CSCE-611: Operating Systems Machine Problem 6

Name: Vamsi Tallam UIN: 432001932

Files modified/added:

1. Blocking_disk.C
2. Blocking_disk.H
3. Mirroring_disk.C
4. Mirroring_disk.H
5. thread.C (From MP 5)
6. kernel.C
7. scheduler.C
8. scheduler.H
9. makefile

Attempted all **BONUS** sections, support for mirroring, interrupts for concurrency, design, and implementation of a thread-safe disk system.

The following are the changes made:

1. **Blocking_disk.C**
    a. It is derived from simple_disk.C and I reused the code from mp5 to fetch the scheduler the interrupts have been perfectly handled by making use of the functions enable_interrupts and disable_interrupts.
    b. The significant change made to eliminate busy wait is using wait_until_ready() and it is modified in a way that the thread which is waiting for an I/O operation yields the CPU if it is busy. So, extern Scheduler* SYSTEM_SCHEDULER has been used to yield and resume the CPU as and when needed.
    c. A disk queue is implemented and the threads that are waiting for the disk will be pushed and extracted in the order of their queueing to allow for fair access to the disk in FIFO.
    d. The read() and write() functions have been modified by adding the functionality of new wait_until_ready()
    e. **BONUS** 1: Concept of Mirroring has been implemented by deriving from Simple_disk.C and it is similar to the implementation of blocking_disk.C
    f. The reading and writing from Fun2 in Kernel.C is handled based on the availability of blocking_disk.C and the mirroring_disk.C.
    g. The issue_operation() has not been modified and it is re-used from Simple_disk.C.

2. **BLOCKING_DISK.H**
   - Defined private variables needed for the read and write operations in blocking_disk.C and are initialized in .C file.
3. **Kernel.C**
   a. The header files were included and the USES_SCHEDULER is uncommented to make use of the scheduler.
   b. SYSTEM_DISK is modified to point to BlockingDisk(MASTER,….).
   c. Also the overloaded delete(void* , size_t) has been added to kernel.C to get rid of linker issues to get the functionality of the scheduler correct.
   d. Console::puti() has been added to print the values onto the screen.
4. **makefile:**
   a. Made necessary modifications to include mirroring_disk.C/H, scheduler.C/H, and other minor changes.
   b. Also made changes to ensure disk gets cleaned upon make clean.

Bonus Section:

- **Support for Disk Mirroring:**
  - The MIRROREDDISK class is derived from simple_disk and a few functions like the readFromRAID_1() and writeToRAID_1() have been defined for issuing low-level operations.
  - read() and write() functions are functionally the same as of those in BlockingDisk.
  - This mirroring is initialised from the BlockingDisk constructor by taking DISK_ID::DEPENDENT as the argument.
- **Using Interrupts for Concurrency**:
  - The status of interrupts is checked every time a new thread is scheduled, thereby making sure that the interruption is handled correctly

    The following checks have been done to make use of interrupts for concurrency

```
if(Machine::interrupts_enabled())
        Machine::disable_interrupts()
 if(Machine::interrupts_enabled())
        Machine::enable_interrupts();
```

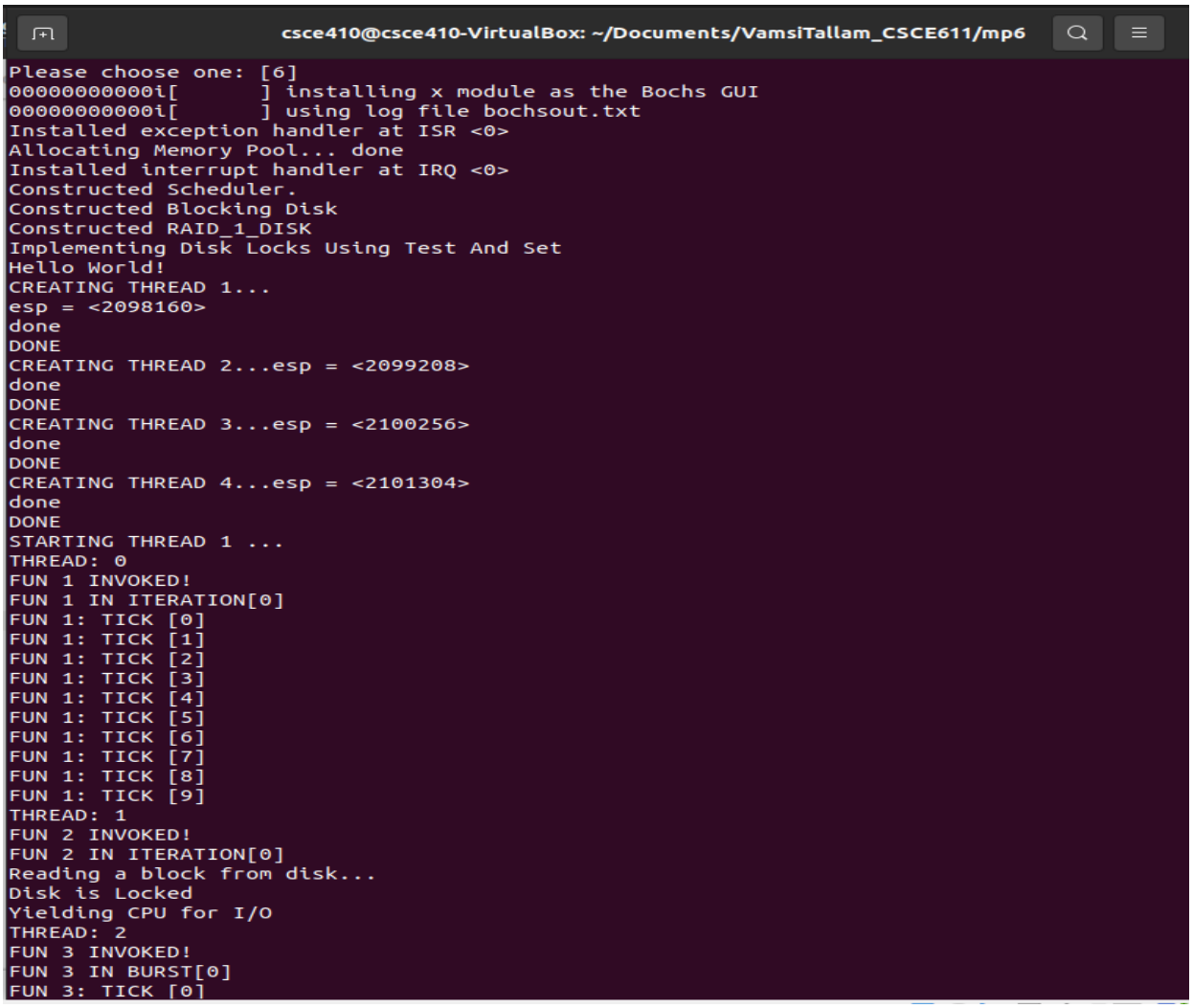
  - These conditional checks for interrupts and properly enabling and disabling them while yielding, adding and resuming the executions of threads and also for disk waiting operations in the wait_until_ready() functions ensured concurrency.
- **BONUS 3 & 4: Design and Implementation of Thread-Safe Disk System:**
  - Thread-safe disk operations can be implemented by making use of locks.
  - Using a simple test and set mechanism while reading and writing can help us make only one thread access the disk at a time

- So, whenever a thread does a read or write, and if the lock is available, the disk is locked until the read or write is completed and then it is unlocked.
- So, in simple words only one thread is entering the critical section if it has the access to the lock.

- **BONUS 4: Implementation:**
  - Whenever a disk operation is issued, the lock is acquired and after the writing is done, the lock is released
  - The Locking is implemented by a simple Test and Set where in we test the current lock and then wait for the lock to be released and acquire it and then lock it and release the lock once done.

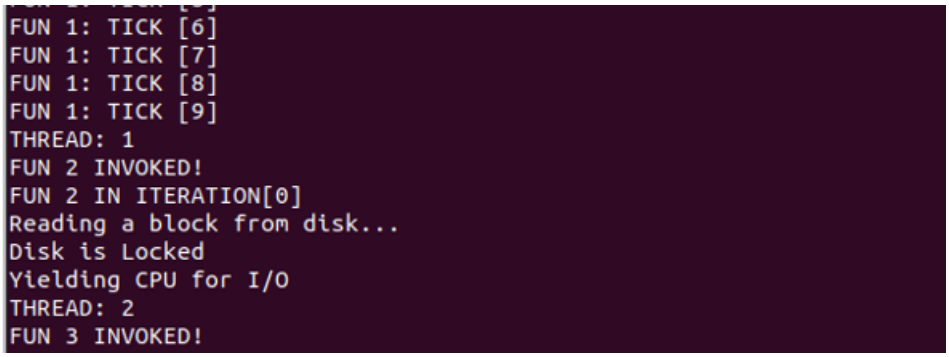
Results:

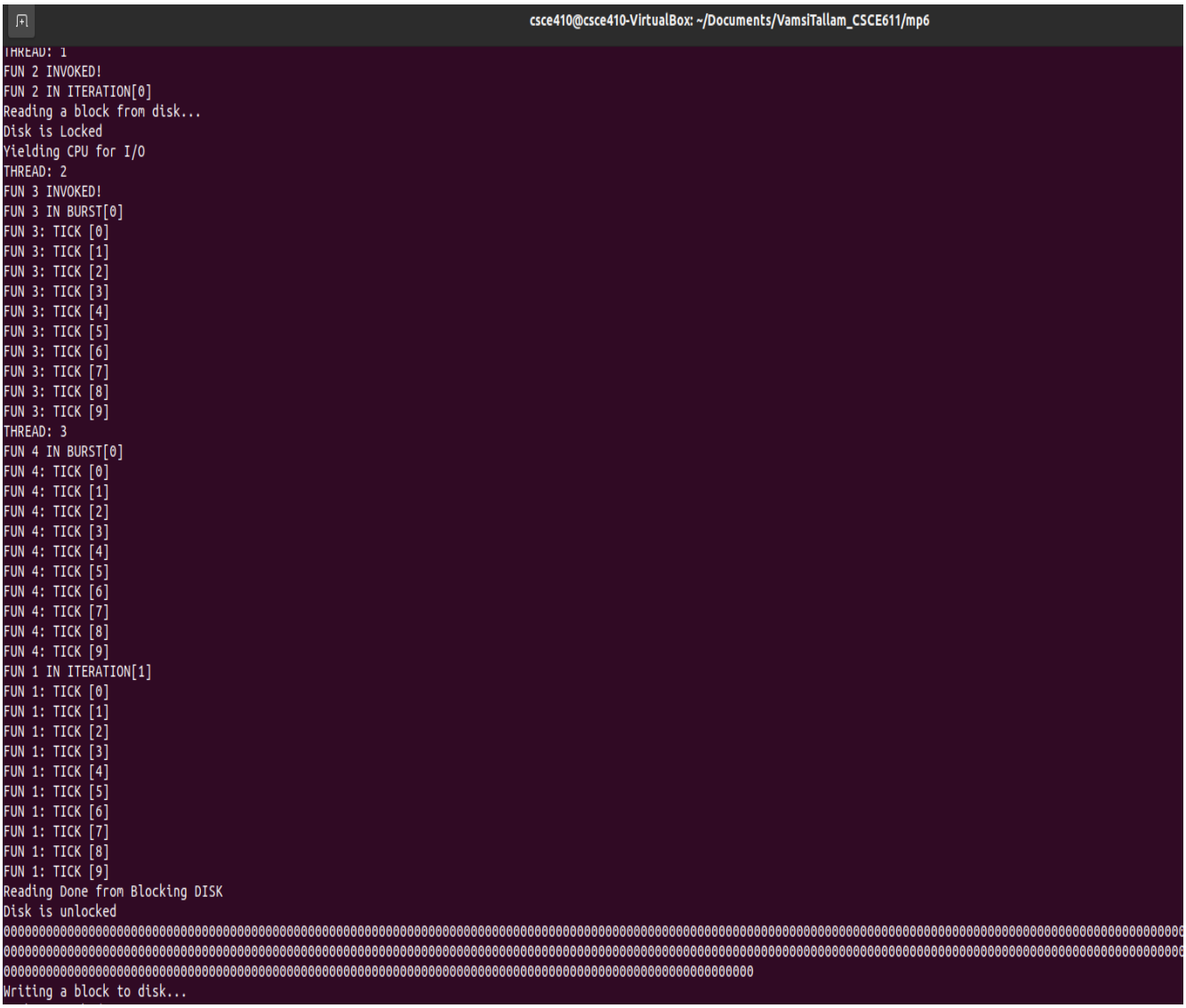
Initialization and starter output:



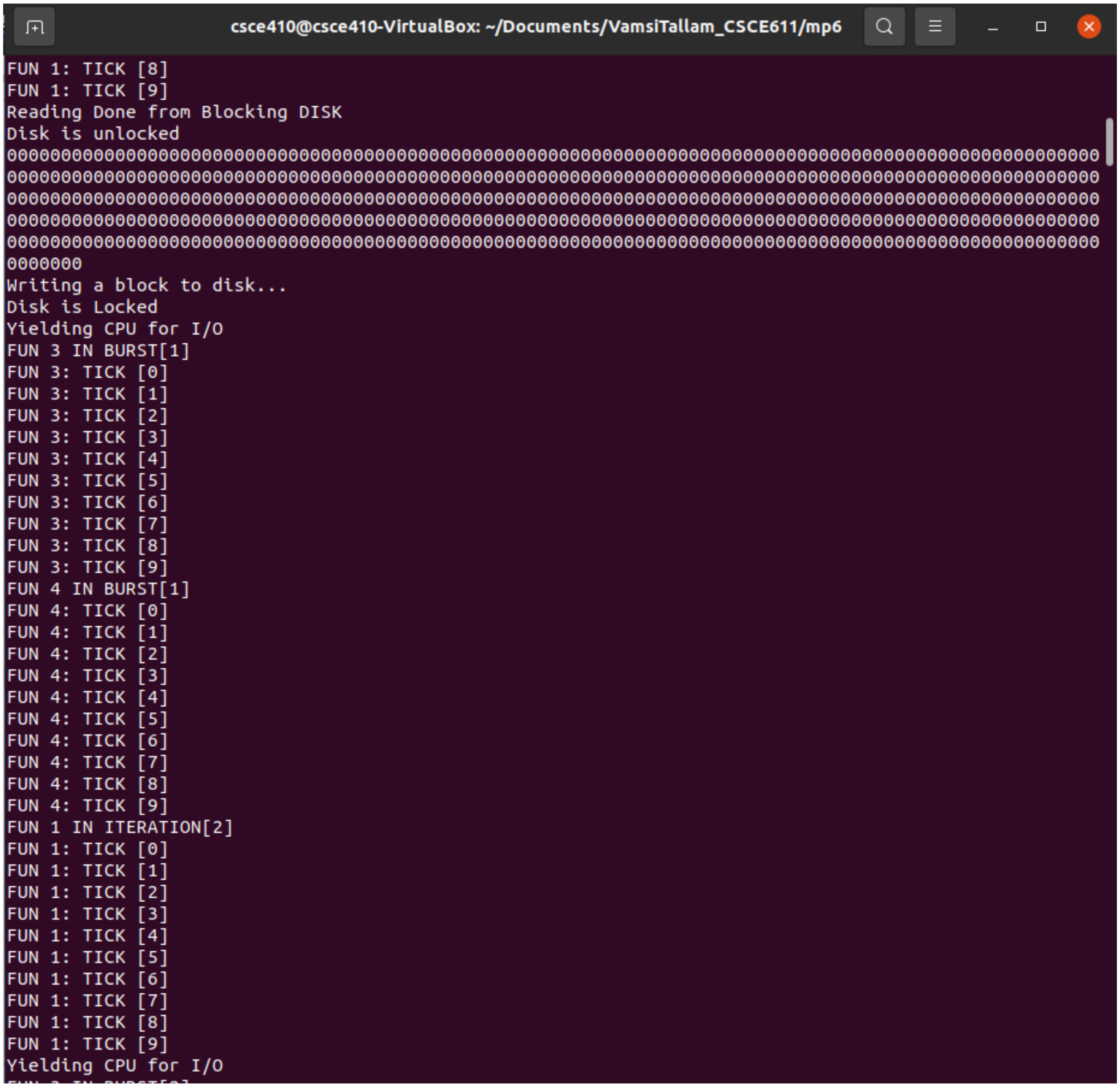
Elimination of Busy Waiting: CPU is yielded once the disk I/O has been issued

Elimination of Busy waiting while read operation:

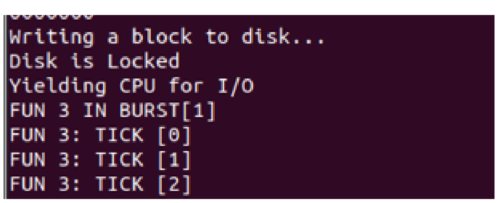




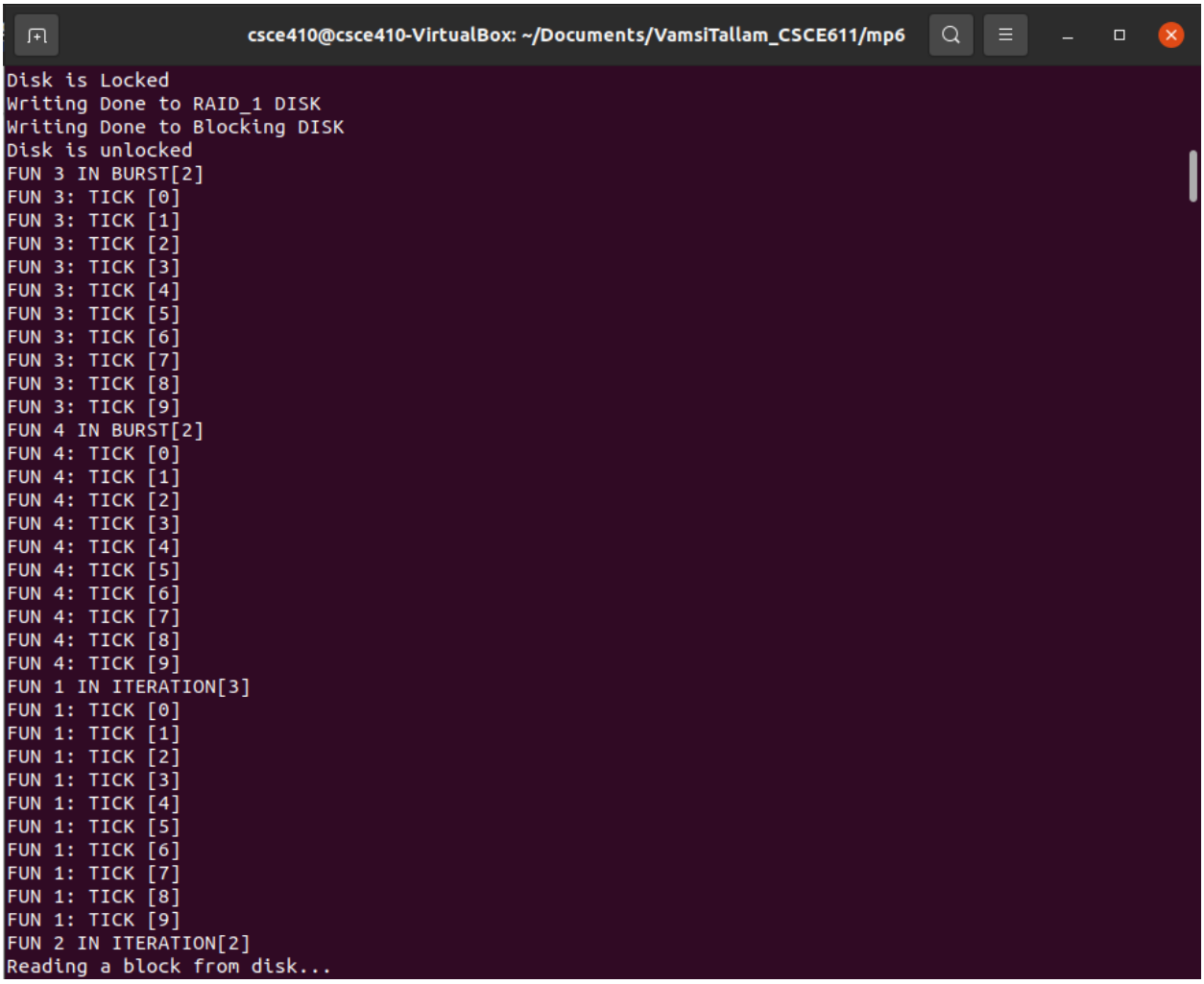
Elimination of Busy waiting while write operation: As we can CPU is yielded once a write is issued and if the disk is not available the successive iterations also yield CPU



Once the disk is free, we start writing



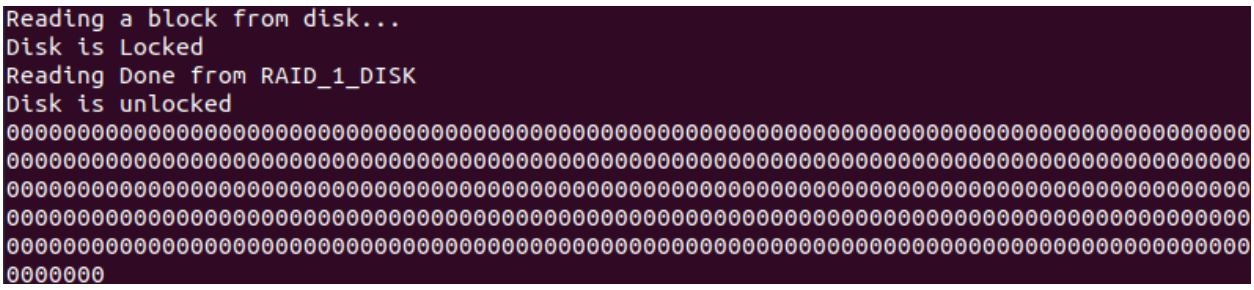
Once written the disk is unlocked



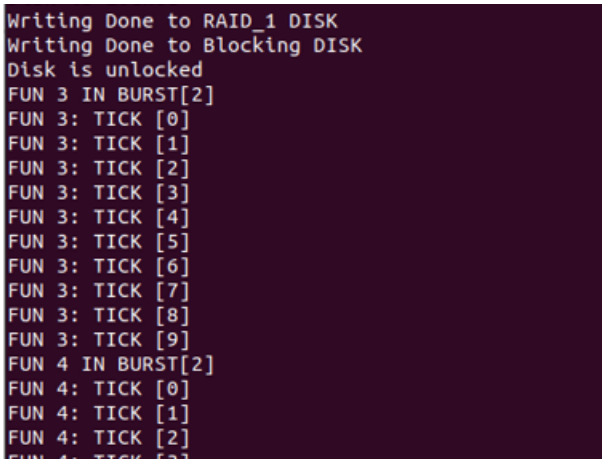
Mirroring Disk:

We write to both the mirroring disk and the blocking disk and can issue operations for read from either of the disks depending on the availability as seen in the image

Read Operation for mirroring:

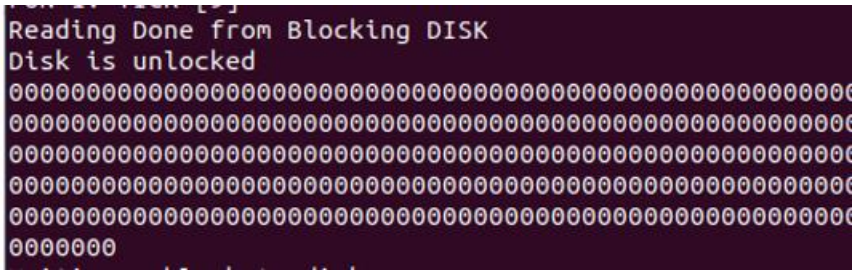


Write Operation for mirroring:



Disk Lock: Thread Safe System:

We can see that whenever there is a read operation, the lock is acquired and it is released once the reading is done.



Similarly, the lock is acquired while we start to write onto the disk and it is released once the writing is done.

Interrupts for concurrency:

A snippet from the code has been attached that is used to ensure interrupts for concurrency

```
        // Interrupt handling for Bonus
        if(Machine::interrupts_enabled())
                Machine::disable_interrupts();
```