`

The github link is https://github.tamu.edu/kausht14/OS_MP6

Design Steps:

a) Run make clean to remove all bin and locked files
b) Run make.
c) All the bonus sections of mirroring, interrupts and concurrency have been implemented and clearly documented how it is performed

For BLOCKING_DISK.C

1) As an initial step, scheduler.C , .H and thread.C and thread.H that are implemented with the help of interrupts have been included from MP5
2) The important function to eliminate busy waiting is wait_until_ready() and it is modified in a way that the thread which is waiting for disk and finds it busy yields the CPU. So extern Scheduler* SYSTEM_SCHEDULER is made use of in blocking_disk.C . The read() and write() functions have been modified by adding the functionality of new wait_until_ready(). Also the bonus section mirroring is also perfomed .Read and write are performed from either of the disks whichever is available
3) Also a disk queue is maintained and the threads that are waiting for the disk have been enqueued and dequeue in the order of their queueing thereby preventing starvation and allowing for fairness to disk access.
4) The issue_operation() has not been modified as said by the Professor and the low level functionality of the issue operation is not altered

FOR BLOCKING_DISK.H

5) The private variables have been defined and also a queue of blocked threads has been maintained for yielding the CPU whenever they request a disk operation. Also the is_ready() function has been used as similar to that of simple_disk.C
6) The scheduler code is brought from my previous MP and is included to handle Scheduling
7) Everytime a thread is added, yielded, resumed, interrupts are checked and properly handled.

For Kernel.C

8) The necessary header files were included and the USES_SCHEDULER is uncommented to make use of scheduler
9) Now SYSTEM_DISK is modified to point to BlockingDisk(MASTER,….) instead of SimpleDisk
10) Also the overloaded delete(void* , size_t) has been added to kernel.c to get rid of linker issues while deleting a queue item of threads when it yields the CPU.
11) Console::putch() is changed to Console::puti() to output the values

FOR MAKEFILE

12) Make file has been changed to include the .C, .H, .O files of scheduler, blocking_disk, mirroring_disk

13) also the remove list is updated to remove the lock generated files as the bonus part of locking is implemented. These locked files can be cleaned by running a make clean

BONUS 1: MIRRORING

For MIRRORING_DISK.C

14) The MIRROREDDISK class is publicly derived from simple_disk and a few functions like the readToMirror() and writeToMirror() have been defined for issuing low level operations
15) The read() and write() functions are functionally same as that of BlockingDisk.
16) The MirroredDisk is initialised from the BlockingDisk constructor by passing DISK_ID::DEPENDENT as the argument and also the wait_until_ready() functionality is same as that of the blocking_disk();

For MIRRORING_DISK.H

17) The private members and the functions are defined same as that of blocking_disk.C and few other needful functions are defined for local use that are used to issue low level issue_operation() for writing and reading to a mirror disk.
18) The MirroredDisk is made use of like DEPENDENT.

BONUS 2 INTERRUPTS HANDLING:

19) The interrupts are properly handled by properly enabling and disbling interrupts by checking
    if(Machine::interrupts_enabled())
    Machine::disable_interrupts()

    If(Machine::interrupts_enabled()){
    Machine::enable_interrupts();

    The above statements have been very helpful in order to check for interrupts and properly enable and disable them while yielding, adding and resume the executions of threads and also for disk waiting operations in the wait_until_ready() functions

BONUS 3 & 4 CONCURRENCY and MUTUAL EXCLUSION, ATOMIC TRANSACTIONS

20) The Design for BONUS 3

a) The concurrency is achived by enforcing mutual exclusion that is no thread is allowed to enter a critical section when someother thread is executiong
b) This is enforced by TESTand SET lock implementation by testing and then setting if the lock is available.
c) Every time a thread wishes to enter a critical section it waits for the lock. If lock is available, it acquires the lock and enters critical section else waits for the lock to be released

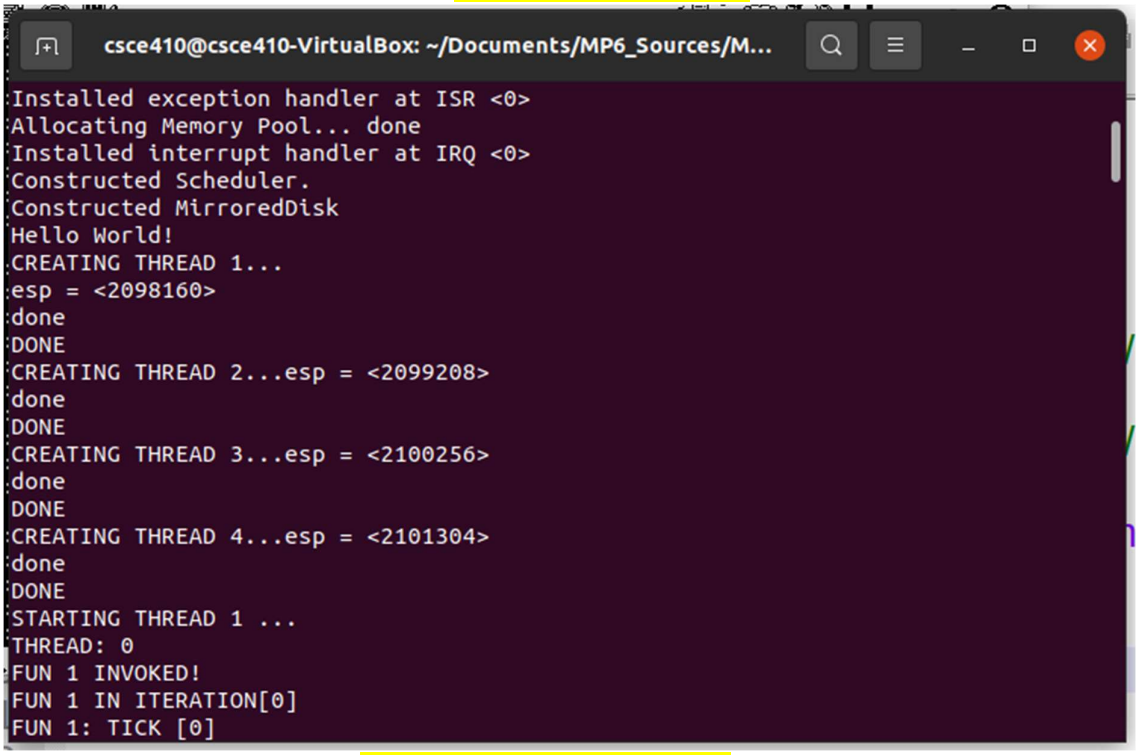
BONUS 4:

IMPLEMENTATION OF BONUS 4:

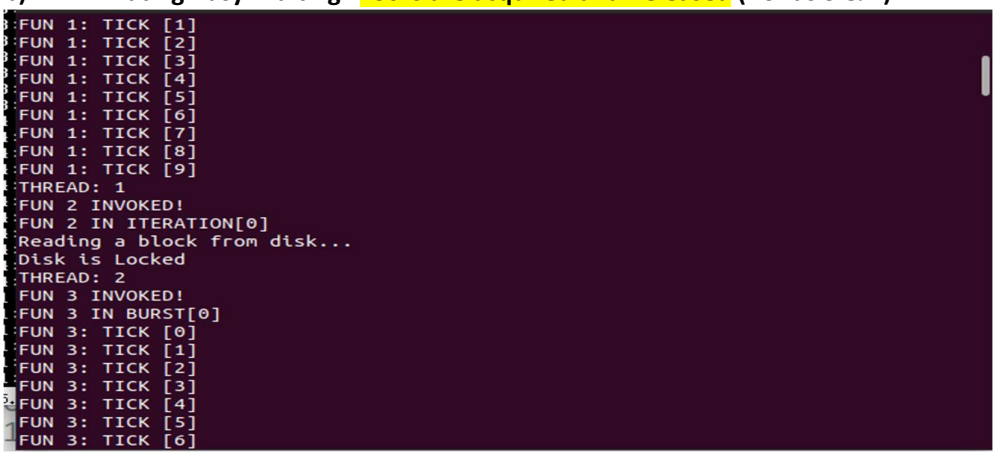
21) The disk operations are atomic
22) The concurrency for disk writing is achieved by TestAndSet Locks
23) Whenever a disk operation is issued , the lock is acquired and after the writing is done, the lock is released
24) The Locking is implemented by the standard Test and Set funtionalisty where in we wait for test the current lock and then wait for the lock to be released and acquire it and then lock it and continue with our operation and release the lock once done

RESULTS:

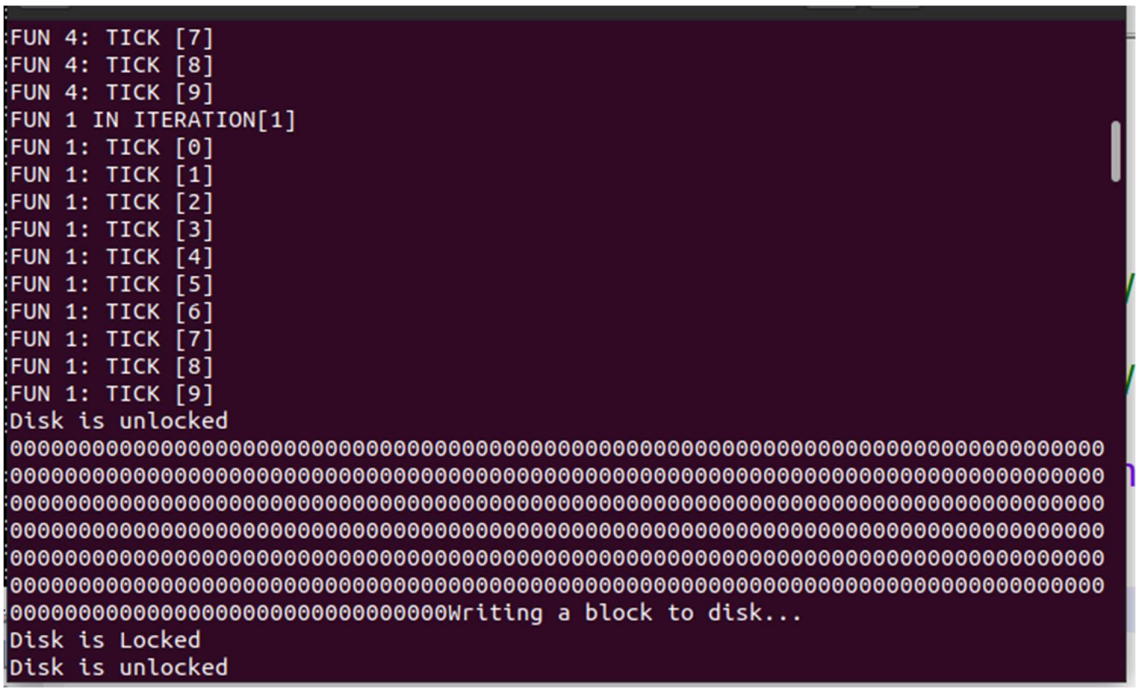
a) Initialisation of Disks: Blocking and Mirroring Disks(BONUS 1)



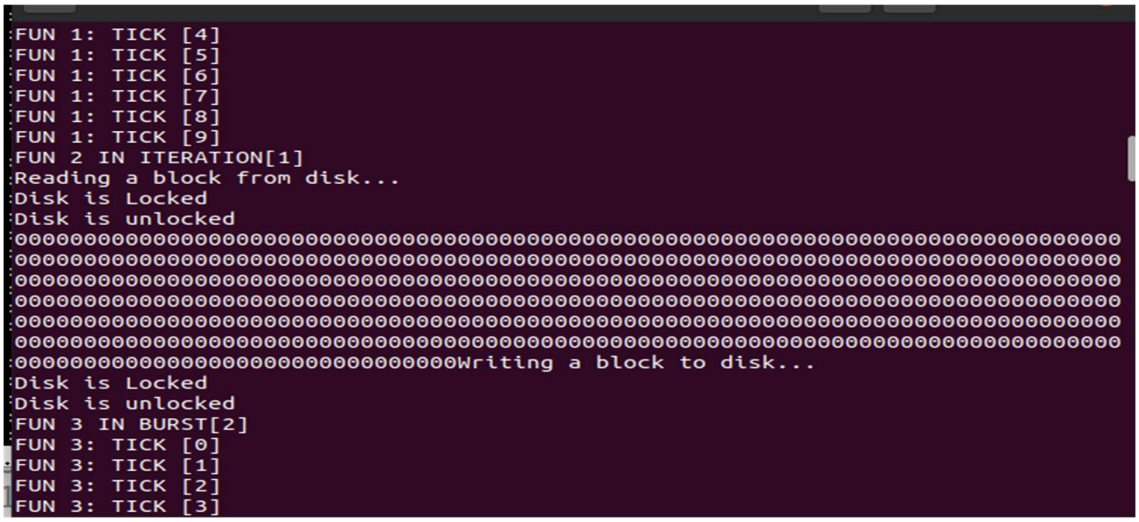
b) Eliminating Busy Waiting . Locks are acquired and Released (Bonus 3 & 4)



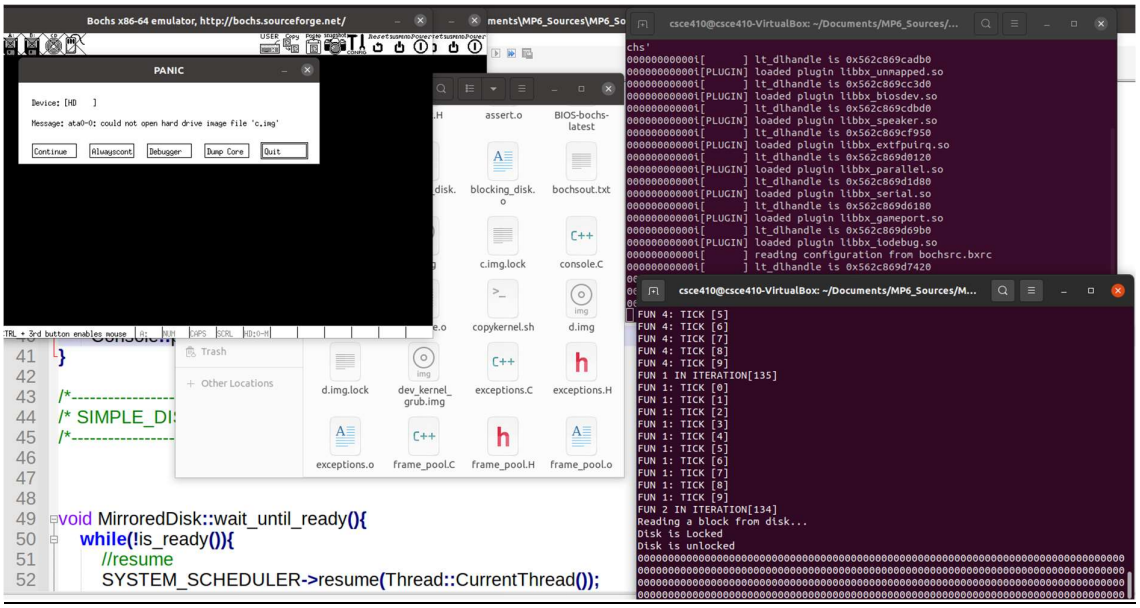
Writing to a disk eliminating Busy Loops and Locked and Unlocked (Bonus 3&4)



For the subsequent disk operations



The disk operations are made atomic by locking. No concurrent disk operation is allowed to perform when an ongoing disk read or write takes place as shown in the screenshot , concurrent disk read/write operation throws a panic message and clearly we can see from the explorer that the c.img and d.img have generated c.img.lock and d.img.lock files



For interrupt handling:

The interrupt handling mechanism is same as that implemented in MP5

```
void Scheduler::yield() {
    // check if interrupts are enabled and properly disable them to yield the CPU
    if(Machine::interrupts_enabled())
    Machine::disable_interrupts();

    // Get the thread next inn queue and make it head
```