

MP6: Primitive Device Driver

Jackson Stewart  
329009382  
CSCE 410

**Assigned Tasks**

Main: done

Bonus Options: not attempted

**Blocking disk implementation (blocking\_disk.C):** Implements the read and write operations

The read and write methods use the same `issue_operation()` and port interaction logic as in `SimpleDisk`, the difference being the way it waits. In this implementation I simply yield the thread to the scheduler until the disk is ready. As mentioned in the code comments, this does still use looping, but is not doing inefficient busy-waiting since it yields the CPU in each iteration. Since I'm not using interrupts (polling), there has to be a loop which checks the status somewhere in the code. For the sake of simplicity I choose to keep it in the read and write functions themselves instead of in the main loop in `kernel.C` with a blocked queue.

```
void BlockingDisk::read(unsigned long _block_no, unsigned char* _buf) {
    SimpleDisk::issue_operation(DISK_OPERATION::READ, _block_no);

#ifdef _USES_SCHEDULER_
    // yield until disk is ready (polling)
    // this is not busy waiting, but rather a way to yield the CPU
    // without having to implement a blocked queue
    while (!SimpleDisk::is_ready()) SYSTEM_SCHEDULER->yield();
#endif

#ifdef _USES_SCHEDULER_
    SimpleDisk::wait_until_ready();
#endif

    // read data from port
    for (int i = 0; i < 256; i++) {
        unsigned short data = Machine::inportw(0x1F0);
        _buf[i * 2] = (unsigned char)data;
        _buf[i * 2 + 1] = (unsigned char)(data >> 8);
    }
}
```

```
void BlockingDisk::write(unsigned long _block_no, unsigned char* _buf) {
    SimpleDisk::issue_operation(DISK_OPERATION::WRITE, _block_no);

#ifdef _USES_SCHEDULER_
    // yield until disk is ready (polling)
    while (!SimpleDisk::is_ready()) SYSTEM_SCHEDULER->yield();
#endif

#ifdef _USES_SCHEDULER_
    SimpleDisk::wait_until_ready();
#endif

    // write data to port
    for (int i = 0; i < 256; i++) {
        unsigned short data = _buf[i * 2] | (_buf[i * 2 + 1] << 8);
        Machine::outportw(0x1F0, data);
    }
}
```