P6 Implementation :: Kernel-Level Device Drivers

Introduction:

Provided a simple_disk.C and simple_disk.H, we can analyze the original implementation. The provided block driver currently uses busy waiting to wait for I/O operations. Our task is to add a layer on top of the device to support a similar read/write implementation as provided by the SimpleDisk class.

Implementation:

In order to override the SimpleDisk read/write functions, I used a wait_until_ready function which would allow for virtualization. For our wait_until_ready, I simply make calls to the SYSTEM_SCHEDULER which called the P5 Scheduler class for the Queue and yield()/add() to the thread queue. I used a call to SimpleDisks's SimpleDisk::is_ready() to make sure that the threads we are yielding are not ready.

Fixes:

During the implementation of P6, I ran into an error with a "Exception 13 upon iteration 21" where Thread 1 attempts to run continuously. After testing increased buffer sizes for Thread 2 (I tried increasing the size of the buffer to 2048 to see if it would change anything), I realized the issue was I never properly updated the tail of my Queue in P5.

That portion is now fixed in the Pop() function of the Queue implementation. I put a comment on the section I added in the code provided in the zip file.

Conclusion:

The point of this project is to make sure the thread is passing the CPU to another thread. The CPU should not be held in a waiting state. We need to yield the CPU to another thread. After understanding what this meant, the issue became clear. I needed to add the current working thread to a queue and then yield the current thread.