15 milliseconds

Ending task Task 0 at 14: 00: 43: 968 after 15 milliseconds

**Execution 2:**

Input Number of Threads = 15

Input factorial (n) = 5

Starting task Task 9 at 12: 13: 46: 004

Starting task Task 3 at 12: 13: 46: 004

Starting task Task 4 at 12: 13: 46: 004

Ending task Task 3 at 12: 13: 46: 004 after 0 milliseconds

Starting task Task 14 at 12: 13: 46: 004

Starting task Task 12 at 12: 13: 46: 004

Starting task Task 6 at 12: 13: 46: 004

Starting task Task 8 at 12: 13: 46: 004

Ending task Task 6 at 12: 13: 46: 019 after 15 milliseconds

Ending task Task 8 at 12: 13: 46: 019 after 15 milliseconds

Starting task Task 5 at 12: 13: 46: 004

Ending task Task 12 at 12: 13: 46: 019 after 15 milliseconds

Starting task Task 13 at 12: 13: 46: 004

Starting task Task 7 at 12: 13: 46: 004

Starting task Task 0 at 12: 13: 46: 004

Starting task Task 10 at 12: 13: 46: 004

Ending task Task 13 at 12: 13: 46: 019 after 15 milliseconds

Ending task Task 5 at 12: 13: 46: 019 after 15 milliseconds

Ending task Task 14 at 12: 13: 46: 004 after 0 milliseconds

Ending task Task 4 at 12: 13: 46: 004 after 0 milliseconds

Starting task Task 11 at 12: 13: 46: 004

Ending task Task 9 at 12: 13: 46: 004 after 0 milliseconds

Ending task Task 11 at 12: 13: 46: 019 after 15 milliseconds

Starting task Task 1 at 12: 13: 46: 004

Starting task Task 2 at 12: 13: 46: 004

Ending task Task 10 at 12: 13: 46: 019 after 15 milliseconds

Ending task Task 0 at 12: 13: 46: 019 after 15 milliseconds

Ending task Task 7 at 12: 13: 46: 019 after 15 milliseconds

Ending task Task 2 at 12: 13: 46: 019 after 15 milliseconds

Ending task Task 1 at 12: 13: 46: 019 after 15 milliseconds

**Execution 3:**

Input Number of Threads = 15

Input factorial (n) = 5

Starting task Task 14 at 12: 20: 35: 941

Starting task Task 3 at 12: 20: 35: 941

Ending task Task 14 at 12: 20: 35: 941 after 0 milliseconds

Starting task Task 2 at 12: 20: 35: 941

Starting task Task 5 at 12: 20: 35: 941

Starting task Task 13 at 12: 20: 35: 941

Ending task Task 5 at 12: 20: 35: 941 after 0 milliseconds

Starting task Task 12 at 12: 20: 35: 941

Starting task Task 8 at 12: 20: 35: 941

Ending task Task 12 at 12: 20: 35: 941 after 0 milliseconds

Ending task Task 13 at 12: 20: 35: 941 after 0 milliseconds

Ending task Task 2 at 12: 20: 35: 941 after 0 milliseconds

Starting task Task 4 at 12: 20: 35: 941

Starting task Task 6 at 12: 20: 35: 941

Ending task Task 4 at 12: 20: 35: 941 after 0 milliseconds

Starting task Task 7 at 12: 20: 35: 941

Ending task Task 7 at 12: 20: 35: 941 after 0 milliseconds

Starting task Task 11 at 12: 20: 35: 941

Starting task Task 9 at 12: 20: 35: 941

Ending task Task 11 at 12: 20: 35: 941 after 0 milliseconds

Ending task Task 9 at 12: 20: 35: 941 after 0 milliseconds

Starting task Task 10 at 12: 20: 35: 941

Ending task Task 3 at 12: 20: 35: 941 after 0 milliseconds

Ending task Task 10 at 12: 20: 35: 941 after 0 milliseconds

Starting task Task 1 at 12: 20: 35: 941

Ending task Task 6 at 12: 20: 35: 941 after 0 milliseconds

Starting task Task 0 at 12: 20: 35: 941

Ending task Task 8 at 12: 20: 35: 941 after 0 milliseconds

Ending task Task 0 at 12: 20: 35: 941 after 0 milliseconds

Ending task Task 1 at 12: 20: 35: 941 after 0 milliseconds

**Output 3:**

Input Number of Threads = 8

Input factorial (n) = 2

a) **Execution 1:**

Input Number of Threads = 8

Input factorial (n) = 2

Starting task Task 7 at 11: 19: 19: 061

Ending task Task 7 at 11: 19: 19: 077 after 16 milliseconds

Starting task Task 5 at 11: 19: 19: 079

Starting task Task 4 at 11: 19: 19: 079

Ending task Task 5 at 11: 19: 19: 079 after 0 milliseconds

Ending task Task 4 at 11: 19: 19: 080 after 1 milliseconds

Starting task Task 3 at 11: 19: 19: 079

Starting task Task 6 at 11: 19: 19: 080

Ending task Task 3 at 11: 19: 19: 080 after 1 milliseconds

Ending task Task 6 at 11: 19: 19: 081 after 1 milliseconds

Starting task Task 2 at 11: 19: 19: 082

Starting task Task 1 at 11: 19: 19: 082

Ending task Task 2 at 11: 19: 19: 082 after 0 milliseconds

Starting task Task 0 at 11: 19: 19: 082

Ending task Task 1 at 11: 19: 19: 082 after 0 milliseconds

Ending task Task 0 at 11: 19: 19: 082 after 0 milliseconds

**Execution 2:**

Input Number of Threads = 8

Input factorial (n) = 2

Starting task Task 6 at 12: 17: 10: 150

Starting task Task 1 at 12: 17: 10: 166

Starting task Task 3 at 12: 17: 10: 150

Ending task Task 6 at 12: 17: 10: 166 after 16 milliseconds

Ending task Task 3 at 12: 17: 10: 166 after 16 milliseconds

Starting task Task 0 at 12: 17: 10: 150

Starting task Task 2 at 12: 17: 10: 150

Starting task Task 5 at 12: 17: 10: 150

Starting task Task 7 at 12: 17: 10: 150

Starting task Task 4 at 12: 17: 10: 150

Ending task Task 7 at 12: 17: 10: 166 after 16 milliseconds

Ending task Task 5 at 12: 17: 10: 166 after 16 milliseconds

Ending task Task 2 at 12: 17: 10: 166 after 16 milliseconds

Ending task Task 0 at 12: 17: 10: 166 after 16 milliseconds

Ending task Task 1 at 12: 17: 10: 166 after 0 milliseconds

Ending task Task 4 at 12: 17: 10: 166 after 16 milliseconds

**Execution 3:**

Input Number of Threads = 8

Input factorial (n) = 2

Starting task Task 4 at 12: 20: 09: 185

Starting task Task 5 at 12: 20: 09: 185

Starting task Task 6 at 12: 20: 09: 185

Starting task Task 7 at 12: 20: 09: 185

Ending task Task 6 at 12: 20: 09: 185 after 0 milliseconds

Ending task Task 7 at 12: 20: 09: 185 after 0 milliseconds

Ending task Task 4 at 12: 20: 09: 185 after 0 milliseconds

Ending task Task 5 at 12: 20: 09: 185 after 0 milliseconds

Starting task Task 3 at 12: 20: 09: 185

Starting task Task 2 at 12: 20: 09: 185

Ending task Task 3 at 12: 20: 09: 185 after 0 milliseconds

Ending task Task 2 at 12: 20: 09: 185 after 0 milliseconds

Starting task Task 1 at 12: 20: 09: 185

Starting task Task 0 at 12: 20: 09: 185

Ending task Task 1 at 12: 20: 09: 185 after 0 milliseconds

Ending task Task 0 at 12: 20: 09: 185 after 0 milliseconds