Jacob Hessong

MP5 Design Document

Options 1 and 2 selected

Modifications: scheduler.H, thread.H/C, added Queue.H, makefile, Interrupts.C, Simple_timer.C. Queue.H implemented a FIFO queue to handle the ready queue to support scheduling, with a statement added to the makefile to include it in compilation. Scheduler.H was then modified to use this structure as well as define NULL for use. The thread files were modified to add a yield function to support round robin scheduling. Interrupts.C was modified to move the interrupt handler call later to support round robin scheduling. Finally, simple_timer.C to create 50ms time quantums for round robin scheduling.

This machine problem focused on the implementation of a basic FIFO scheduler for kernel threads. To begin doing this, a FIFO queue class needed to be created that would be able to keep track of these threads. This class has a simple list of Thread structures with a pointer to the next thread. One constructor initializes everything to NULL, while another has a thread passed in as an argument and is added to the list during construction. This class also contains an enqueue function, which sets the first item to the argument if the list is empty and sets next to the argument if not. This function uses recursion to iterate through the list. Finally, this class also has a dequeue function, which simply iterates through the list and returns the top thread.

The scheduler class had a queue object added to it to maintain the ready queue as well as an int to track its size. The scheduler constructor simply sets the size of the ready queue to 0. The yield function checks if there are any items in the ready queue and returns NULL if not. If there is, the size of the queue is decremented, the top thread is dequeued, and then the scheduler dispatches to this thread. The resume and add functions simply add the passed thread to the ready queue and increment the size. Finally, the terminate function was implemented by looping through the ready queue and finding the thread with the same ID as the argument and deleting it. This function also needed the

thread_shutdown function in thread.C, which terminates through the scheduler and deletes the current thread before yielding to bring in another thread.

For the bonuses, option 1 was implemented by enabling interrupts when threads are started. Option 2 was implemented by first creating time quantums in the simple_timer.C file. Next, the call for the interrupt handler in the dispatch_interrupt function was moved later in the function after the OI signal. Next, a new function in the thread files was added that resumes a thread and yields another through the scheduler to help with yielding the thread by putting the current thread at the end of the queue and loading a new one.