MP6: Primitive Disk Device Driver

Dhanraj Murali
UIN: 734003894
CSCE611: Operating System

Assigned Tasks:

Main: Completed.
Bonus:
Option 1 Disk Mirroring - Completed.
Option 3 Design Thread Safe Disk - Completed

System Design:

1. Disk Device Driver that yields when there is I/O wait so that other threads can execute.

Code Description:

I. Whenever a read or write function is called, the current thread is added to the back of the queue and the control is yielded by called the yield function.
II. Basic scheduler execution from MP5 is continued, the thread that yields the control on an I/O operation from above step is pushed to the end of the queue and when it gains control again as per the order of execution in the queue, it checks if I/O device is ready and if not it yields control again.

Files modified:

a. simple_fifo.H – MP5
b. thread.C – MP5
c. scheduler.C – MP5
d. scheduler.H – MP5
e. kernel.C
f. blocking_disk.C
g. blocking_disk.H

Function Description:

MP Basic Target:

In this MP, we have designed and implemented a disk device driver that supports block and execute without busy waiting. Blocking disk class is updated with the following functionality in the wait_until_ready function in which every time a read or write operation is called, the function checks if the I/O is ready and if it is not ready the thread gets pushed back to the queue and it yields control until its next execution time as per the queue. In this way busy waiting is avoided.

Bonus Option – 1:
Disk Mirroring

```
4
5     /* -- DISK DEVICE -- */
6
7     //SYSTEM_DISK = new SimpleDisk(DISK_ID::MASTER, SYSTEM_DISK_SIZE);
8     //#define _USES_MIRRORING_
9     #ifdef _USES_MIRRORING_
0     SYSTEM_DISK = new MirroringDisk(DISK_ID::MASTER, SYSTEM_DISK_SIZE);
1     #else
2     SYSTEM_DISK = new BlockingDisk(DISK_ID::MASTER, SYSTEM_DISK_SIZE);
3     #endif
4
5
```

Mirroring Disk class is derived public from blocking disk class. It has 2 objects of the type blockingdisk with disk ids as master and dependent. The modifications that are implemented are as follows,

1. issue_operation function takes in disk id as an input argument.
2. Read function is called by issue_operation on both master and dependent and whichever returns first will be used to read.
3. Write function is called upon both master and dependent disks so that they have identical data and act as backups.

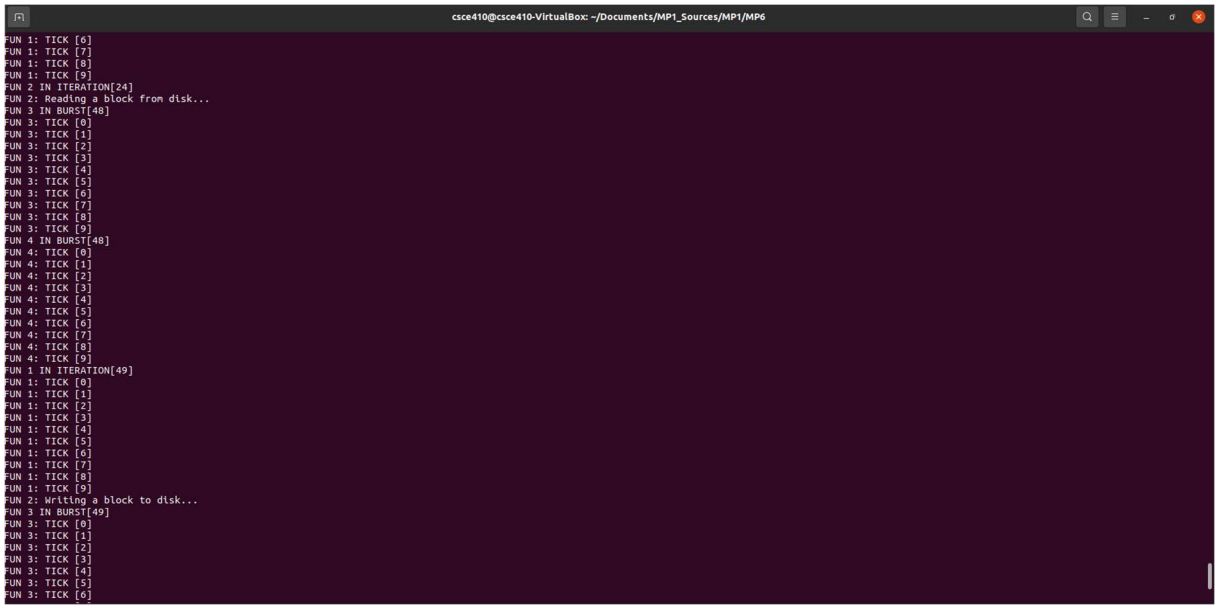
In kernel.C file, the MirroringDisk object is created based on if defined condition that checks if "_USES_MIRRORING_" variable is defined or not. We can uncomment the line in order to test this functionality.

Bonus Option – 3:
Thread safe disk system

Normally to avoid race conditions, we can deem certain portions of the code to be critical section. These critical sections are such that any improper or concurrent access to the code of that section will lead to race conditions between the threads involved causing a dip in the utilization of the CPU. We can use few methods to implement the critical section resolution. I prefer using the test and set method to lock the critical section. In the functionality that we are interested in this machine problem, both read and write operations are critical. Hence, we can use test and set method to lock and unlock the critical section whenever a read or write operation is called.

OUTPUT:

When function 2 calls "read", thread yields control and is pushed at the end of the queue.



Normal Simple Disk Implementation: