# Design Document MP6 Primitive Disk Device Driver

- Vasudha Devarakonda

Following changes are made to accommodate the MP

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   MP6/blocking_disk.C
        modified:   MP6/blocking_disk.H
        modified:   MP6/kernel.C
        modified:   MP6/makefile
        modified:   MP6/simple_disk.H

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        MP6/disk_mirror.C
        MP6/disk_mirror.H
        MP6/queue.H
        MP6/scheduler.C
        MP6/scheduler.H

vasudhadevarakonda@connect-172-31-41-9  ~/Desktop/OS/MPs  main ++
```

Problem statements attempted

1. Basic Blocking Disk
2. [BONUS] Disk Mirroring
3. [BONUS] Interrupts for Concurrency
4. [BONUS] Logic -> Thread safe disk system
5. [BONUS] Implementation of above thread disk system

Description

1. Basic blocking disk

This class mainly uses 2 methods write and read

- block_queue maintains the list of threads performing disk operations
- Write - This function checks if the disk is ready and based on that either writes else adds itself to disk_block_queue and gives up CPU

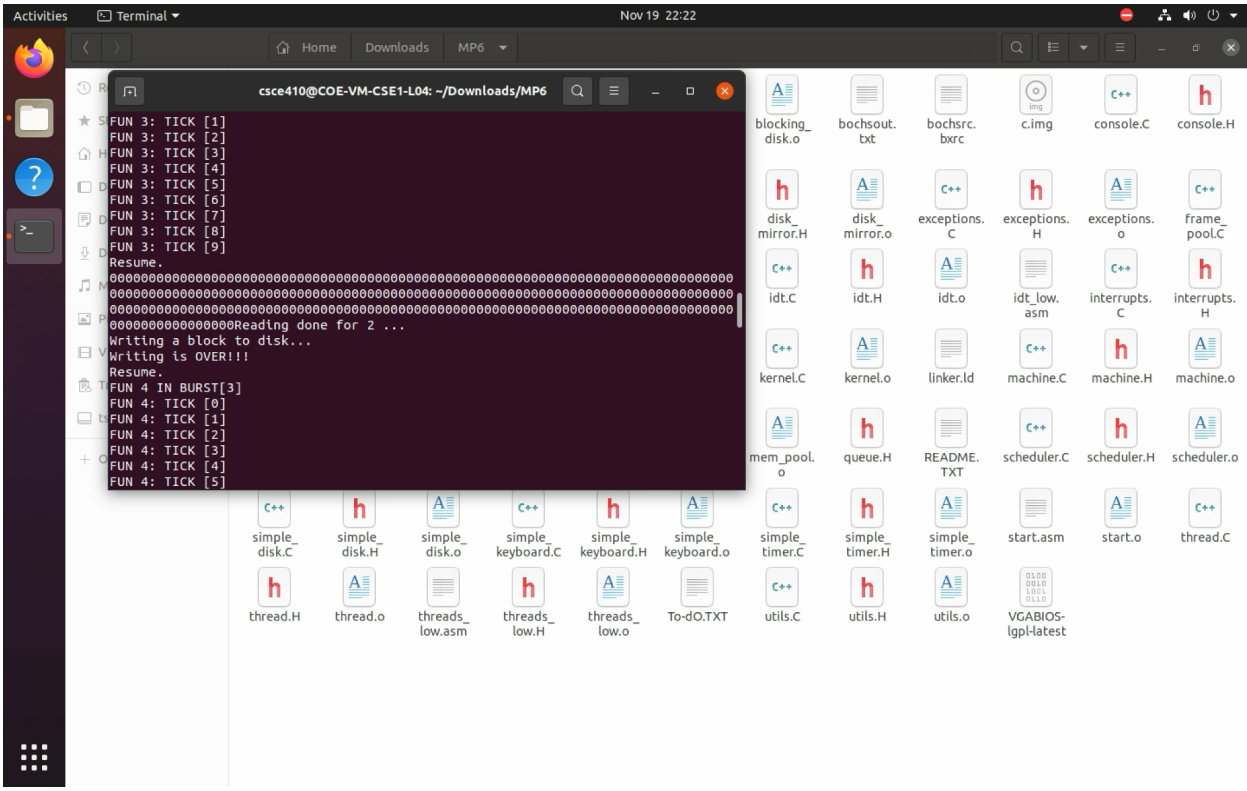
```
void BlockingDisk::write(unsigned long _block_no, unsigned char *_buf)
{
  if (block_disk == false) //if disk is not blocked, no disk operation is to be
performed i.e. skipped
  {
    block_disk = true;
    SimpleDisk::issue_operation(DISK_OPERATION::WRITE, _block_no);
    if (!is_ready())
    {
      Thread *t = Thread::CurrentThread();
```

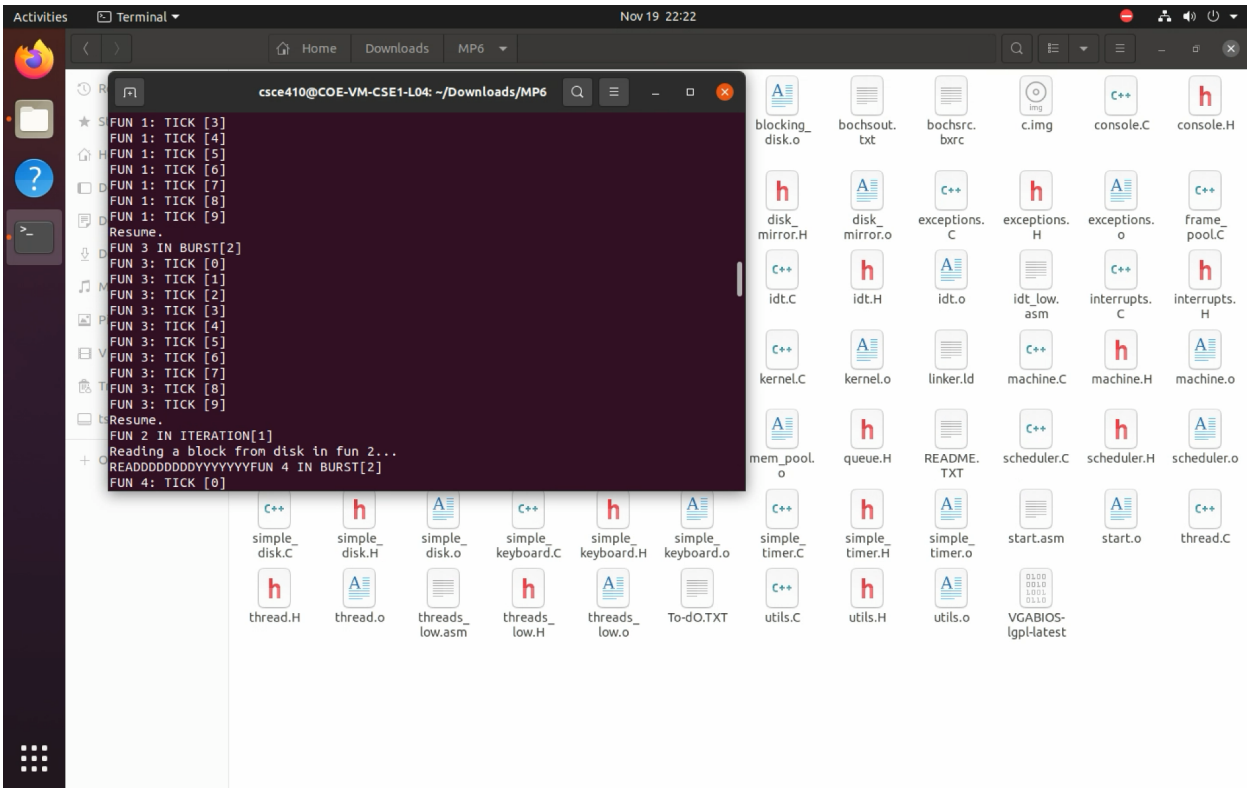
```
      block_queue->enqueue(t);
      block_queue_size++;
      SYSTEM_SCHEDULER->yield();
    }
    int i;
    unsigned short tmpw;
    for (i = 0; i < 256; i++)
    {
      tmpw = _buf[2 * i] | (_buf[2 * i + 1] << 8);
      Machine::outportw(0x1F0, tmpw);
    }
    block_disk = false;
  }
  else{
    Console::puts("Disk is busy....\n");
  }
}
```

- Read - This function checks if the disk is ready and based on that either writes else adds itself to disk_block_queue and gives up CPU

```
  void BlockingDisk::read(unsigned long _block_no, unsigned char *_buf)
  {
    if (block_disk == false)  //if disk is not blocked, no disk operation is to
be performed i.e. skipped
    {
      block_disk = true;
      SimpleDisk::issue_operation(DISK_OPERATION::READ, _block_no);
      if (!is_ready())
      {
        Thread *t = Thread::CurrentThread();
        //Console::puts("Disk is not ready so adding to block queue......\n");
        block_queue->enqueue(t);
        block_queue_size++;
        SYSTEM_SCHEDULER->yield();
        //Console::puts("Lets move to next thread......\n");
      }
      int i;
      unsigned short tmpw;
      for (i = 0; i < 256; i++)
      {
        tmpw = Machine::inportw(0x1F0);
        _buf[i * 2] = (unsigned char)tmpw;
        _buf[i * 2 + 1] = (unsigned char)(tmpw >> 8);
      }
      block_disk = false;
      //Console::putch('\n');
    }
    else{
      Console::puts("Disk is busy....\n");
```

```
    }
}
```


Activities
Terminal
Nov 19 22:22
Home
Downloads
MP6
csce410@COE-VM-CSE1-L04: ~/Downloads/MP6
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]
Resume.
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000Reading done for 2 ...
Writing a block to disk...
Writing is OVER!!!
Resume.
FUN 4 IN BURST[3]
FUN 4: TICK [0]
FUN 4: TICK [1]
FUN 4: TICK [2]
FUN 4: TICK [3]
FUN 4: TICK [4]
FUN 4: TICK [5]


Activities
Terminal
Nov 19 22:22
Home
Downloads
MP6
csce410@COE-VM-CSE1-L04: ~/Downloads/MP6
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]
Resume.
FUN 3 IN BURST[2]
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]
Resume.
FUN 2 IN ITERATION[1]
Reading a block from disk in fun 2...
READDDDDDDDYYYYYYYFUN 4 IN BURST[2]
FUN 4: TICK [0]

2. Disk Mirroring - Class `DiskMirror` is created which makes two `Blocking Disk objects`, master and slave

```
DiskMirror::DiskMirror(DISK_ID _disk_id, unsigned int _size) :
BlockingDisk(_disk_id, _size)
{
  master = new BlockingDisk(DISK_ID::MASTER, _size);
  slave = new BlockingDisk(DISK_ID::DEPENDENT, _size);
}
```

Write and read operations are modified

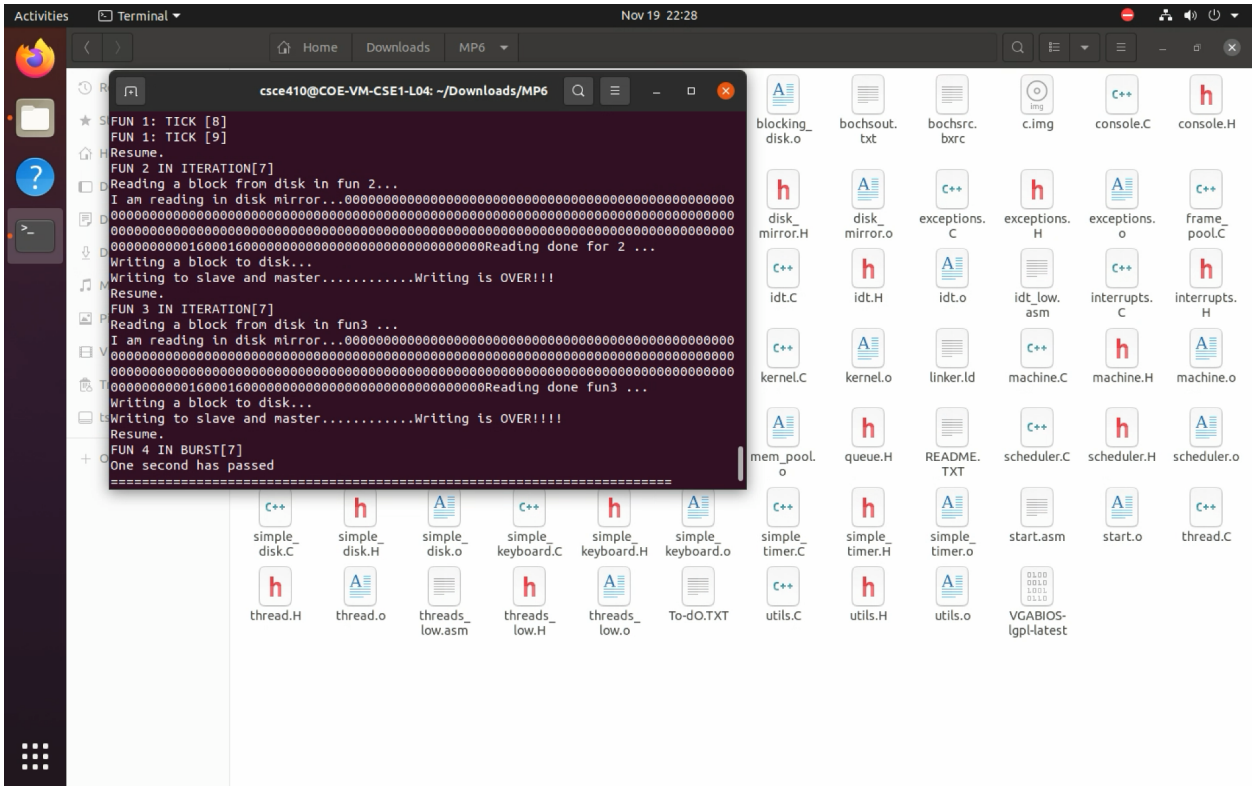
Write -> write operation is performed on both master and slave Blocking Disk

```
void DiskMirror::write(unsigned long _block_no, unsigned char *_buf)  // write
to both slave and master disks
{
  Console::puts("Writing to slave and master...........");
  master->write(_block_no, _buf);
  slave->write(_block_no, _buf);
}
```

Read -> this function first checks which disk, master or slave is ready first and accordingly perform read operation on that disk

void DiskMirror::read(unsigned long _block_no, unsigned char *_buf) // check which disk is ready, master or slave and then perform read operation on the disk which is ready first

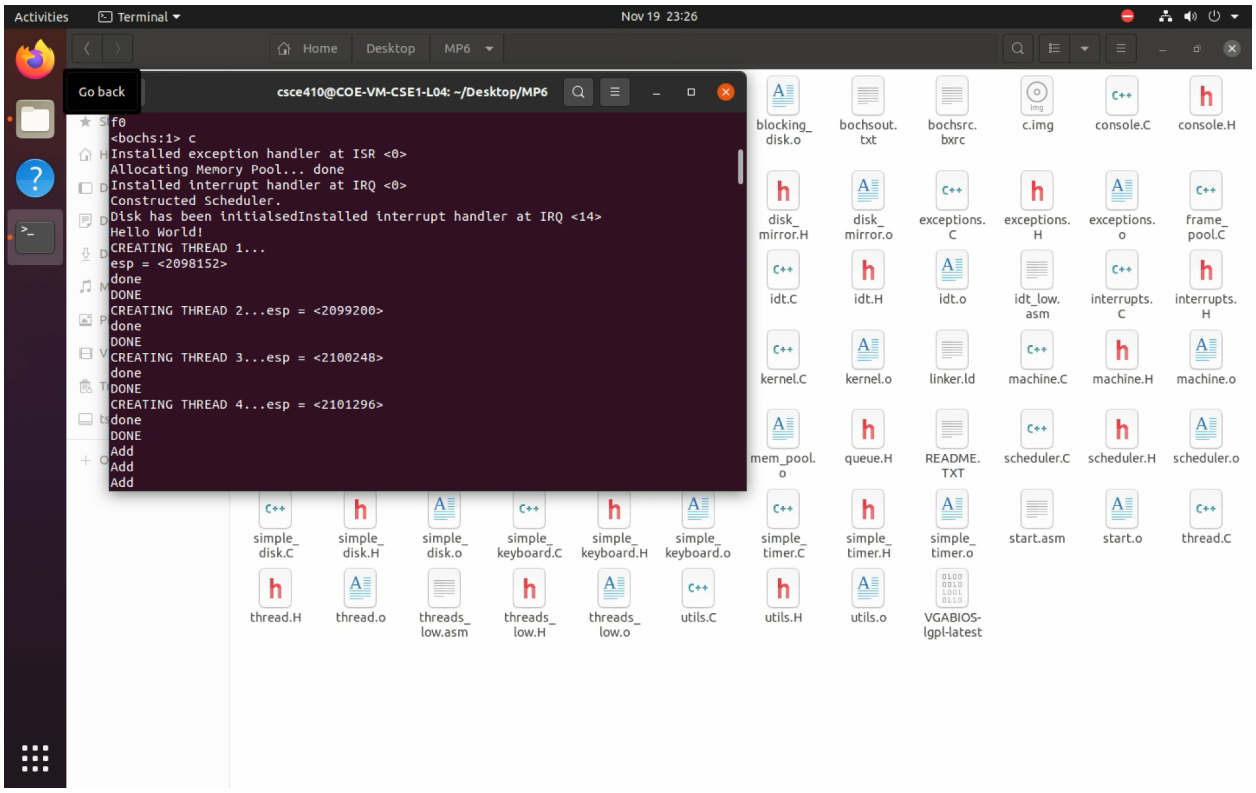
```
{
  Console::puts("Reading in disk mirror...");
  bool master_ready = master->is_ready() && !master->block_disk;
  bool slave_ready = slave->is_ready() && !slave->block_disk;
  while (master_ready || slave_ready) -> this checks if either master or slave is ready
  {
    if (master_ready) -> selects the first one which is read
    {
      master->read(_block_no, _buf);
    }
    else if (slave_ready)
    {
      slave->read(_block_no, _buf);
    }
    else
    {
      Thread *t = Thread::CurrentThread();
      master->block_queue->enqueue(t);
      slave->block_queue->enqueue(t);
      master->block_queue_size++;
      slave->block_queue_size++;

    }
  }
}
```

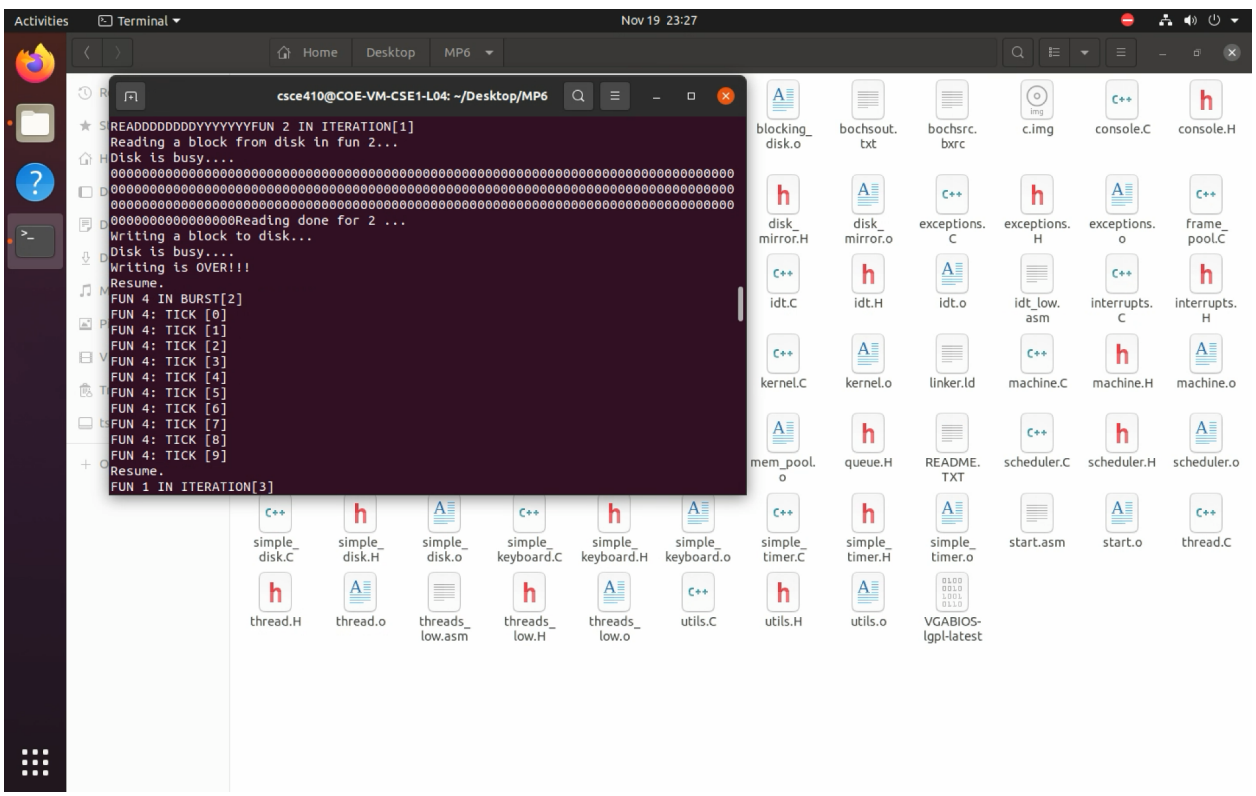



3. Interrupts for Concurrency - Interrupt 14 is used to manage thai feature. When interrupt arises the thread is dequeued from block_queue and appended back to thread/scheduler queue

```
void BlockingDisk::disk_enqueue()
{
  if (block_queue_size > 0)
  {
    Thread *t = block_queue->dequeue();
    block_queue_size--;
    SYSTEM_SCHEDULER->resume(t);
  }
```

}




4. & 5 Thread safe disk system

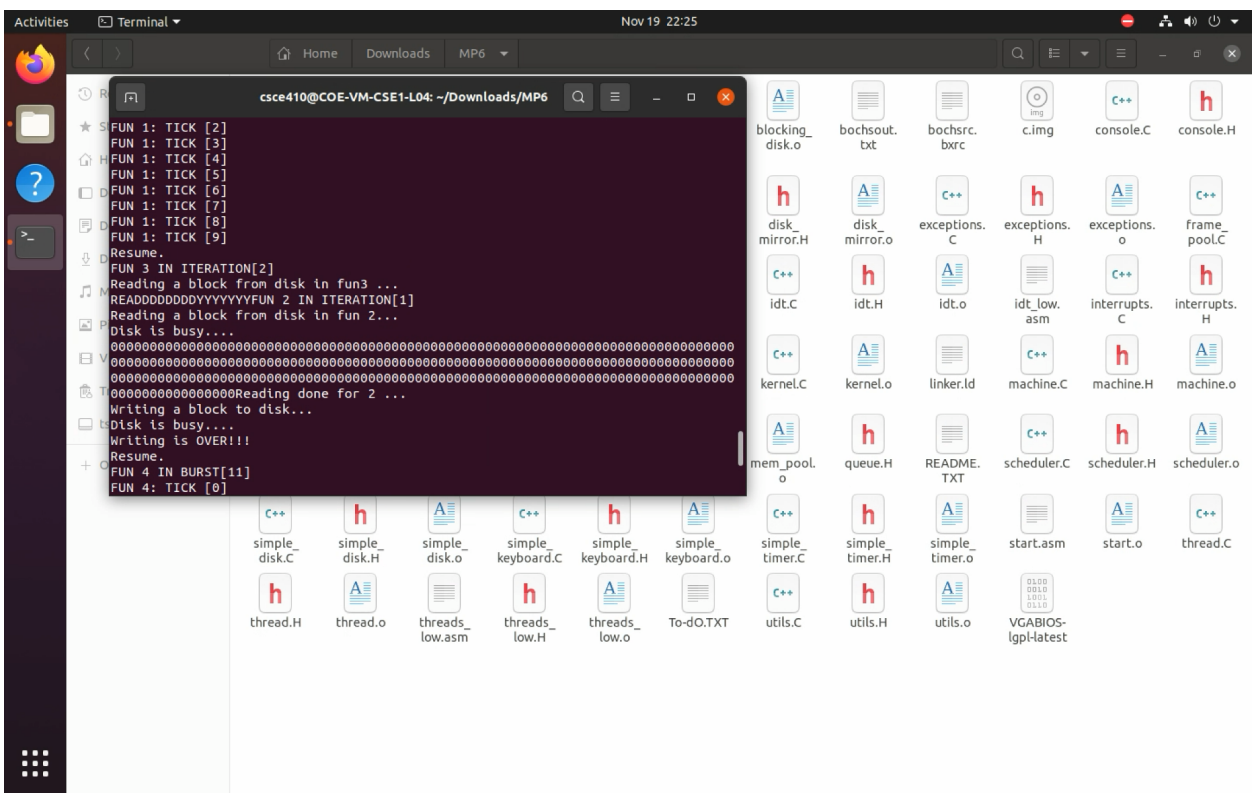
This is achieved by introducing a tracker variable which turns true/false when a disk operation is being carried out. With the help of this variable at a time only one thread is making use of the disk. A queue block_queue maintains list of threads attempting to perform disk operations and take care of race condition

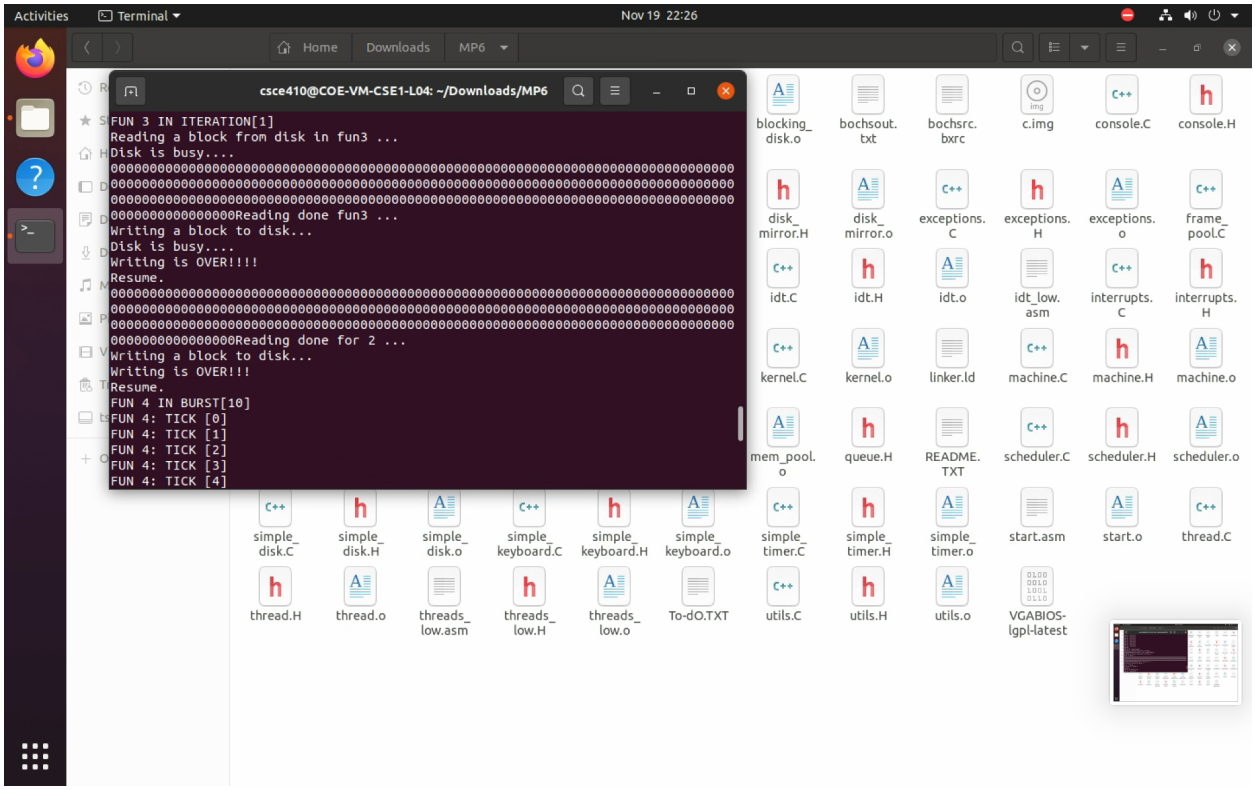
```
block_disk = false; // to check if the disk is blocked
```

All disk operations are carried out by locking the disk using the above variable

```
  void BlockingDisk::read(unsigned long _block_no, unsigned char *_buf)
  {
    if (block_disk == false)  //if disk is not blocked, no disk operation is to
be performed i.e. skipped
    {
      block_disk = true;
      SimpleDisk::issue_operation(DISK_OPERATION::READ, _block_no);
      if (!is_ready())
      {
        Thread *t = Thread::CurrentThread();
        //Console::puts("Disk is not ready so adding to block queue......\n");
        block_queue->enqueue(t);
        block_queue_size++;
        SYSTEM_SCHEDULER->yield();
        //Console::puts("Lets move to next thread......\n");
      }
      int i;
      unsigned short tmpw;
      for (i = 0; i < 256; i++)
      {
        tmpw = Machine::inportw(0x1F0);
        _buf[i * 2] = (unsigned char)tmpw;
        _buf[i * 2 + 1] = (unsigned char)(tmpw >> 8);
      }
      block_disk = false;
      //Console::putch('\n');
    }
    else{
      Console::puts("Disk is busy....\n");
    }
  }
void BlockingDisk::write(unsigned long _block_no, unsigned char *_buf)
{
  if (block_disk == false) //if disk is not blocked, no disk operation is to be
performed i.e. skipped
  {
    block_disk = true;
    SimpleDisk::issue_operation(DISK_OPERATION::WRITE, _block_no);
    if (!is_ready())
    {
      Thread *t = Thread::CurrentThread();
```

```
      block_queue->enqueue(t);
      Block_queue_size++;  -> Add threads to block_queue for performing disk
operations
      SYSTEM_SCHEDULER->yield();
    }
    int i;
    unsigned short tmpw;
    for (i = 0; i < 256; i++)
    {
      tmpw = _buf[2 * i] | (_buf[2 * i + 1] << 8);
      Machine::outportw(0x1F0, tmpw);
    }
    block_disk = false;
  }
  else{
    Console::puts("Disk is busy....\n");
  }
}
```






## Testing

`kernel.C`

1. `#define DISK_MIRROR` -> Uncomment this to perform disk mirroring (2)
2. `#define _ENABLE_INTERRUPT` -> uncomment this part to enable interrupt for concurrency feature (3)
3. Replace fun3 with following code to test multiple threads performing disk operation i.,e thread safe system (4&5)

```
void fun3()
{
    Console::puts("THREAD: ");
    Console::puti(Thread::CurrentThread()->ThreadId());
    Console::puts("\n");
    Console::puts("FUN 3 INVOKED!\n");
    unsigned char buf[DISK_BLOCK_SIZE];
    int read_block = 1;
    int write_block = 0;
    for (int j = 0;; j++)
    {
```

```
        Console::puts("FUN 3 IN ITERATION[");
        Console::puti(j);
        Console::puts("]\n");
        /* -- Read */
        Console::puts("Reading a block from disk in fun3 ...\n");
        SYSTEM_DISK->read(read_block, buf);
        // Console::puts("Now READING in FUN3...........\n");
        /* -- Display */
        int i;
        for (i = 0; i < DISK_BLOCK_SIZE; i++)
        {
            Console::puti(buf[i]);
        }
        Console::puts("Reading done fun3 ...\n");
        Console::puts("Writing a block to disk...\n");
        SYSTEM_DISK->write(write_block, buf);
        Console::puts("Writing is OVER!!!!\n");
        /* -- Move to next block */
        write_block = read_block;
        read_block = (read_block + 1) % 10;
        /* -- Give up the CPU */
        pass_on_CPU(thread4);
    }
}
```