

# 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 MIRRORDDISK 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:

```
csce410@csce410-VirtualBox: ~/Documents/VamsiTallam_CSCE611/mp6
Please choose one: [6]
000000000000i[      ] installing x module as the Bochs GUI
000000000000i[      ] using log file bochsout.txt
Installed exception handler at ISR <0>
Allocating Memory Pool... done
Installed interrupt handler at IRQ <0>
Constructed Scheduler.
Constructed Blocking Disk
Constructed RAID_1_DISK
Implementing Disk Locks Using Test And Set
Hello World!
CREATING THREAD 1...
esp = <2098160>
done
DONE
CREATING THREAD 2...esp = <2099208>
done
DONE
CREATING THREAD 3...esp = <2100256>
done
DONE
CREATING THREAD 4...esp = <2101304>
done
DONE
STARTING THREAD 1 ...
THREAD: 0
FUN 1 INVOKED!
FUN 1 IN ITERATION[0]
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]
THREAD: 1
FUN 2 INVOKED!
FUN 2 IN ITERATION[0]
Reading a block from disk...
Disk is Locked
Yielding CPU for I/O
THREAD: 2
FUN 3 INVOKED!
FUN 3 IN BURST[0]
FUN 3: TICK [0]
```

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

### Elimination of Busy waiting while read operation:

```
FUN 1: TICK [6]
FUN 1: TICK [7]
FUN 1: TICK [8]
FUN 1: TICK [9]
THREAD: 1
FUN 2 INVOKED!
FUN 2 IN ITERATION[0]
Reading a block from disk...
Disk is Locked
Yielding CPU for I/O
THREAD: 2
FUN 3 INVOKED!
```

[illegible]

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

[illegible]

Once the disk is free, we start writing

```

Writing a block to disk...
Disk is Locked
Yielding CPU for I/O
FUN 3 IN BURST[1]
FUN 3: TICK [0]
FUN 3: TICK [1]
FUN 3: TICK [2]

```

Once written the disk is unlocked

```
csce410@csce410-VirtualBox: ~/Documents/VamsiTallam_CSCE611/mp6
Disk is Locked
Writing Done to RAID_1 DISK
Writing Done to Blocking DISK
Disk is unlocked
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]
FUN 4 IN BURST[2]
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[3]
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[2]
Reading a block from disk...
```

### 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:

[illegible]

Write Operation for mirroring:

```
Writing Done to RAID_1 DISK
Writing Done to Blocking DISK
Disk is unlocked
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]
FUN 4 IN BURST[2]
FUN 4: TICK [0]
FUN 4: TICK [1]
FUN 4: TICK [2]
FUN 4: TICK [3]
```

## 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.

```
FUN 2 INVOKED!  
FUN 2 IN ITERATION[0]  
Reading a block from disk...  
Disk is Locked  
Yielding CPU for I/O  
THREAD: 2  
FUN 3 INVOKED!  
FUN 3 IN BURST[0]  
FUN 3: TICK [0]
```

[illegible]

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

```
Disk is Locked
Writing Done to RAID_1 DISK
Writing Done to Blocking DISK
Disk is unlocked
```

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();
```