Machine Problem 6: Primitive Disk Device Driver

Nachiket Umesh Naganure
UIN: 532008698
CSCE611: Operating System

Assigned Tasks

Main Task Completed.
Bonus Option 1: Completed.
Bonus Option 2: Did not attempt
Bonus Option 3: Completed
Bonus Option 3: Completed

System Design

The goal of the machine problem was to design kernel-level device driver to implement a non-blocking read and writes on top of existing busy waiting implementation of a simple disk.

1. Implement a BlockingDisk Class to handle IO threads which do not block CPU
2. Make changes in yield function of scheduler class to accommodate and yield to IO threads only if disk is ready
3. Implement MirroredDisk Class take care of Master and Dependent disks

1. **Main Task: BlockingDisk**
   (a) Main idea is to not wait for the IO thread wait till the disk operation is and IO read/write from machine port
   (b) When IO thread starts we make sure to yield immediately if the disk is not ready. The thread is put in block queue before yielding
   (c) In yield function of scheduler, changes are made to ensure that before dispatching a blocked thread, the disk is ready, otherwise we dispatch a thread from ready queue which is not IO thread.
2. **Bonus Option 1: MirrorDisk**
   (a) In Mirror Disk is extended from BlockingDisk and two new variables are added namely the master BlockingDisk and DependentDisk.
   (b) We also need to override read and write functions. In read we yield when the read is done to either disk and in write we make sure both disks are written to.
   (c) A new function wait is added which waits only till one of the disks is ready for the operation. This is used primarily in read() function.
3. **Bonus Option 3 & 4: Thread Safe and Concurrent**
   (a) In order to support concurrency, the threads should yield periodically and no one thread should hog up the CPU.
   (b) Since we have a single processor, the scheduler class from previous MP with some modifications takes care of rotating the threads access to CPU

(c) We also need to make sure that at any given time only one thread does IO operation on the disk

(d) Since we have a queue, we make sure that the thread at the front is only removed from the queue which the disk is ready to perform operation. This helps in avoiding multiple threads issuing IO operations to the disk

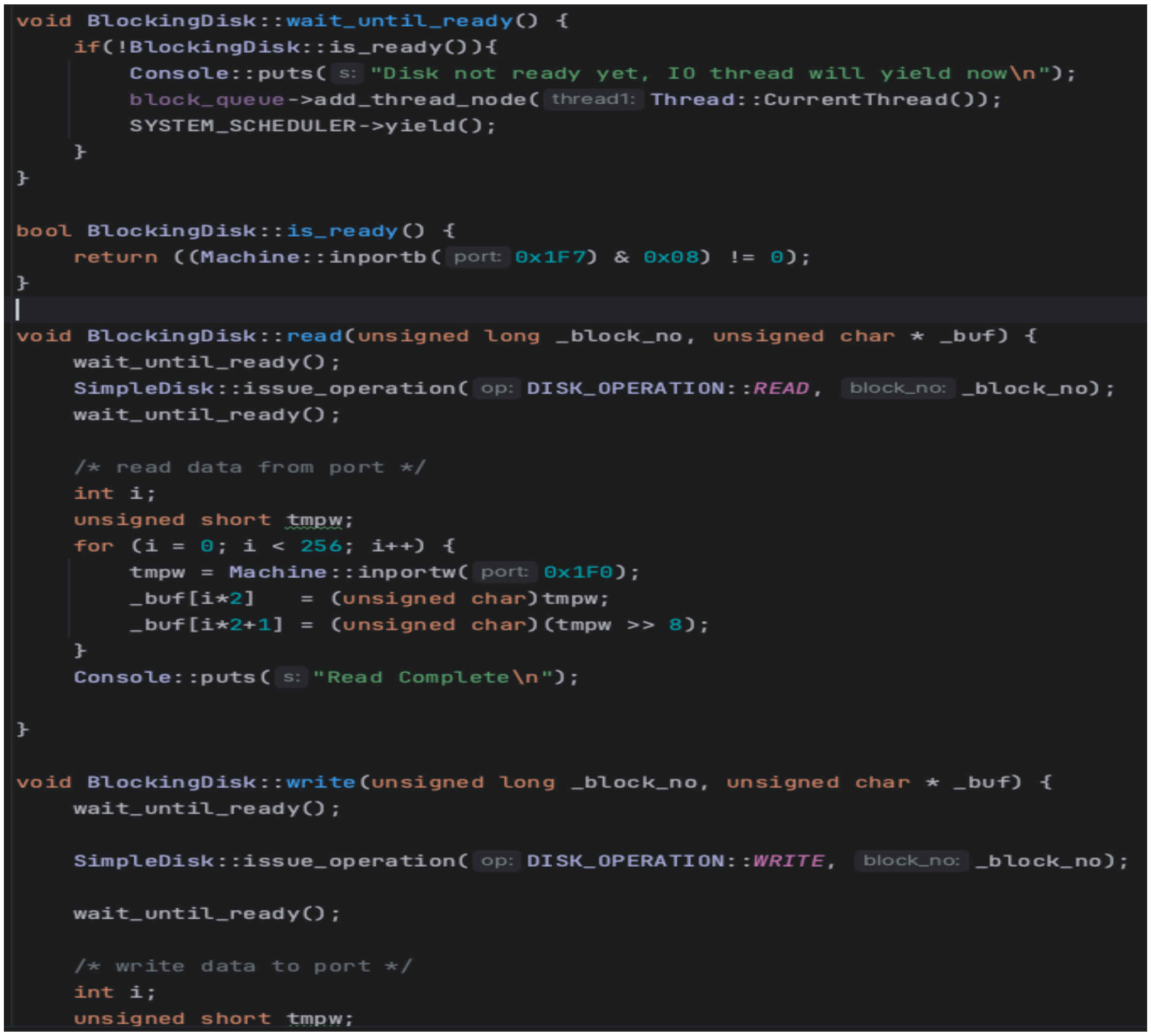
(e) So the scheduler in conjunction with the queue make sure only one thread does IO on disk.

(f) Another way to ensure thread safe systems is to use software implemented locks which can ensure that the read and write are done atomically.

Code Description

I mainly made changes in ”scheduler.c”, ”scheduler.h” and ”blocking_disk.c”, ”mirrored_disk.c”. To compile the code, you need to run the make file. To run the simulator, use copykernel.sh script and run the command ”bochs -f bochsrc.bxrc” to start the bochs simulator.

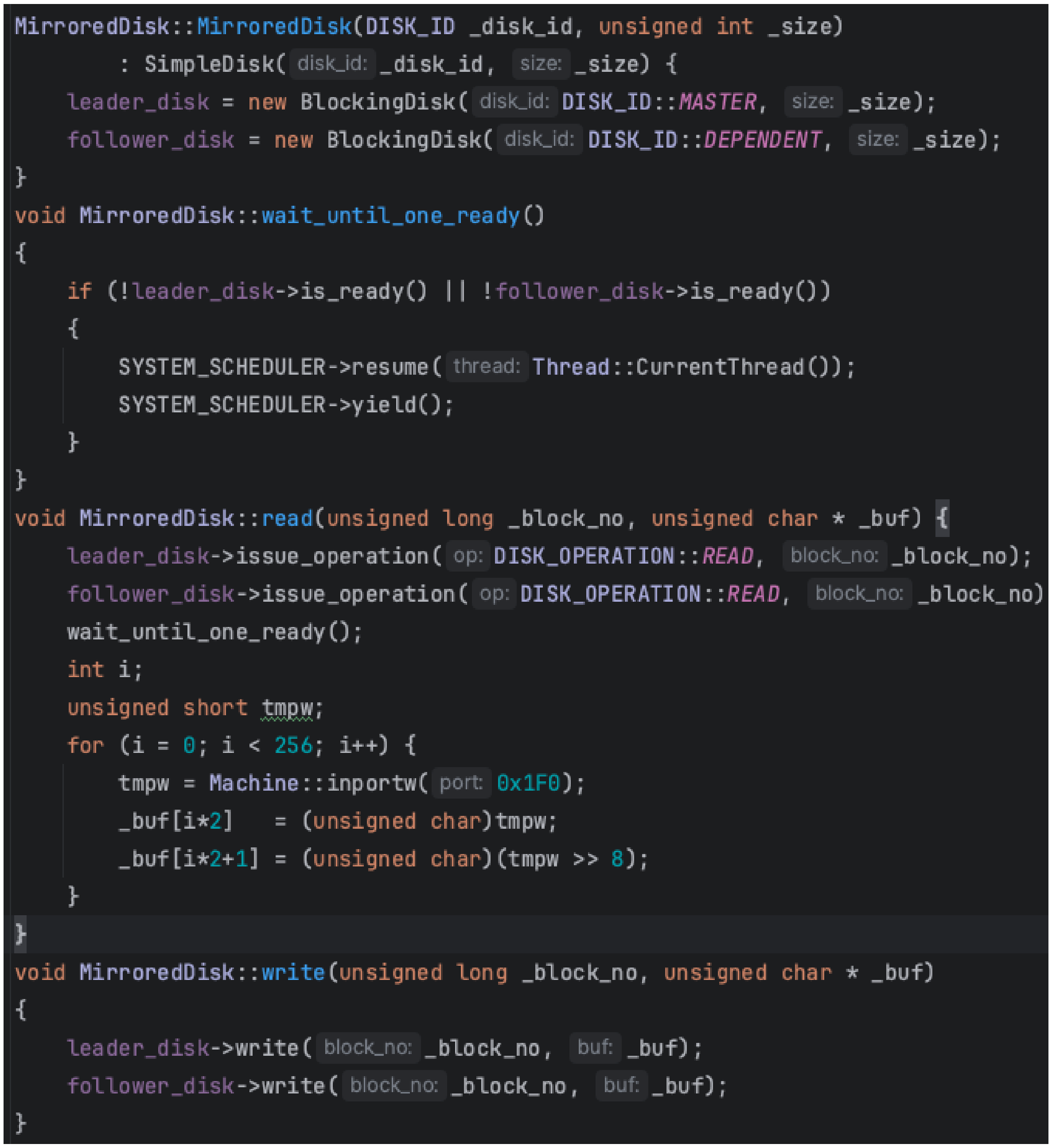
BlockingDisk Class :



```
void BlockingDisk::wait_until_ready() {
    if(!BlockingDisk::is_ready()){
        Console::puts( s: "Disk not ready yet, IO thread will yield now\n");
        block_queue->add_thread_node( thread1: Thread::CurrentThread());
        SYSTEM_SCHEDULER->yield();
    }
}

bool BlockingDisk::is_ready() {
    return ((Machine::inportb( port: 0x1F7) & 0x08) != 0);
}

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

    /* read data from port */
    int i;
    unsigned short tmpw;
    for (i = 0; i < 256; i++) {
        tmpw = Machine::inportw( port: 0x1F0);
        _buf[i*2]    = (unsigned char)tmpw;
        _buf[i*2+1]  = (unsigned char)(tmpw >> 8);
    }
    Console::puts( s: "Read Complete\n");

}

void BlockingDisk::write(unsigned long _block_no, unsigned char * _buf) {
    wait_until_ready();

    SimpleDisk::issue_operation( op: DISK_OPERATION::WRITE,  block_no: _block_no);

    wait_until_ready();

    /* write data to port */
    int i;
    unsigned short tmpw;
```

Figure 1: BlockingDisk Class

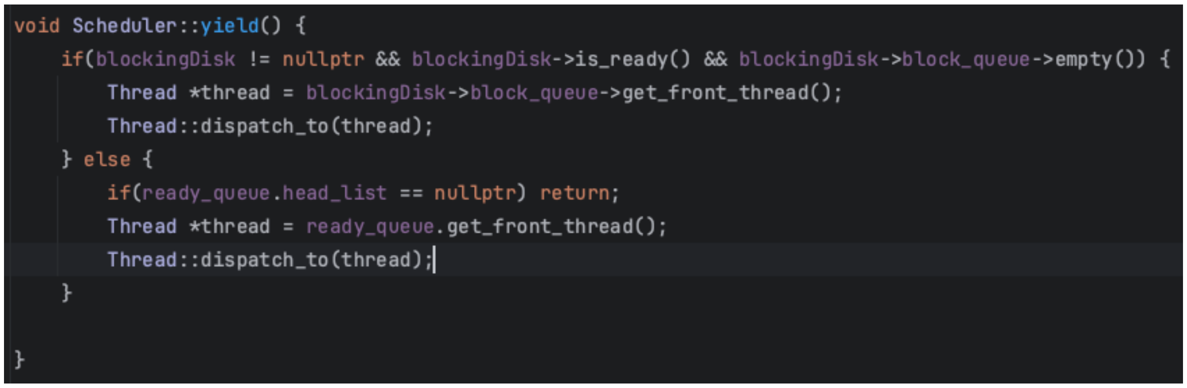
MirroredDisk Class :



```cpp
MirroredDisk::MirroredDisk(DISK_ID _disk_id, unsigned int _size)
        : SimpleDisk( disk_id: _disk_id,  size: _size) {
    leader_disk = new BlockingDisk( disk_id: DISK_ID::MASTER,  size: _size);
    follower_disk = new BlockingDisk( disk_id: DISK_ID::DEPENDENT,  size: _size);
}
void MirroredDisk::wait_until_one_ready()
{
    if (!leader_disk->is_ready() || !follower_disk->is_ready())
    {
        SYSTEM_SCHEDULER->resume( thread: Thread::CurrentThread());
        SYSTEM_SCHEDULER->yield();
    }
}
void MirroredDisk::read(unsigned long _block_no, unsigned char * _buf) {
    leader_disk->issue_operation( op: DISK_OPERATION::READ,  block_no: _block_no);
    follower_disk->issue_operation( op: DISK_OPERATION::READ,  block_no: _block_no)
    wait_until_one_ready();
    int i;
    unsigned short tmpw;
    for (i = 0; i < 256; i++) {
        tmpw = Machine::inportw( port: 0x1F0);
        _buf[i*2]   = (unsigned char)tmpw;
        _buf[i*2+1] = (unsigned char)(tmpw >> 8);
    }
}
void MirroredDisk::write(unsigned long _block_no, unsigned char * _buf)
{
    leader_disk->write( block_no: _block_no,  buf: _buf);
    follower_disk->write( block_no: _block_no,  buf: _buf);
}
```

Figure 2: MirroredDisk Class

Yield Function from scheduler class :

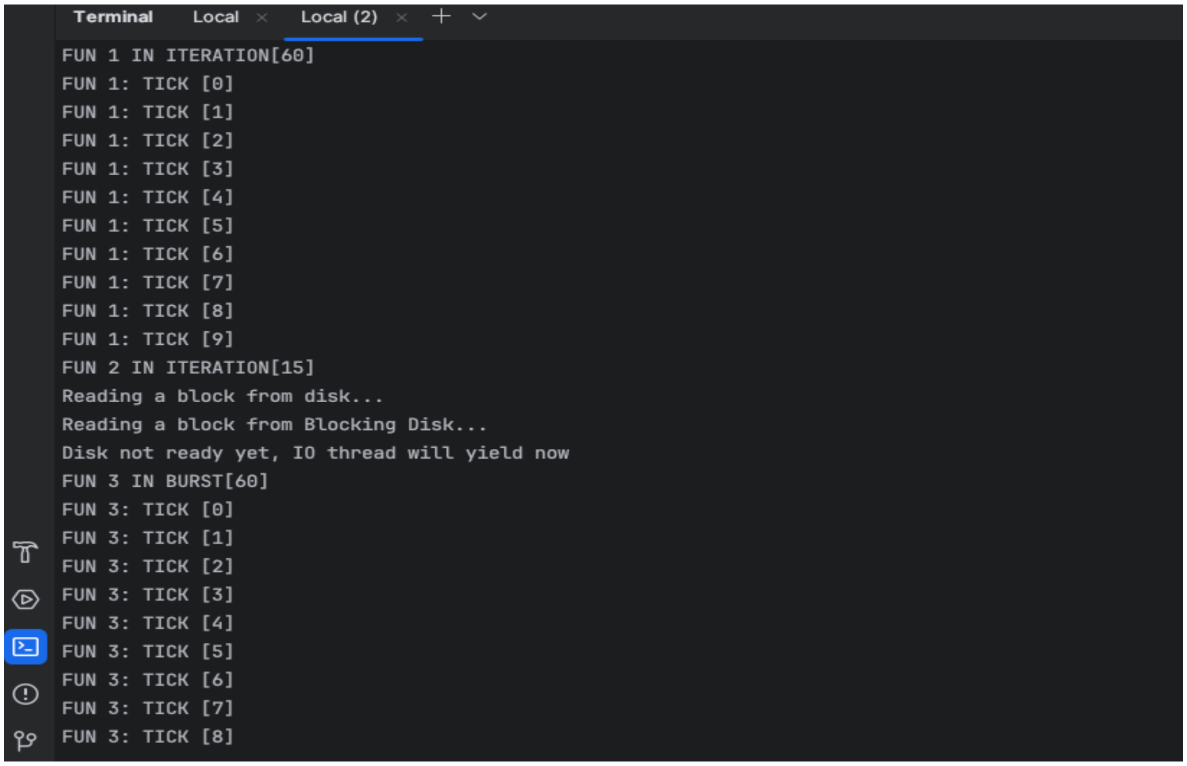


```
void Scheduler::yield() {
    if(blockingDisk != nullptr && blockingDisk->is_ready() && blockingDisk->block_queue->empty()) {
        Thread *thread = blockingDisk->block_queue->get_front_thread();
        Thread::dispatch_to(thread);
    } else {
        if(ready_queue.head_list == nullptr) return;
        Thread *thread = ready_queue.get_front_thread();
        Thread::dispatch_to(thread);
    }

}
```

Figure 3: Yield Function from scheduler class

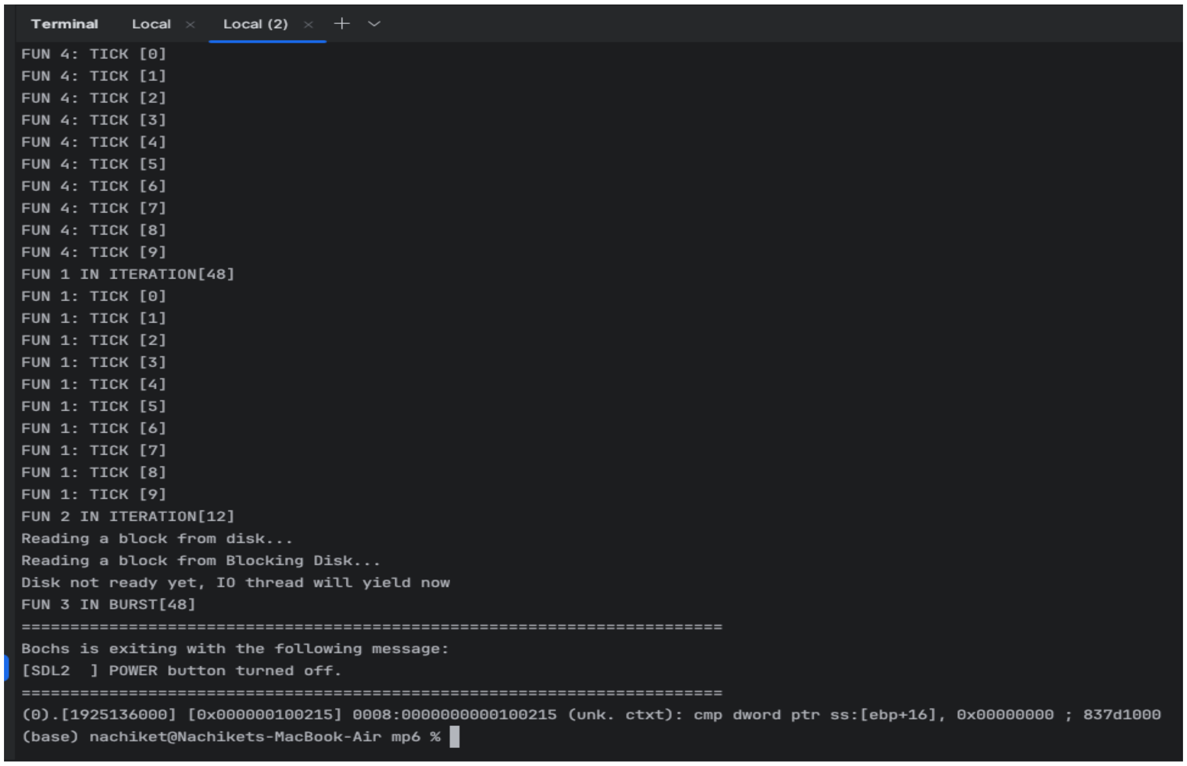
Testing

For testing, I used the given test function from kernel.C. I ran two scenarios where the _USES_SCHEDULER_, _MIRRORED_DISK_ and _BLOCKING_DISK_ macro was set and unset. All the tests are passing, for given and additional scenarios.



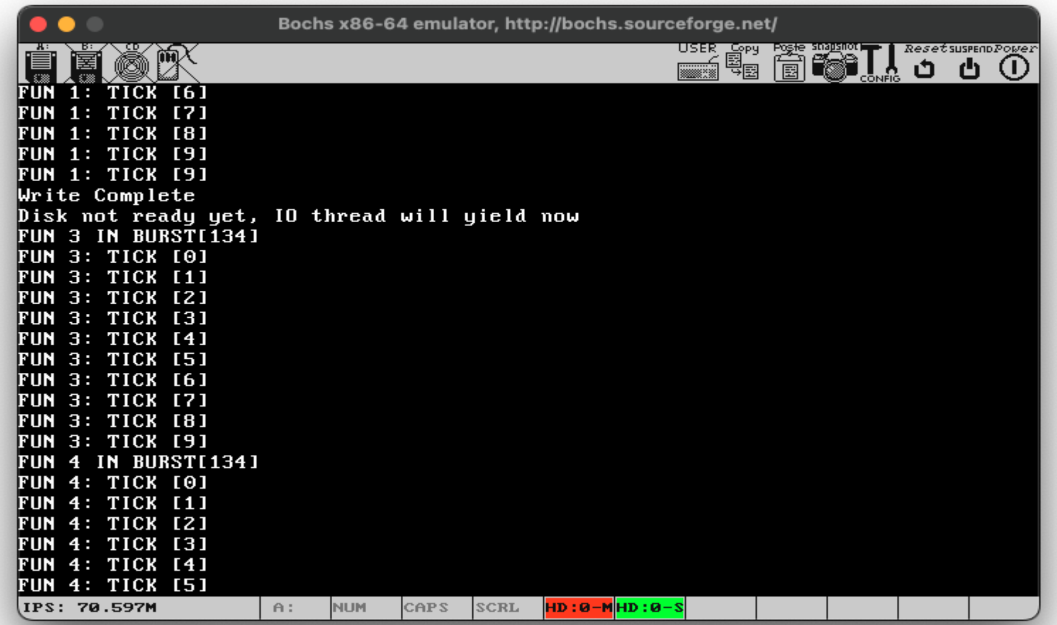
```
FUN 1 IN ITERATION[60]
FUN 1: TICK [0]
FUN 1: TICK [1]
FUN 1: TICK [2]
FUN 1: TICK [3]
FUN 1: TICK [4]
FUN 1: TICK [5]
FUN 1: TICK [6]
FUN 1: TICK [7]
FUN 1: TICK [8]
FUN 1: TICK [9]
FUN 2 IN ITERATION[15]
Reading a block from disk...
Reading a block from Blocking Disk...
Disk not ready yet, IO thread will yield now
FUN 3 IN BURST[60]
FUN 3: TICK [0]
FUN 3: TICK [1]
FUN 3: TICK [2]
FUN 3: TICK [3]
FUN 3: TICK [4]
FUN 3: TICK [5]
FUN 3: TICK [6]
FUN 3: TICK [7]
FUN 3: TICK [8]
```

Figure 4: Testing BlockedDisk, here thread 2 returns from just after read operation is issued due to disk not being ready



```
FUN 4: TICK [0]
FUN 4: TICK [1]
FUN 4: TICK [2]
FUN 4: TICK [3]
FUN 4: TICK [4]
FUN 4: TICK [5]
FUN 4: TICK [6]
FUN 4: TICK [7]
FUN 4: TICK [8]
FUN 4: TICK [9]
FUN 1 IN ITERATION[48]
FUN 1: TICK [0]
FUN 1: TICK [1]
FUN 1: TICK [2]
FUN 1: TICK [3]
FUN 1: TICK [4]
FUN 1: TICK [5]
FUN 1: TICK [6]
FUN 1: TICK [7]
FUN 1: TICK [8]
FUN 1: TICK [9]
FUN 2 IN ITERATION[12]
Reading a block from disk...
Reading a block from Blocking Disk...
Disk not ready yet, IO thread will yield now
FUN 3 IN BURST[48]
========================================================================
Bochs is exiting with the following message:
[SDL2  ] POWER button turned off.
========================================================================
(0).[1925136000] [0x000000100215] 0008:0000000000100215 (unk. ctxt): cmp dword ptr ss:[ebp+16], 0x00000000 ; 837d1000
(base) nachiket@Nachikets-MacBook-Air mp6 %
```

Figure 5: Thread yield from read before thread 3 takes over.



```
FUN 1: TICK [6]
FUN 1: TICK [7]
FUN 1: TICK [8]
FUN 1: TICK [9]
FUN 1: TICK [9]
Write Complete
Disk not ready yet, IO thread will yield now
FUN 3 IN BURST[134]
FUN 3: TICK [0]
FUN 3: TICK [1]
FUN 3: TICK [2]
FUN 3: TICK [3]
FUN 3: TICK [4]
FUN 3: TICK [5]
FUN 3: TICK [6]
FUN 3: TICK [7]
FUN 3: TICK [8]
FUN 3: TICK [9]
FUN 4 IN BURST[134]
FUN 4: TICK [0]
FUN 4: TICK [1]
FUN 4: TICK [2]
FUN 4: TICK [3]
FUN 4: TICK [4]
FUN 4: TICK [5]
```

Figure 6: In MirrorDisk testing, write is done for master but yields before dependent write is done since disk is not ready