Machine Problem 6: Simple Disk Device Driver

Nimoshika

UIN: 631000852

Objective: investigate kernel-level device drivers on top of a simple programmed-I/O block device. Block device used busy waiting. A layer is added on top to support Read and write operations.

Blocking Drive:

- Blocking Disk: Creates a blocking disk device with the given size connected to the master or slave slot of the primary ATA controller
- Disk Read: Reads 512 bytes from the given block of the disk and copies them to the given buffer
- Disk Write: writes 512 bytes from the buffer to the given block on the disk

  - **I have completed options 3 and 4**
- This design of device drivers will safely run multiple thread for a single processor.
- As several threads call the read/write function we queue the read write requests to a blocked queue and make the thread give up the cpu.
- For multiprocessor system, we will race condition between several cpu to access the I/O queue and will need some form of protection. For this we will need to implement a thread safe queue and a lock on the I/O device.

  The thread safe queue will make sure that different processors do not update the queue in wrong manner (if they are in a race condition) and a lock on the I/O
- device will ensure that we do not give up the resource if it is busy. This implementation cannot be tested in our current setup as we only have one processor.

- The line where we see “Reading a block from disk in fun3” in the screen shot below is where the read request is made. THIS IS NOT the actual read operation but the debug that the thread is requesting a read operation. After this if another thread reads, it is simply queued.
- The next line later which says “Now READING in FUN2” is when the actual read is happening from the device. This shows that the FIFO order is guaranteed.
- As there are 2 threads requesting read operation, we made them queue up in the blocked thread. The actual operation is sequential and in FIFO order and therefore the thread safe case is tested.
- Disk enqueue, is ready and wait until ready is used for this option 3.