Design Document Machine Problem 5:
Kernel-Level Thread Scheduling

Operating Systems : MP 5 **Rajendra Thottempudi**
UIN : 933004901

Design :

In this Machine Problem, we are basically asked to implement a scheduler to manage multiple threads with the help of a ready queue. The first task of this particular machine problem is to build a first in first out (FIFO) queue to handle the threads in a sequential manner without worrying about the termination condition. The second part of the problem is to handle how to terminate a thread using the terminate function of the scheduler.

FIFO Queue :

I have used a queue data structure to store the threads in a sequence. These threads will be processed one after the other by the scheduler. Each entry in the queue will be a structure node which contains a thread as well as some additional information related to the node. It will also contain the details of the next node that follows it.

Different method of the readyQueue class are :

1. **Enqueue/Insert** : This is the primary method where an entry is inserted into the current queue. We just update the end pointer of the queue to this new element and update the mapping of the previous end node to point to this new end node as well. This way the queue is updated. In case if the queue is empty, we just update the first and last pointers of the queue to point to this newly created node and return it.

2. **Pop/getFirst** : This method is added to the queue class to return the first thread in the queue that needs to be run by the scheduler.

3. **Pop** : This method is added to the queue to remove the topmost element from the queue. This is used in combination with the getFirst method where a scheduler once it receives the thread that needs to be run - removes the top most element from the queue by moving the first pointer to point to the next node.

4. **deleteThisNode** : This method is added to provide a functionality where we can remove a thread from a queue whenever a situation arises for the scheduler. In this method we iterate through the elements of the queue and remove the elements containing the corresponding thread by updating the pointers.

Scheduler Class :

After the implementation of the queue, I used the scheduler class methods and implemented them in the following manager :

1. **Yield** : In the yield method, the idea is that when this method of the scheduler is invoked, we have to make the next thread ready to be run by the CPU. In order to know which thread to run next, we go to our queue and get the top element of it. We then send this top element to the dispatcher to make the context switch that is required for the CPU.

2. **Resume** : The resume method of the scheduler is called when we want to insert a particular thread back into the queue and resume running it some time later as per the queue elements. So, in this method we take the thread and we send it to be inserted into the queue.

3. **Add** : This method is similar to the resume method, except that this is called when we have a new thread to be run by the CPU i.e. when a process wants to add a new thread to the queue in the scheduler. We simply add this thread by inserting it into the queue.

4. **Terminate** : This method is called whenever a thread is meant to be terminated i.e. the scheduler should not run it unless explicitly added again to the scheduler. We do this by removing the thread from our queue by calling the terminate function of the queue.

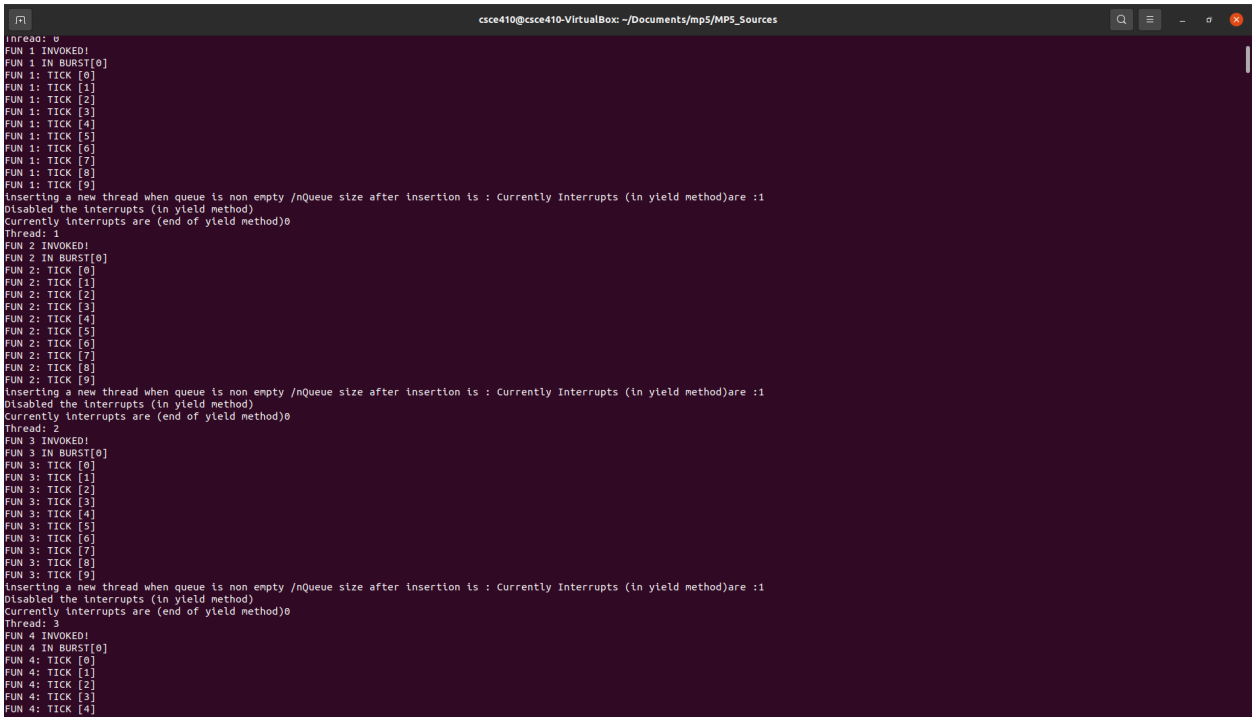
Also, in order to properly terminate a thread we have to release the memory attached to it and delete the thread whenever it returns from the function. This is done by updating the thread_shutdown method in the thread.c file. We implement this by first calling the terminate method of the scheduler to remove any instances of the thread in the queue. We then delete the thread and make the next element in the queue ready by calling the yield method of the scheduler which also does the context switch for the next thread to be run by the CPU.

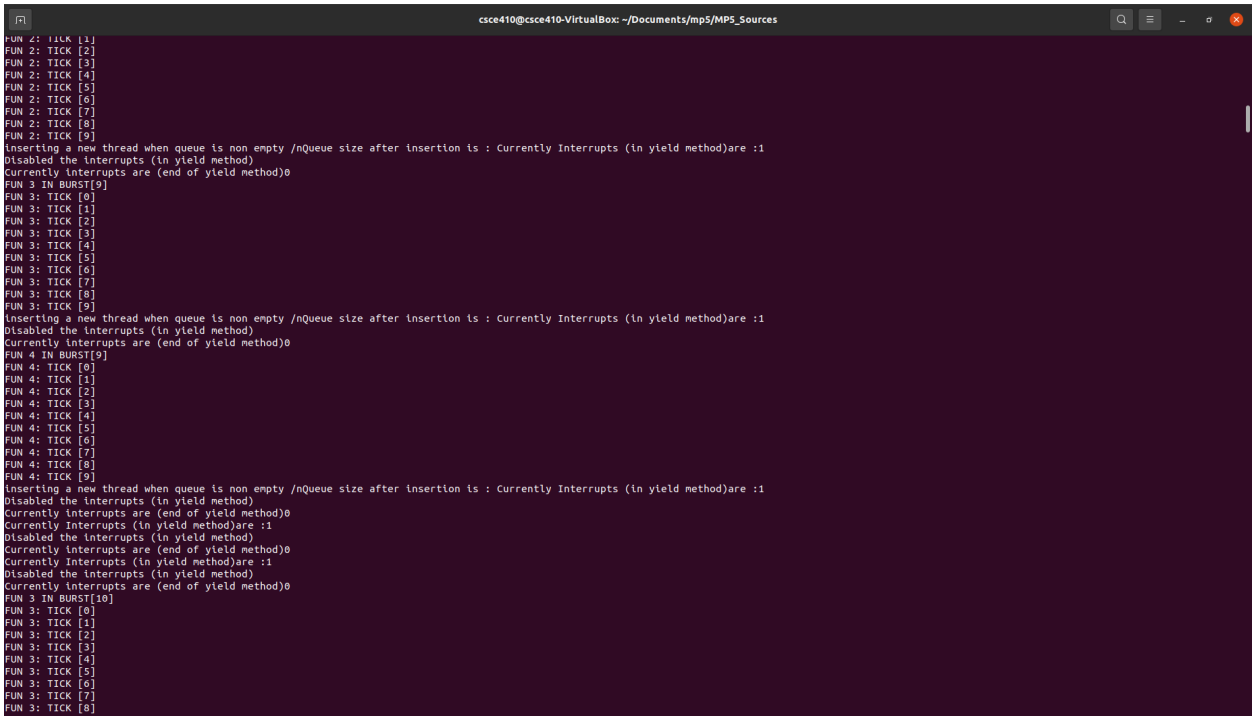
Bonus Points : Correct Handling of the Interrupts :

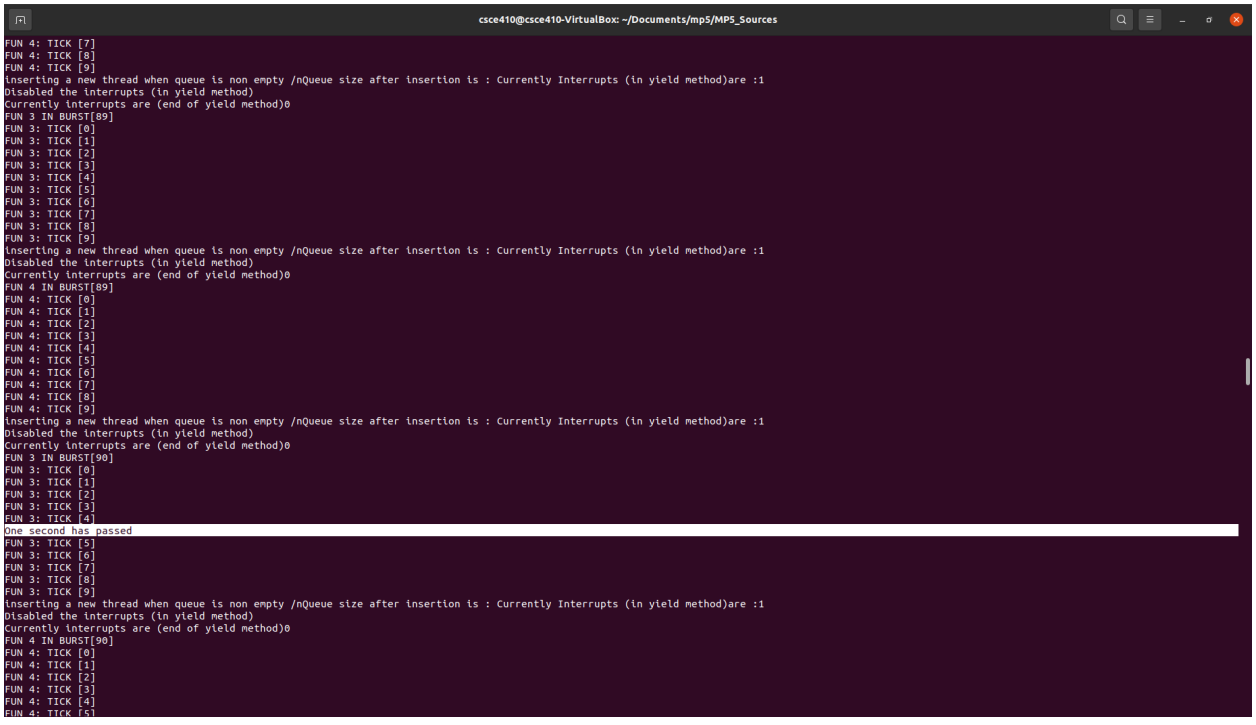
Initially the interrupts have been disabled while the threads are running and we will not see the “One second has passed” message in the terminal because of that. In order to re-enable the interrupts, we have to ensure that we adhere to the principle of mutual exclusion. As seen from the machine.H file, we have the enable and disable methods which when called enable or disable the interrupts. In order to ensure that mutual exclusion is followed, I disabled the interrupts before performing any operation on the queue in the scheduler methods and re-enabled them after the action had been taken care of i.e. after the operations on the queue and before leaving from the scheduler method. In addition, for enabling the interrupts we add the enable interrupts method in the thread_start method of the thread class.

Please find the screenshots of the various scenarios present below :

1. Showing the four threads being executed
2. Showing only the third and fourth threads being executed after the termination of first two threads
3. The message : “One second has passed” indicating the proper execution of interrupt handler.




csce410@csce410-VirtualBox: ~/Documents/mp5/MP5_Sources
FUN 2: TICK [1]
FUN 2: TICK [2]
FUN 2: TICK [3]
FUN 2: TICK [4]
FUN 2: TICK [5]
FUN 2: TICK [6]
FUN 2: TICK [7]
FUN 2: TICK [8]
FUN 2: TICK [9]
inserting a new thread when queue is non empty /nQueue size after insertion is : Currently Interrupts (in yield method)are :1
Disabled the interrupts (in yield method)
Currently interrupts are (end of yield method)0
FUN 3 IN BURST[9]
FUN 3: TICK [0]
FUN 3: TICK [1]
FUN 3: TICK [2]
FUN 3: TICK [3]
FUN 3: TICK [4]
FUN 3: TICK [5]
FUN 3: TICK [6]
FUN 3: TICK [7]
FUN 3: TICK [8]
FUN 3: TICK [9]
inserting a new thread when queue is non empty /nQueue size after insertion is : Currently Interrupts (in yield method)are :1
Disabled the interrupts (in yield method)
Currently interrupts are (end of yield method)0
FUN 4 IN BURST[9]
FUN 4: TICK [0]
FUN 4: TICK [1]
FUN 4: TICK [2]
FUN 4: TICK [3]
FUN 4: TICK [4]
FUN 4: TICK [5]
FUN 4: TICK [6]
FUN 4: TICK [7]
FUN 4: TICK [8]
FUN 4: TICK [9]
inserting a new thread when queue is non empty /nQueue size after insertion is : Currently Interrupts (in yield method)are :1
Disabled the interrupts (in yield method)
Currently interrupts are (end of yield method)0
Currently Interrupts (in yield method)are :1
Disabled the interrupts (in yield method)
Currently interrupts are (end of yield method)0
Currently Interrupts (in yield method)are :1
Disabled the interrupts (in yield method)
Currently interrupts are (end of yield method)0
FUN 3 IN BURST[10]
FUN 3: TICK [0]
FUN 3: TICK [1]
FUN 3: TICK [2]
FUN 3: TICK [3]
FUN 3: TICK [4]
FUN 3: TICK [5]
FUN 3: TICK [6]
FUN 3: TICK [7]
FUN 3: TICK [8]


csce410@csce410-VirtualBox: ~/Documents/mp5/MP5_Sources
FUN 4: TICK [7]
FUN 4: TICK [8]
FUN 4: TICK [9]
inserting a new thread when queue is non empty /nQueue size after insertion is : Currently Interrupts (in yield method)are :1
Disabled the interrupts (in yield method)
Currently interrupts are (end of yield method)0
FUN 3 IN BURST[89]
FUN 3: TICK [0]
FUN 3: TICK [1]
FUN 3: TICK [2]
FUN 3: TICK [3]
FUN 3: TICK [4]
FUN 3: TICK [5]
FUN 3: TICK [6]
FUN 3: TICK [7]
FUN 3: TICK [8]
FUN 3: TICK [9]
inserting a new thread when queue is non empty /nQueue size after insertion is : Currently Interrupts (in yield method)are :1
Disabled the interrupts (in yield method)
Currently interrupts are (end of yield method)0
FUN 4 IN BURST[89]
FUN 4: TICK [0]
FUN 4: TICK [1]
FUN 4: TICK [2]
FUN 4: TICK [3]
FUN 4: TICK [4]
FUN 4: TICK [5]
FUN 4: TICK [6]
FUN 4: TICK [7]
FUN 4: TICK [8]
FUN 4: TICK [9]
inserting a new thread when queue is non empty /nQueue size after insertion is : Currently Interrupts (in yield method)are :1
Disabled the interrupts (in yield method)
Currently interrupts are (end of yield method)0
FUN 3 IN BURST[90]
FUN 3: TICK [0]
FUN 3: TICK [1]
FUN 3: TICK [2]
FUN 3: TICK [3]
FUN 3: TICK [4]
One second has passed
FUN 3: TICK [5]
FUN 3: TICK [6]
FUN 3: TICK [7]
FUN 3: TICK [8]
FUN 3: TICK [9]
inserting a new thread when queue is non empty /nQueue size after insertion is : Currently Interrupts (in yield method)are :1
Disabled the interrupts (in yield method)
Currently interrupts are (end of yield method)0
FUN 4 IN BURST[90]
FUN 4: TICK [0]
FUN 4: TICK [1]
FUN 4: TICK [2]
FUN 4: TICK [3]
FUN 4: TICK [4]