MP5: Threading

Joshua Clapp

UIN: 930004089

CSCE410: Operating System

Assigned Tasks

Main: Completed.

Bonus Option 1: Interrupts

Completed

Bonus Option 2: Round Robin

Completed

Bonus Option 3:

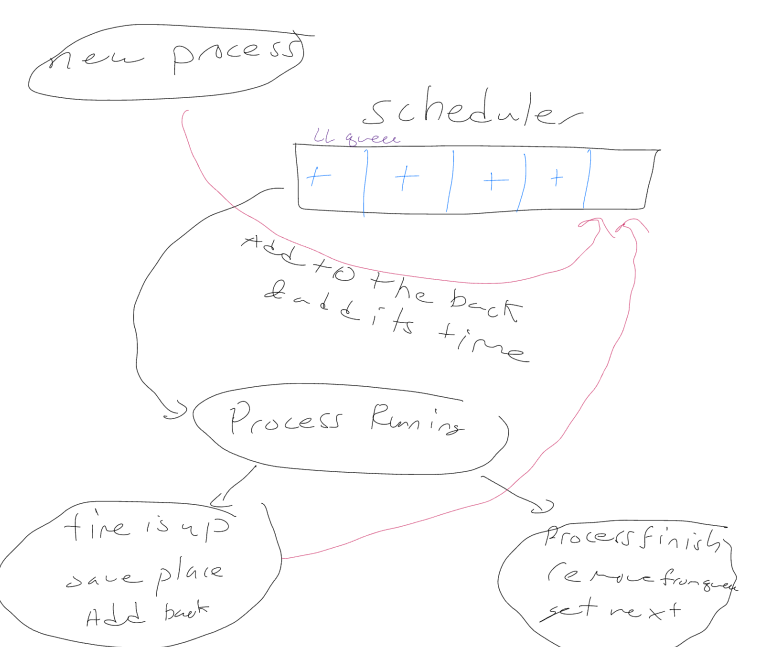
none.

Bonus Option 4:

none.

System Design

* Take a process operation and create a threading Round Robin context switch that can properly work with interrupts.
  + This process is defined to switch between threads every 50 msec.

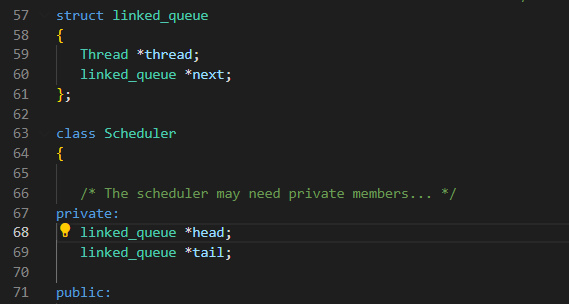


Code Description

I changed Thread.C, and wrote the Schedule.C to compile the code type make, ./copykernel.sh and then run bochs.

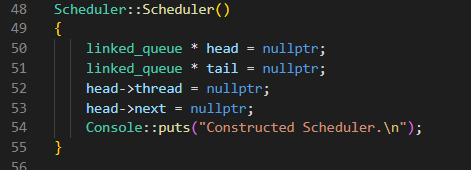
Schedule.H:Private Data Members:

This is a linked list that contains the item thread. To improve the resume and yield cases I have used a doubly linked list.



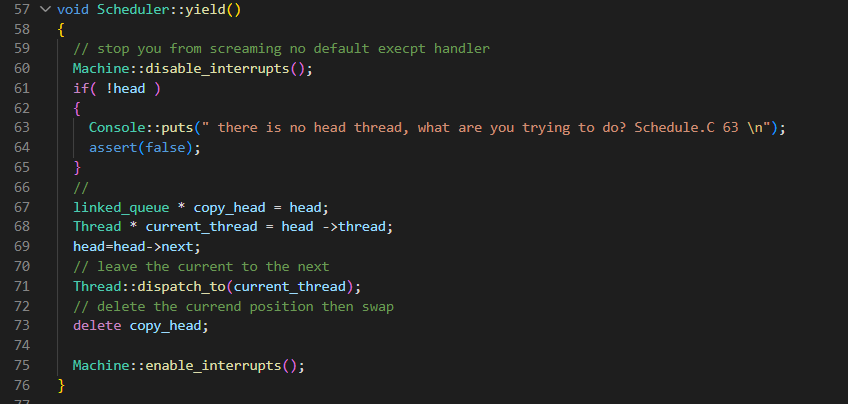
Schedule.C:Schedule():

This initializes our linked list.



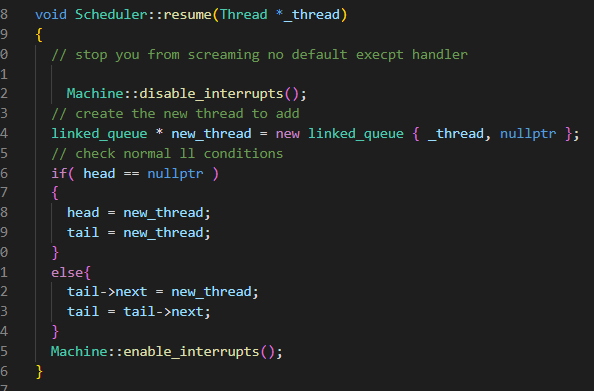
Schedule.C: Yield() :

Take a copy of head, then change to the next head pointer and switch out, deleting the copy of head after.



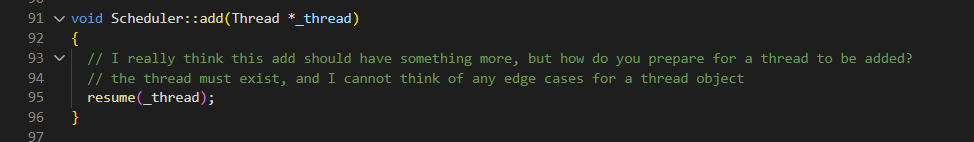
Schedule.C: Resume() :

We create a new thread, properly place it in the linked list, and lastly re-enable interrupts.



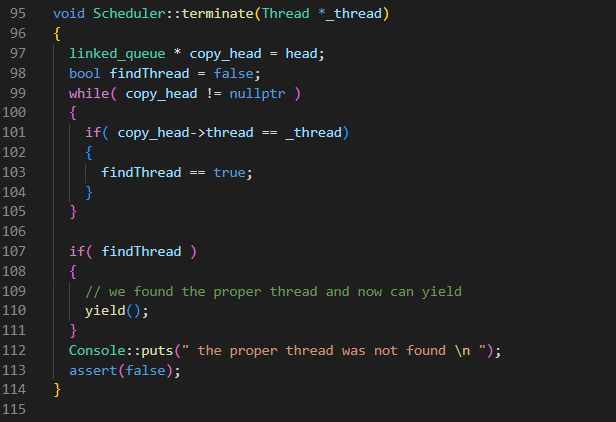
Schedule.C: Add() :

This calls resume, as the thread is already runable



Schedule.C: terminate() :

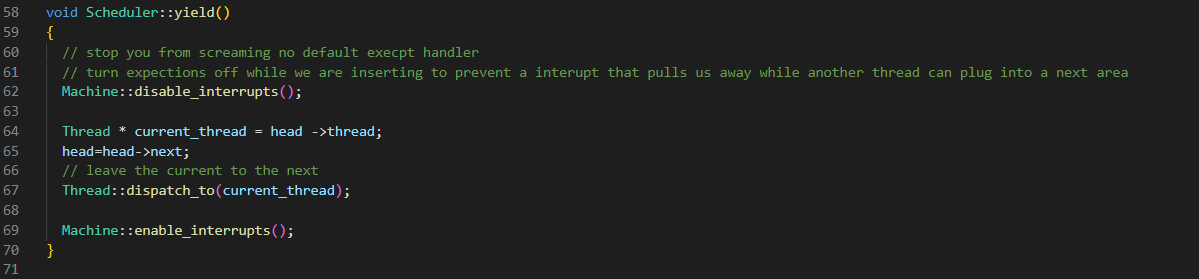
Delete the thread, and swap out of the currently running thread, which in yield will delete the current head and context switch out of the now deleted thread.



Bonus 1:

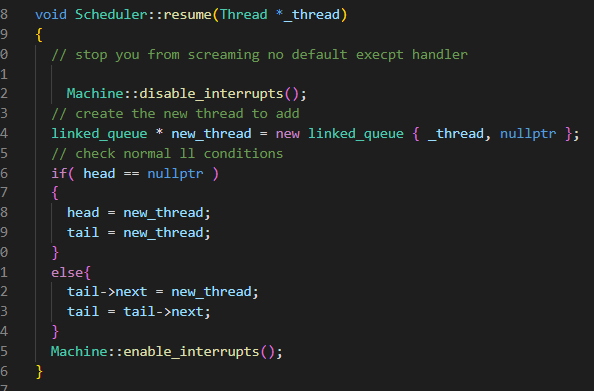
Schedule.C: Yield() :

Disable interrupts to safely swap out in the context switch and enable them after the switch is finished.



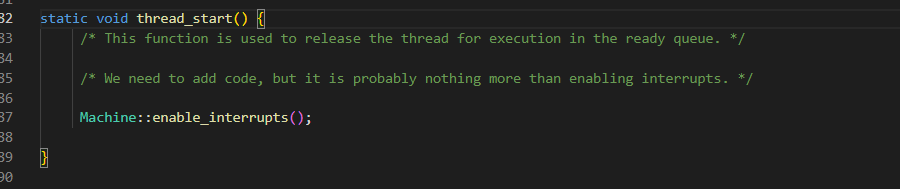
Schedule.C: Resume() :

Disable interrupts to create and insert the thread, enable them later to allow the machine to interrupt if needed.



Thread.C: thread\_start( )

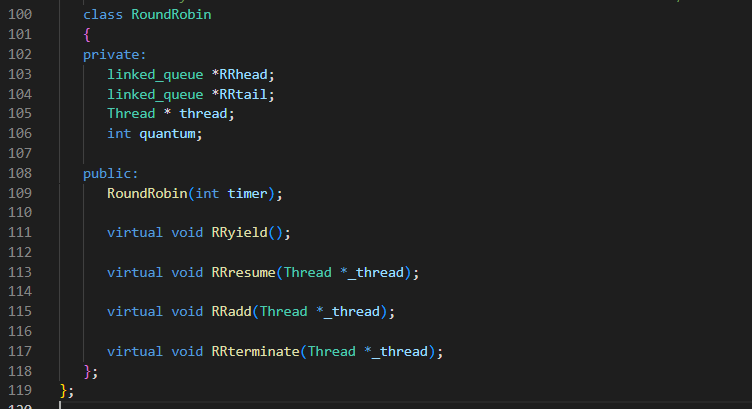
Enable interrupts to start the thread, and allow the garbage collector to handle the rest.



Bonus 2:

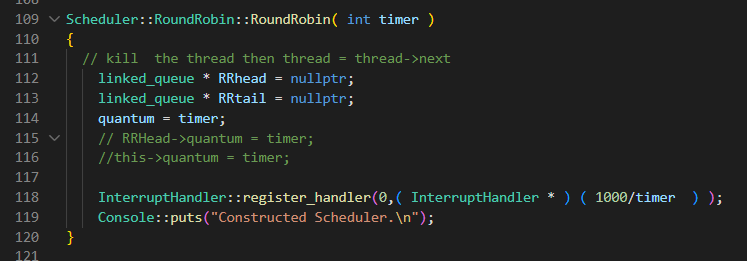
Schedule.H: RoundRobin Private Data Members:

Round Robin has a private linked list as well with a defined line quantum that will get modified each time the process is ejected. The default run time is defined to be 50ms.



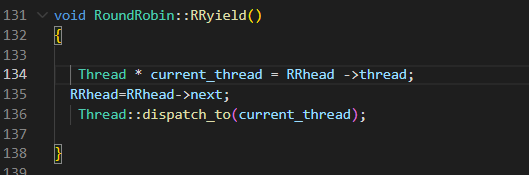
Schedule.C: RoundRobin::RoundRobin( int timer )

Initialize the head and tail, create the time quantum for the time the thread is allowed to run, and then set an interrupt that will trigger after 50ms.



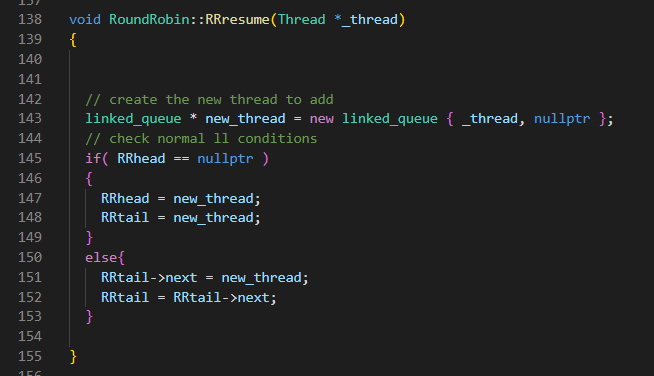
Schedule.C: RoundRobin::RRYield( )

Take a copy of the head, change to the next thread and delete the copy of the thread.



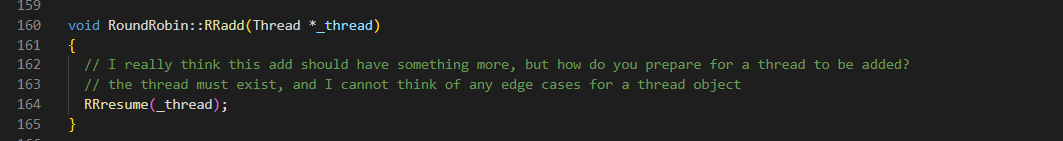
Schedule.C: RoundRobin::RRResume( Thread \* \_thread)

Create a new thread and allocate it into its proper position.



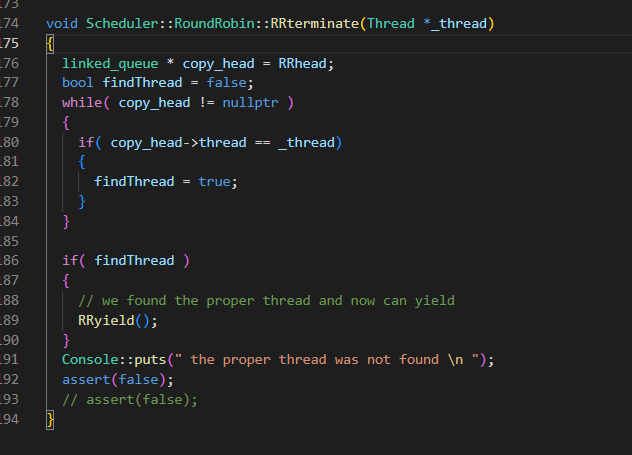
Schedule.C: RoundRobin::RRadd( )

Add the thread back into the queue to make it resume.

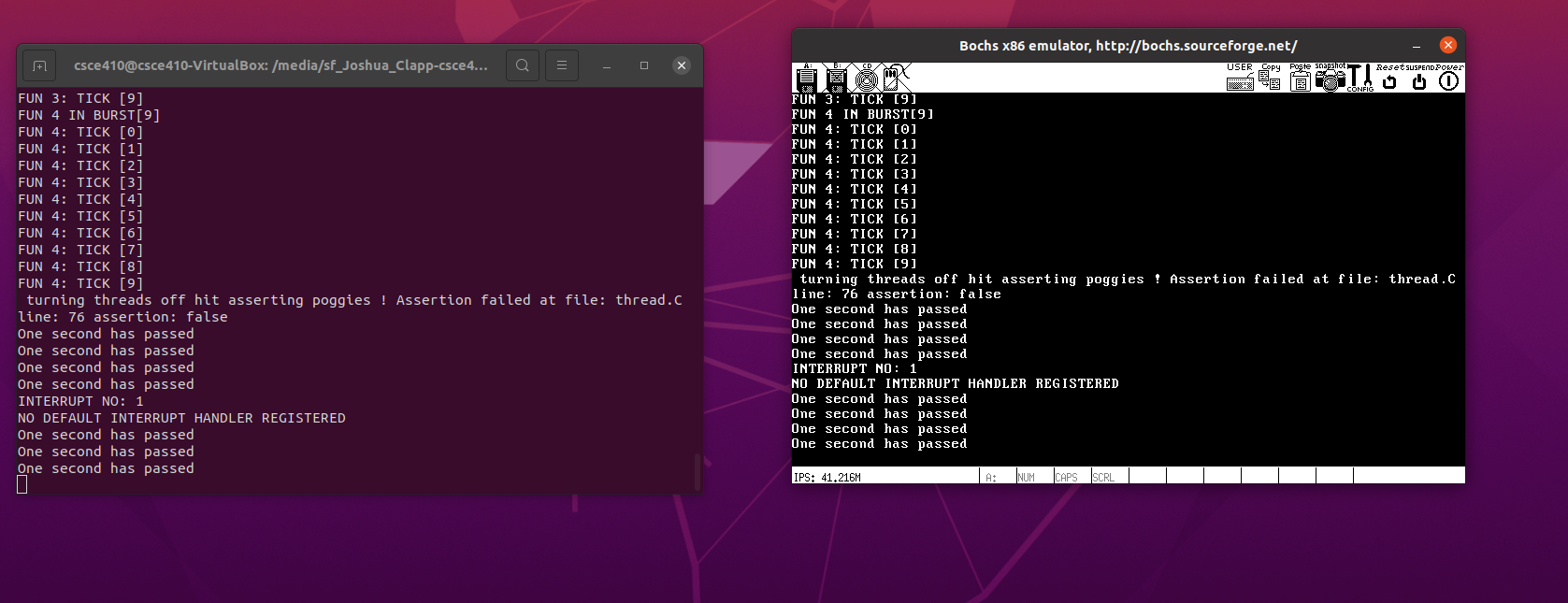


Schedule.C: RoundRobin::RRterminate( )

Delete the thread and swap back into the proper thread.



Testing

I have implemented no other testing, and attached a screenshot of the final output.I believe I am ignoring the atomic operations as there's no mux or locking from a context switch mid thread besides disabling the locks.  
  
FIFO Scheduling with Interrupts Enabled Output  


Round Robin Scheduling with Interrupts Enabled Output  
