CSCE611: Operating System
MP6: Primitive Disk Device Driver

Name: Rahul Ravi Kadam
UIN: 834005554

Assigned Tasks

Main: Completed
Bonus Option 1: Attempted

System Design

The goal of this machine problem is to add a layer on top of a simple device driver to support non-blocking read and write operations as the basic implementation, but without busy waiting in the device driver code. The user should be able to call 'read' and 'write' operations without worrying that the call may either return prematurely or tie up the entire system waiting for the device to return.

Blocking Disk Implementation:

A class 'Queue' is introduced to handle operations on the ready queue and blocked threads queue. The queue is implemented as a linked list of thread objects. The class 'Queue' declares constructors and public functions to add (enqueue) and remove (dequeue) threads from a queue. To perform the enqueue operation, we first check if there exists a thread in the queue already. If a thread exists in the queue, we traverse to the end of the queue and append the new thread. To perform the dequeue operation, the thread at the top of the queue is removed and the pointer is updated to point to the next thread in the queue.

In this machine problem, we implement a 'NonBlockingDisk' device driver which inherits from 'SimpleDisk' and supports I/O operations (read and write operations) without busy waiting in the device driver code. Using the scheduler, threads yield the CPU if the disk is not ready for I/O operations.

When a thread requests a disk I/O operation, it triggers the 'NonBlockingDisk::wait_until_ready()' method which enqueues the current thread into a separate 'blocked_queue' and yields the CPU to the next thread in the ready queue.

When a thread completes execution and yields, it triggers the 'Scheduler::yield()' method. In this method, we check if the disk is ready and that there is at least one thread present in the blocked queue. If there exists a blocked thread in the blocked queue and the disk is ready, then we dequeue the thread at the top of the blocked queue and dispatch to it immediately to complete the I/O operation. If there are no threads present in the blocked queue or if the disk is not ready, then we dequeue the thread at the top of the ready queue (which contains threads which do not perform I/O operations and do not get blocked) and dispatch to it as usual. (Note: Changes were made in 'simple_disk.C' to simulate disk I/O delay)

In this manner, if the driver is not ready, the scheduler yields the CPU so that other threads can be executed in the meantime and we avoid busy waiting in the device driver.

OPTION 1: Design of a thread-safe disk system: The current design of device drivers handles the scenario of multi-threading on a uni-core system. Since in this machine problem we are given only one processor, multiprogramming is achieved solely by multi-threading. When multiple threads

perform the Read/Write operations, the threads are added to the blocked queue and yield the CPU. When doing so, the interrupts are enabled and disabled in the critical section, to ensure mutual exclusion.

For a multiprocessor system, to prevent race conditions between multiple CPUs, along with the above, protection need to be implemented using thread-safe queue and I/O device locking. The thread-safe queue will ensure multiple process don't incorrectly update the queue and the lock on the I/O will make sure that the system device is not given up if busy. However, this implementation can't be tested as we only have one processor in the current setup.

List of files modified:

The following files were modified:

1. scheduler.H
2. scheduler.C
3. nonblocking_disk.H
4. nonblocking_disk.C
5. queue.H
6. thread.C
7. kernel.C
8. makefile

Code Description

1. scheduler.H : class Scheduler:

In class Scheduler, data structures and functions for scheduling of kernel-level threads and management of the ready queue are declared.

- Queue ready_queue - Queue data object for handling the ready queue of threads
- int qsize - Variable to track the number of threads waiting in the ready queue

2. scheduler.C : Scheduler(): This method is the constructor for class Scheduler. The ready queue size is initialized to zero.

3. scheduler.C : Scheduler :: yield():

This method is used to pre-empt the currently running thread and give up the CPU. A new thread is selected for CPU time by dequeuing the top element in the ready queue and dispatching to it. Moreover, interrupts are disabled before the start of any operations on the ready queue, and interrupts are re-enabled after completing operations on the ready queue. In order to handle non-blocking read/write operations without busy waiting, modifications were made to the 'yield' method. Using the 'check_blocked_thread_in_queue' method, we check if the disk is ready and that at least one thread is present in the blocked threads queue. If this condition is true, we dequeue the thread at the top of the blocked threads queue (threads which block due to I/O operations) and dispatch to it immediately. If the condition is not true, we dequeue the top thread in the ready queue (threads which do not perform I/O operations) and dispatch to it.

4. scheduler.C : Scheduler :: resume(Thread * _thread):

This method is used to add a thread to the ready queue of the scheduler. This method is called for threads that were waiting for an event to happen, or that have to give up the CPU in response to a pre-emption. Interrupts are disabled before performing any operations on the ready queue, and interrupts are re-enabled after completing operations on the ready queue.

5. scheduler.C : Scheduler :: add(Thread * _thread):

This method is used to add a thread to the ready queue of the scheduler. The thread is made runnable by the scheduler. This method is called on thread creation. Interrupts are disabled before performing any operations on the ready queue, and interrupts are re-enabled after completing operations on the ready queue.

6. scheduler.C : Scheduler :: terminate(Thread * _thread):

This method is used to remove a thread from the ready queue. We iterate through the ready queue; dequeue each thread and compare the Thread ID. If the Thread ID does not match with the thread ID of the thread to be terminated, then we enqueue the threads back into the ready queue. Interrupts are disabled before performing any operations on the ready queue, and interrupts are re-enabled after completing operations on the ready queue.

7. nonblocking_disk.H : class NonBlockingDisk:

In class NonBlockingDisk, data structures and functions for scheduling of kernel-level threads and management of the blocked queue are declared. It inherits from class SimpleDisk.

- Queue blocked_queue - Queue data object for handling the queue of blocked threads
- int blocked_queue_size - Variable to track the number of threads in the blocked queue

8. nonblocking_disk.C : NonBlockingDisk():

This method is the constructor for class NonBlockingDisk. The blocked threads queue is initialized and the blocked threads queue size is set to zero.

9. nonblocking_disk.C : get_top_thread():

This method returns the blocked thread located at the top of the blocked queue by performing a ‘dequeue’ operation.

10. nonblocking_disk.C : wait_until_ready():

This method adds the current running thread to the end of the blocked queue by performing an ‘enqueue’ operation. The thread then yields the CPU to the next thread.

11. nonblocking_disk.C : check_blocked_thread_in_queue():

This method checks if the disk is ready to transfer data and if threads exist in the blocked queue. If the disk is ready to transfer data and threads exist in the blocked queue, a Boolean true is returned. Otherwise, a Boolean false is returned.

12. queue.H : class Queue:

In class Queue, data structures and functions for managing the ready queue and blocked queue are defined:

- Thread * thread - Pointer to the thread at the top of the queue
- Queue * next - Pointer to next Queue type object (next thread in the queue)
- Queue() - Constructor for initial setup of the variables
- Queue(Thread * new_thread) - Constructor for setting up subsequent threads
- Void enqueue(Thread * new_thread) - Method to add a thread to the end of the queue
- Thread * dequeue() - Function to remove the thread at the top of the queue and point to the next thread

13. thread.C : thread_shutdown():

This method is used to terminate a thread by releasing the memory and other resources held by the thread. The below changes were made to handle threads with terminating thread functions. The currently running thread is first

terminated. Next, memory is freed up using the 'delete' operation. Finally, the 'yield' method is called to give up CPU and select the next thread in the ready queue.

14. thread.C : thread_start():

Changes were made in this method to enable interrupts at the start of the thread using the 'enable_interrupts' method.

15. kernel.C :

The following changes were made in kernel.C for the purpose of testing:

- The macro '_USES_SCHEDULER' was uncommented to enable support for the scheduler
- Replaced 'SimpleDisk' objects with 'NonBlockingDisk' objects to make use of NonBlockingDisk device
- Increased stack size allocated for Thread 2 from 1024 bytes to 4096 bytes to resolve stack overflow bug when interrupts are enabled

16. makefile :

Changes were made in the makefile to include 'scheduler.C', 'scheduler.H' and 'queue.H' files.

Testcase and Outputs:

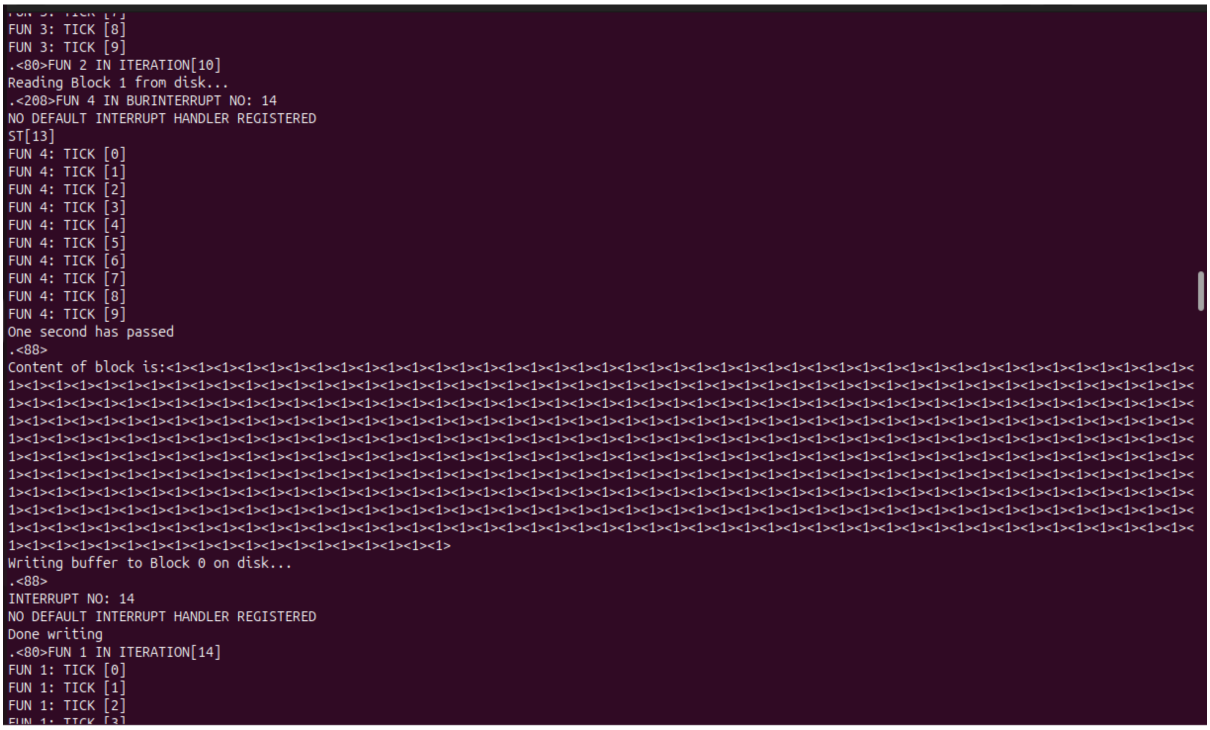
Testcase: Single thread (Thread 2) issues disk I/O operations; while other threads run regular non-terminating functions

Expected Result: Thread 2 yields CPU to another thread on initiating Disk I/O operation. Busy waiting is avoided. Disk I/O operation is completed.

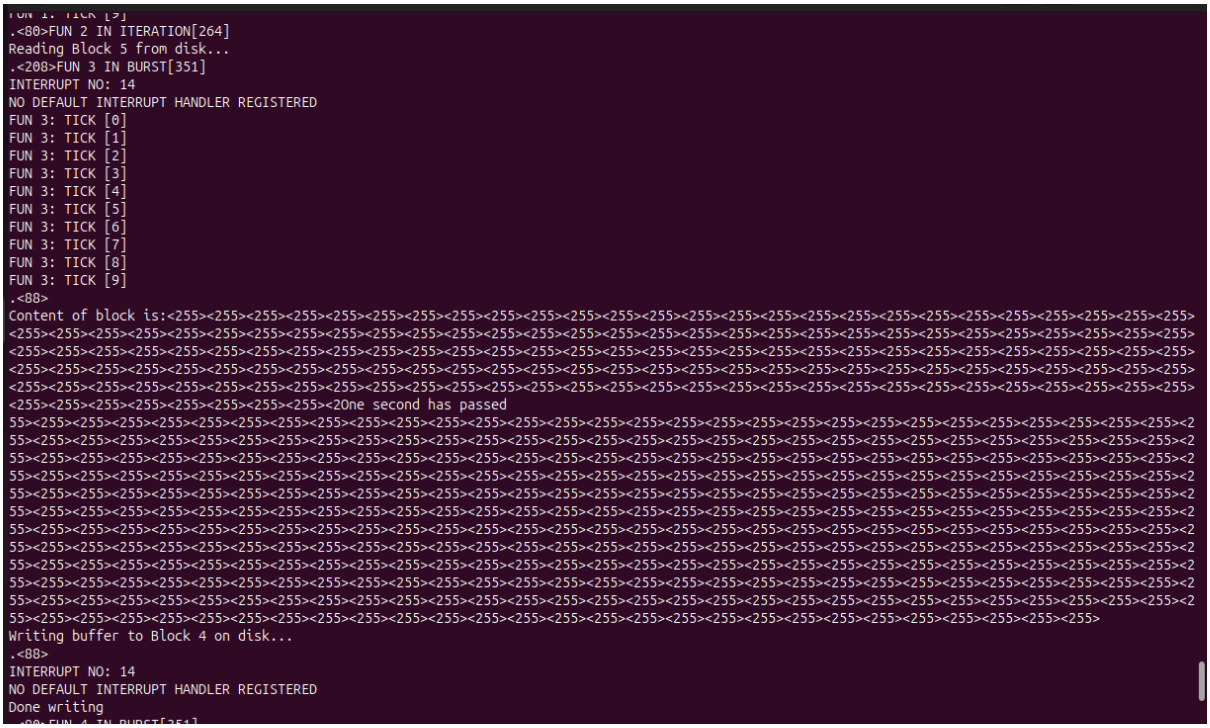
Outputs:

```
rahul-611@rahul-611-VirtualBox:~/Desktop/MP6_Sources/MP6_Sources$ make run
Linux
qemu-system-x86_64 -kernel kernel.bin -serial stdio \
-device piix3-ide,id=ide -drive id=disk,file=c.img,format=raw,if=none -device ide-hd,drive=disk,bus=ide.0
Installed exception handler at ISR <0>
Allocating Memory Pool... done
Installed interrupt handler at IRQ <0>
Constructed Scheduler.
Hello World!
CREATING THREAD 1...
esp = <2101212>
done
DONE
CREATING THREAD 2...esp = <2105332>
done
DONE
CREATING THREAD 3...esp = <2109452>
done
DONE
CREATING THREAD 4...esp = <2113572>
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]
```

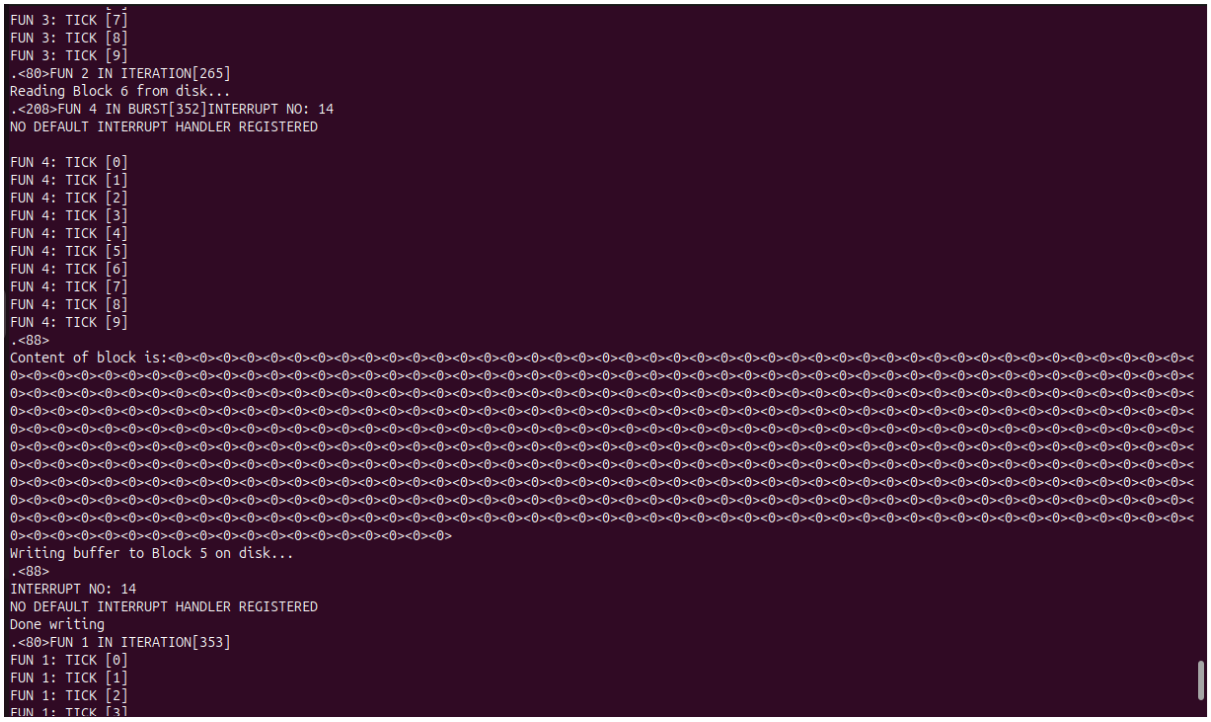
```
FUN 1: TICK [6]
FUN 1: TICK [7]
FUN 1: TICK [8]
FUN 1: TICK [9]
.<80>FUN 2 IN ITERATION[9]
Reading Block 0 from disk...
.<208>FUN 3 IN BURST[12]
INTERRUPT NO: 14
NO DEFAULT INTERRUPT HANDLER REGISTERED
FUN 3: TICK [0]
FUN 3: TICK [1]
FUN 3: TICK [2]
FUN 3: TICK [3]
FUN 3: TICK [4]
FUN 3: TICK [5]
FUN 3: TICK [6]
FUN 3: TICK [7]
FUN 3: TICK [8]
FUN 3: TICK [9]
.<88>
Content of block is:<0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><
0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><
0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><
0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><
0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><
0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><
0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><
0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><
0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><
0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><
0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0>
Writing buffer to Block 9 on disk...
.<88>
INTERRUPT NO: 14
NO DEFAULT INTERRUPT HANDLER REGISTERED
Done writing
.<80>FUN 4 IN BURST[12]
FUN 4: TICK [0]
FUN 4: TICK [1]
FUN 4: TICK [2]
```



```
FUN 3: TICK [7]
FUN 3: TICK [8]
FUN 3: TICK [9]
.<80>FUN 2 IN ITERATION[10]
Reading Block 1 from disk...
.<208>FUN 4 IN BURINTERRUPT NO: 14
NO DEFAULT INTERRUPT HANDLER REGISTERED
ST[13]
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]
One second has passed
.<88>
Content of block is:<1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><
1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><
1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><
1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><
1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><
1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><
1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><
1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><
1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><
1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><
1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1><1>
Writing buffer to Block 0 on disk...
.<88>
INTERRUPT NO: 14
NO DEFAULT INTERRUPT HANDLER REGISTERED
Done writing
.<80>FUN 1 IN ITERATION[14]
FUN 1: TICK [0]
FUN 1: TICK [1]
FUN 1: TICK [2]
FUN 1: TICK [3]
```



```
FUN 1: TICK [9]
.<80>FUN 2 IN ITERATION[264]
Reading Block 5 from disk...
.<208>FUN 3 IN BURST[351]
INTERRUPT NO: 14
NO DEFAULT INTERRUPT HANDLER REGISTERED
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]
.<88>
Content of block is:<255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255>
<255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255>
<255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255>
<255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255>
<255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255>
<255><255><255><255><255><255><255><255><2One second has passed
55><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><2
55><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><2
55><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><2
55><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><2
55><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><2
55><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><2
55><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><2
55><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><2
55><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><2
55><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><2
55><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><2
55><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255><255>
Writing buffer to Block 4 on disk...
.<88>
INTERRUPT NO: 14
NO DEFAULT INTERRUPT HANDLER REGISTERED
Done writing
.<80>FUN 4 IN BURST[351]
```



Single thread (Thread 2) issues disk I/O operations; Disk operations complete successfully

Disk read operation is initiated, Thread 2 yields CPU and gives control to another thread for execution. After a while, control is returned to Thread 2 and Disk Write operation is completed.

Test Successful!