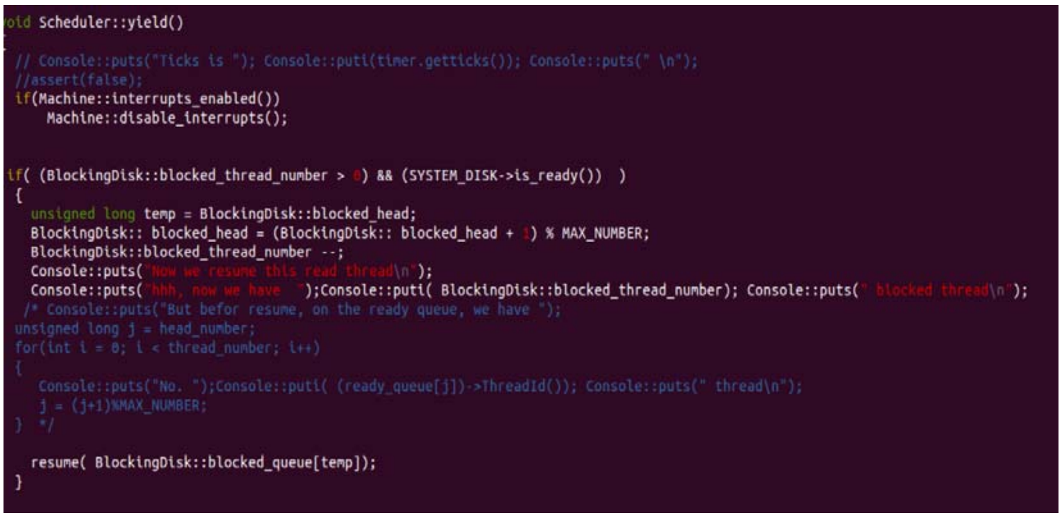
Design of MP6

Basic Assignment

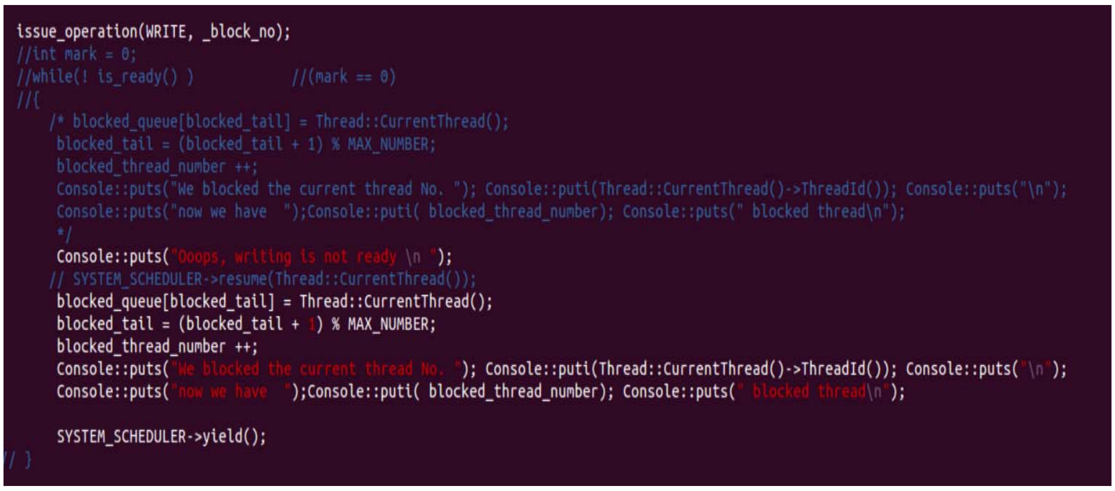
In this MP, I only implemented the basic required disk function. What I submitted in the basic assignment folder include 8 files. In order to realize FIFO, I added *scheduler.H, scheduler.C, and modified makefile*. And I also modified *simple_disk.C* and *simple_disk.H* in order to derive blocking_disk in a more convenient way. Also, in *kernel.C*, I enabled *_USES_SCHEDULER_* and changed *SYSTEM_DISK* to the pointer to the BlockingDisk object instead of the pointer to the SimpleDisk Object. So directly link these 8 files with other files provided on ecampus, we can realize the basic assignment of MP6. Now let's begin.
In my submission, I set a blocked queue in the class of BlockingDisk. And I used the FIFO scheduler from the previous MP5, instead of taking time on implementing new scheduler with the interrupts currency disk system. However, in yield function, I made a little modification as is shown in Figure 1. Every time we yield the CPU, we check the status of registers of our device and decide whether resume the thread from the blocked queue in BlockingdDisk to the ready queue in the scheduler.



```
oid Scheduler::yield()

 // Console::puts("Ticks is "); Console::puti(timer.getticks()); Console::puts(" \n");
 //assert(false);
 if(Machine::interrupts_enabled())
     Machine::disable_interrupts();


if( (BlockingDisk::blocked_thread_number > 0) && (SYSTEM_DISK->is_ready())  )
 {
   unsigned long temp = BlockingDisk::blocked_head;
   BlockingDisk:: blocked_head = (BlockingDisk:: blocked_head + 1) % MAX_NUMBER;
   BlockingDisk::blocked_thread_number --;
   Console::puts("Now we resume this read thread\n");
   Console::puts("hhh, now we have  ");Console::puti( BlockingDisk::blocked_thread_number); Console::puts(" blocked thread\n");
  /* Console::puts("But befor resume, on the ready queue, we have ");
 unsigned long j = head_number;
 for(int i = 0; i < thread_number; i++)
 {
    Console::puts("No. ");Console::puti( (ready_queue[j])->ThreadId()); Console::puts(" thread\n");
    j = (j+1)%MAX_NUMBER;
 }  */

   resume( BlockingDisk::blocked_queue[temp]);
 }
```

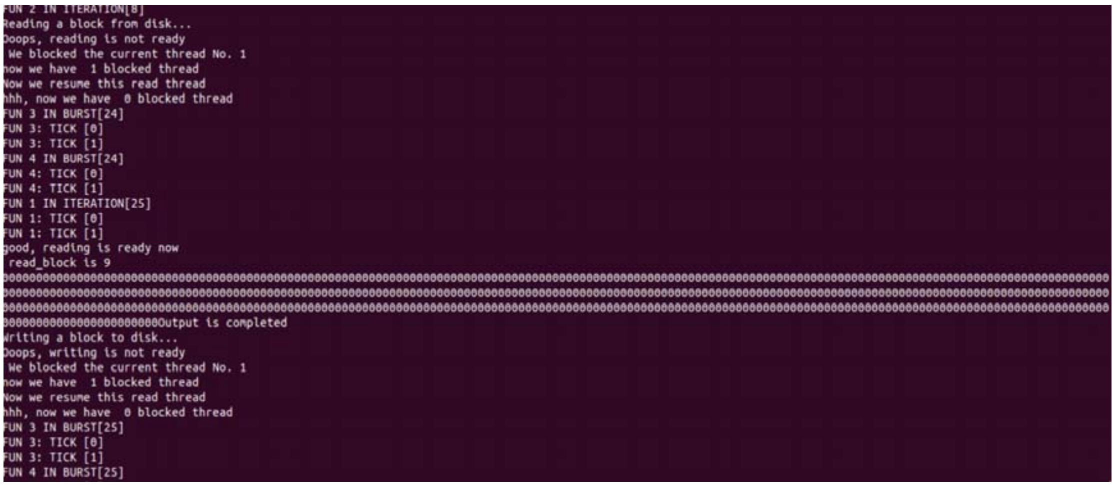
Figure 1

And the design of my BlockingDisk is quite simple. Instead of using busy waiting, we yield the CPU and dispatch to the next thread. The only tricky thing needed to present here is that Bochs really has a bad simulation of the write operation. In this condition, issue_operation (write, _block_no) always return very fast. So in order to simulate the real condition, I force it to yield the CPU, as is shown in Figure 2. And our final result is shown in Figure 3. And I changed the output of the other three thread functions to highlight the result of thread 2, which does not have any negative influence on our project.



```
issue_operation(WRITE, _block_no);
//int mark = 0;
//while(! is_ready() )            //(mark == 0)
//{
    /* blocked_queue[blocked_tail] = Thread::CurrentThread();
     blocked_tail = (blocked_tail + 1) % MAX_NUMBER;
     blocked_thread_number ++;
     Console::puts("We blocked the current thread No. "); Console::puti(Thread::CurrentThread()->ThreadId()); Console::puts("\n");
     Console::puts("now we have  ");Console::puti( blocked_thread_number); Console::puts(" blocked thread\n");
     */
     Console::puts("Ooops, writing is not ready \n ");
    // SYSTEM_SCHEDULER->resume(Thread::CurrentThread());
     blocked_queue[blocked_tail] = Thread::CurrentThread();
     blocked_tail = (blocked_tail + 1) % MAX_NUMBER;
     blocked_thread_number ++;
     Console::puts("We blocked the current thread No. "); Console::puti(Thread::CurrentThread()->ThreadId()); Console::puts("\n");
     Console::puts("now we have  ");Console::puti( blocked_thread_number); Console::puts(" blocked thread\n");

     SYSTEM_SCHEDULER->yield();
// }
```

Figure 2



```
FUN 2 IN ITERATION[8]
Reading a block from disk...
Ooops, reading is not ready
 We blocked the current thread No. 1
now we have  1 blocked thread
Now we resume this read thread
hhh, now we have  0 blocked thread
FUN 3 IN BURST[24]
FUN 3: TICK [0]
FUN 3: TICK [1]
FUN 4 IN BURST[24]
FUN 4: TICK [0]
FUN 4: TICK [1]
FUN 1 IN ITERATION[25]
FUN 1: TICK [0]
FUN 1: TICK [1]
good, reading is ready now
 read_block is 9
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000Output is completed
Writing a block to disk...
Ooops, writing is not ready
 We blocked the current thread No. 1
now we have  1 blocked thread
Now we resume this read thread
hhh, now we have  0 blocked thread
FUN 3 IN BURST[25]
FUN 3: TICK [0]
FUN 3: TICK [1]
FUN 4 IN BURST[25]
```

Figure 3