CSCE 611

OPERATING SYSTEMS

MP 5

Kernel-Level Thread Scheduling

Student: K Ram Sankar

The objective of this MP5 is to implement the scheduling of kernel level threads. I have also implemented option 1 (i.e) correct handling of interrupts

The scheduler is built using a queue data structure and is defined as ready_q

The queue data structure is implemented as a linked list of queue nodes

The queueNode contains a thread pointer and a pointer to the next queueNode

```
Struct queueNode

{

        Thread * t;

        queueNode* next;

}
```

The queue class contains size , head and tail queueNode pointers

A head and tail queueNode pointers are used for the insertion (push) and deletion (pop) of the qeueNodes and follows FIFO order (First in First out)

Scheduler APIs

Scheduler::add(Thread * thread)

The add API inserts the thread into the ready queue

Here the push function of the queue data structure is used

This API is called initially when all the threads are pushed into ready queue.

Scheduler::resume(Thread * thread)

The resume API has the same functionality as add but it is used when the thread is dispatched and ready to execute again.

Scheduler::yield()

This function deques the next thread in the ready queue and passes the thread object to the dispatcher for execution

Here the pop function of the queue data structure is used

Scheduler::terminate(Thread * thread)

This function removes the thread from the ready queue by iteratively popping and pushing to queue until the thread id is matched with the given thread

Later the stack memory of the thread is freed, and the control is passed to the next thread by using the yield function

Thread APIs

thread_start()

Here just interrupts are enabled using enable_interrupts function declared in Machine.c

Thread_shutdown()

Here the current thread is identified using CurrentThread() API and the responsibility of the terminating the thread is given to scheduler. The scheduler terminate function defined above explains the thread termination in detail

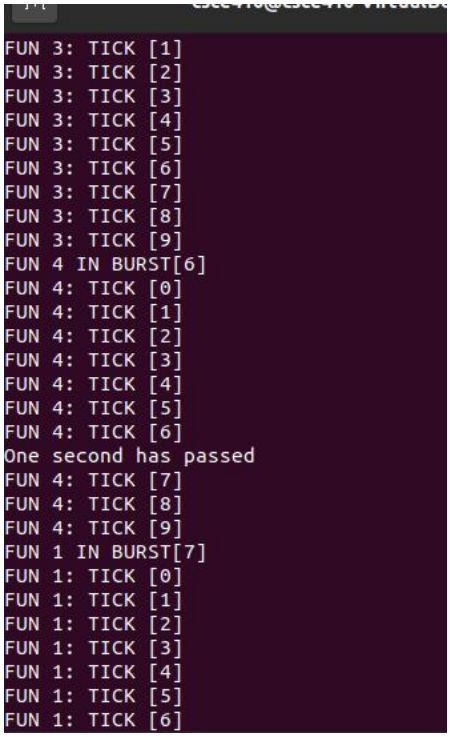
OPTION -1 IMPLEMENTATION

Handling of the interrupts

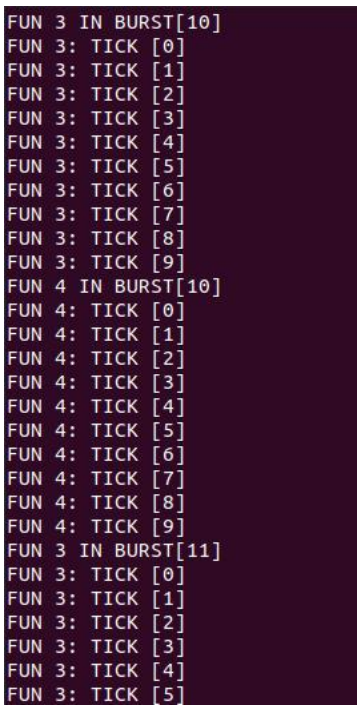
For this, I have defined two functions check_and_enable_interrupts, check_and_disable_interrupts which are used in the scheduler functions when we perform operations related to the ready queue because it is the shared data structure of the threads

Both check_and_enable_interrupts() and check_and_disable_interrupts() using the enable and disable interrupts functions defined in the machine.C

RESULTS



The interrupt functionality is working fine as we could see the message of the interrupt handler



Thread 3 and Thread 4 run infinitely. Thread 1 and Thread 2 get terminated after 10 burst