Jacob Hessong

MP6 Design Document

Options attempted – 3 and 4

Changes: added scheduler created with previous project including queue class, updated makefile to include these new files, updated kernel to test blocking disks, and added queue to blocking disk files to support scheduling.

The focus of this machine problem was to create a basic device driver for a disk using blocking disks in accordance with the design for option 3. To do this, the scheduler from the previous machine problem had to be updated with a blocking disk object. This is done by adding a function to add a disk to the scheduler, as well as adjusting the yield function to first check the disk object and verify if there is anything to read before scheduling the thread.

Next, the blocking disk class had to be implemented. This is done by using a queue of blocked threads on the disk. Another member is added to track the size of the queue. The other things added to this class are the enqueue_on_disk(), is_ready(), and wait_until_ready() functions. Enqueue_on_disk() adds the thread to the queue of blocked threads and updates the size of the queue. Wait_until_ready() checks if the disk is ready. If not, the current thread is added to the blocked thread queue and then the thread is yielded. Is_ready() simply calls the simple disk equivalent of this function since blocking disk are a child class.

Threads added to the blocked queue will only get the CPU when the disk is ready. Otherwise, the threads added to scheduler queue are looped through and threads waiting on I/O are in a separate queue so they won't get the CPU. This avoids busy waiting because threads waiting on I/O will not get scheduled by the scheduler.

Once yield finds that a disk is ready, it will check if there are any threads in the blocked thread queue. If so, the thread is dequeued, decrements the size of the queue, then dispatches the CPU to the thread. This thread can then perform read or write operations for what was blocked before yielding. This removes busy waiting.

Option 3 Design: This design will safely run multiple threads for one processor. Since several threads call read/write functions on the one CPU, the requests are queued into the blocked thread queue and are handled by the scheduler. This makes it so that only a single read/write request is done at once. In a multiprocessor system, there will be a race condition between the different CPUs to access I/O and protection will be needed. To this, the queue would need to be made thread safe and a lock would be needed for the I/O device. The thread safe queue ensures that the different processors do not update the queue at wrong times to avoid the race condition. Furthermore, the I/O lock makes sure that the device is not given up if it is already busy. The multiprocessor implementation of this problem cannot be implemented in this project as we only have a single processor.