Machine Problem 6 : Simple Disk Device Driver

Design :

In this programming assignment, we implement the I/O operations – blocking read and write without busy waiting in the device driver code. Here we implement a device called BlockingDisk, which is derived from the existing low-level device SimpleDisk. Since the read and write methods are stuck in busy waiting in method wait_until_ready, we override it with a new implementation in class Blocking disk. This method puts the thread in waiting kernel scheduler's queue and passes the CPU onto another thread and so the CPU is not blocked. The scheduler used is similar to the FIFO one used in the previous MP. The class "Queue" has this property of FIFO queue. It will maintain a list of Thread structures and the next pointer for thread. This class provides two basic queue operations which are enqueue and dequeue. For enqueuing a thread object in the ready queue, we just check if there is an existing member in the queue already or not, if not we update the next pointer. For dequeue operation, we take the top thread and return. The scheduler class declares a queue object and also has a size member. The yield api will first check if there is any thread available to switch and then dequeue the thread from queue. The resume and add in the scheduler are very similar where we enqueue the given thread to the queue. The terminate api loops through the queue and finds the thread with same thread id and then delete it.

Functions used in blocking_disk.C/H :

- **Constructor :**

```
size = 0;
this->disk_queue =  new Queue();
```

- **wait_until_ready() –** Main function used to implement read/write without busy waiting

```
if (!is_ready()) {                          // checking if ready
Thread *cur_thread = Thread::CurrentThread(); // store CurrentThread in a variable
this->disk_enqueue(cur_thread);             // add this to the queue
SYSTEM_SCHEDULER->yield();                  // yield the CPU
}
```

- **disk_enqueue()**

```
this->disk_queue->enqueue(thread);
size++;
```

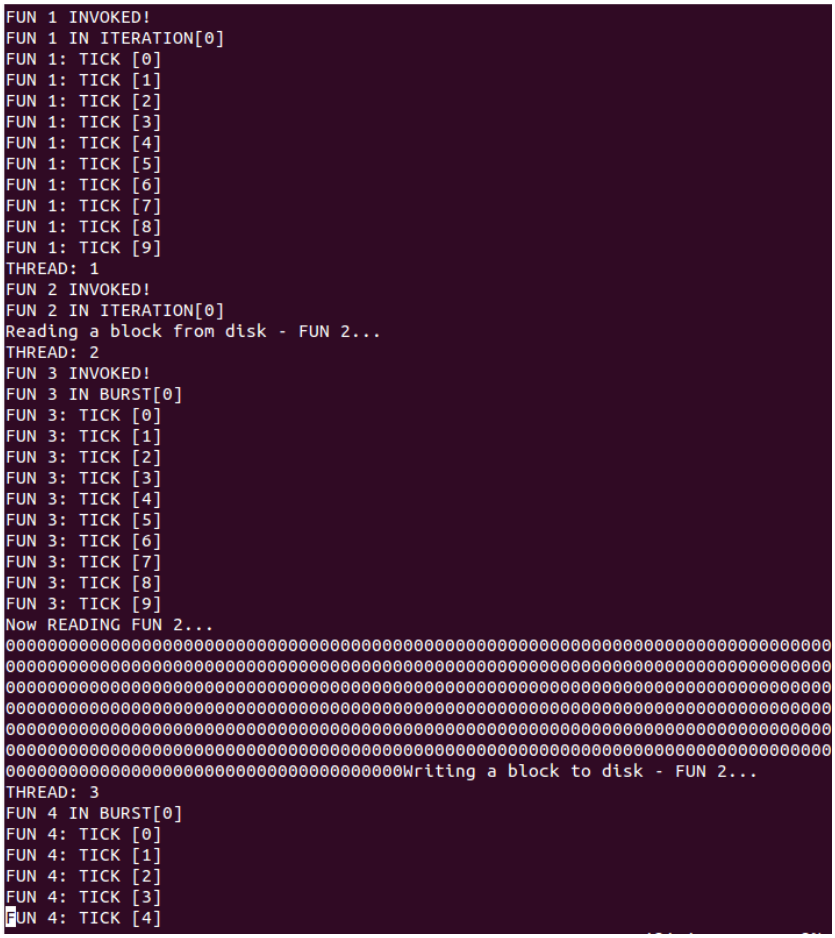
- **is_ready()**

```
SimpleDisk::is_ready();    // same as the implementation in SimpleDisk class
```

- The read and write functions are same as the ones in SimpleDisk class.

Testing :

The below screenshot shows how it yields the cpu during read operation and thus avoiding busy waiting. Once the reading starts, it switches over to the next thread and continues the same when reading and current thread is done.



Note – Putch was replaced by puti to display the buffer that was read.

The entire output is saved in the results.txt file in the github link.

Modified Files :

- blocking_disk.C/H
- scheduler.C/H
- kernel.C
- makefile
- console.C (for printing logs into console)
- boschsrc.bxrx (for printing logs into console)