Design Document MP5 Operating Systems CSCE 611

Chinmay Ajit Sawkar

Implementation

FIFO Scheduler:

Implemented a ready queue without using the new keyword with help of linked list in the class Queue. The class Queue is defined in the scheduler.H file. In this I define constructors to create a queue object initially without thread and also another constructor to initialize a queue with a thread. I define two variables: a thread object th and Queue object q_nxt to define the next element in the queue.

There are three functions of the queue:

1. insert_thread(Thread * _thread): This is used to insert a thread into the queue. This works by checking if the th object is null, if not null it inserts the new thread by calling the insert_thread function recursively. Else if the thread object th is null I directly assign the thread object as the input thread.
2. pop_thread() : This function is used to output the top most thread in the ready queue. If the thread object th is Null the function returns Null. The value of the th is stored in a variable to return. Next I check if q_nxt is Null, then assign th as Null, else the values of th and q_nxt of the next thread are stored in the current thread and the next queue object is deleted.
3. Remove_thread - this function is used to terminate the thread. In this we go through the queue to find the thread to be terminated. For this I compare each element in the ready queue to the thread to terminate, if it's not the thread to terminate I put it back in the ready queue in the same order, else the thread to terminate is not added to the ready queue.

Scheduler Class Implementation

In this I have implemented in the scheduler.C file where in the constructor I have defined the ready_queue length as 0 initially. In this class the FIFO scheduler is implemented by following functions. In this class the termination of the threads is also implemented.

1. yield()- Used to yield the CPU from the current thread and dispatch the next thread in the ready queue. If the ready queue is empty, a proper message is output.
2. resume() - This is used to resume the execution of a thread after preemption/yielding the CPU and done by adding it to the end of the ready queue.
3. add()- this is done to add the thread to the ready queue initially and has same implementation as the resume()
4. terminate() - for this I call the remove thread() function of the Queue class. After terminating the yield function is called to dispatch to the next thread.

The termination of threads is done by defining the macro TERMINATING FUNCTIONS which terminates the thread 1 and 2 (fun1, fun2) after 10 interactions. The implementation result is given below:

```
FUN 3: TICK [7]
FUN 3: TICK [8]
FUN 3: TICK [9]
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]
A thread is terminated
A thread is terminated
FUN 4 IN BURST[10]
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]
FUN 3 IN BURST[10]
FUN 3: TICK [0]
```

Bonus: Handling interrupts

For handling interrupts we enable machine interrupts in the Thread.C file. In the scheduler.C in all the functions where the queue operations are performed the interrupts are distabled before that and again enabled after doing the queue operations.

The working of interrupts is confirmed from the console log where the ‘on second has passed’ message.