MP6: Primitive Disk Device Driver

Kai-Chih Huang
UIN: 333000021
CSCE611: Operating System

Assigned Tasks

Main Part: Finished
Bonus Option 1:
Bonus Option 2: Not Finished
Bonus Option 3: Not Finished
Bonus Option 4: Not Finished

System Design

The goal of this machine problem is to implement kernel-level device driver that will not hold up CPU when the disk is not available. Instead of busy waiting, I call the scheduler, that I have done in previous machine problem, to queue up current thread then give up CPU to the next thread in the queue. The thread will wait in a FIFO queue until next round to check if the disk is available.

Code Description

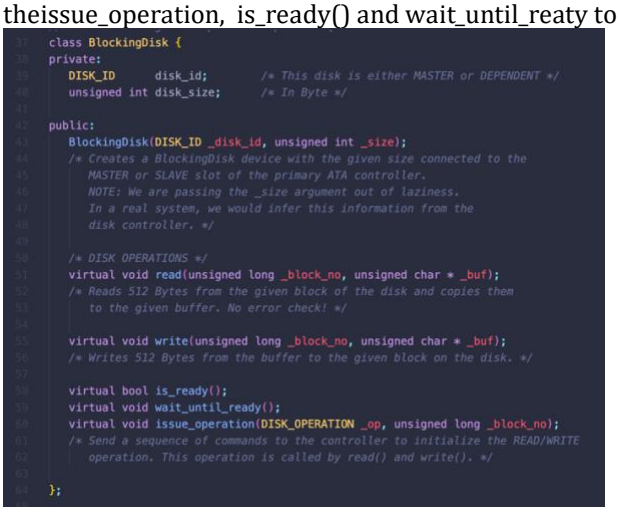
I added the scheduler.H, scheduler.C that I did on MP5, then changed makefile, blocking_disk.H, blocking_disk.C for this machine problem. To compile code, simply run following command lines under MP6_Sources directory:

```
$ make clean    // clean the old compile files before we compile

$ make          // compile files

$ ./copykernel.sh (if permission denied, try chmod u+x ./copykernel.sh then do it again)    //copy kernel

$ bochs -f bochsrc.bxrc     // run bochs
```

I will walk through the functions/methods defined in above files as follows.

blocking_disk.H

The design is basically the same as SimpleDisk. I rewrite the functions in the class and move theissue_operation, is_ready() and wait_until_reaty to public to prevent some unexpected bugs.

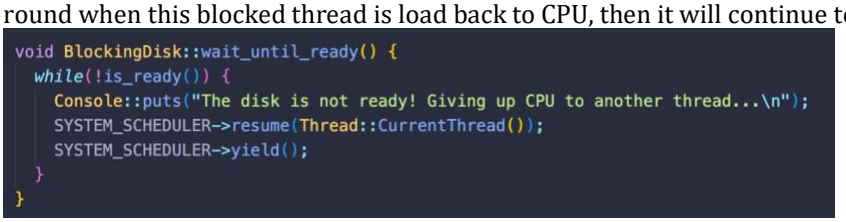


```
class BlockingDisk {
private:
   DISK_ID      disk_id;        /* This disk is either MASTER or DEPENDENT */
   unsigned int disk_size;      /* In Byte */

public:
   BlockingDisk(DISK_ID _disk_id, unsigned int _size);
   /* Creates a BlockingDisk device with the given size connected to the
      MASTER or SLAVE slot of the primary ATA controller.
      NOTE: We are passing the _size argument out of laziness.
      In a real system, we would infer this information from the
      disk controller. */

   /* DISK OPERATIONS */
   virtual void read(unsigned long _block_no, unsigned char * _buf);
   /* Reads 512 Bytes from the given block of the disk and copies them
      to the given buffer. No error check! */

   virtual void write(unsigned long _block_no, unsigned char * _buf);
   /* Writes 512 Bytes from the buffer to the given block on the disk. */

   virtual bool is_ready();
   virtual void wait_until_ready();
   virtual void issue_operation(DISK_OPERATION _op, unsigned long _block_no);
   /* Send a sequence of commands to the controller to initialize the READ/WRITE
      operation. This operation is called by read() and write(). */

};
```

blocking_disk.C

BlockingDisk::wait_until_ready()

This is the only function that I change in this MP. If the disk is not ready, then we will call the scheduler to add current thread to the end of the queue, then give up CPU to the next thread waiting in the queue. On the next round when this blocked thread is load back to CPU, then it will continue to another loop to check the disk.



```
void BlockingDisk::wait_until_ready() {
  while(!is_ready()) {
    Console::puts("The disk is not ready! Giving up CPU to another thread...\n");
    SYSTEM_SCHEDULER->resume(Thread::CurrentThread());
    SYSTEM_SCHEDULER->yield();
  }
}
```

BlockingDisk::read() / BlockingDisk::write()

The implentation in the function is identical to which in the SimpleDisk. The difference is that the algorithm of wait_until_ready() it calls has changed, therefore when doing read or write, the thread will not keep holding up the CPU.

```
void BlockingDisk::read(unsigned long _block_no, unsigned char * _buf) {
  /* Reads 512 Bytes in the given block of the given disk drive and copies them
   to the given buffer. No error check! */

  issue_operation(DISK_OPERATION::READ, _block_no);

  wait_until_ready();

  /* read data from port */
  int i;
  unsigned short tmpw;
  for (i = 0; i < 256; i++) {
    tmpw = Machine::inportw(0x1F0);
    _buf[i*2]   = (unsigned char)tmpw;
    _buf[i*2+1] = (unsigned char)(tmpw >> 8);
  }
}
```

```
void BlockingDisk::write(unsigned long _block_no, unsigned char * _buf) {
  /* Writes 512 Bytes from the buffer to the given block on the given disk drive. */

  issue_operation(DISK_OPERATION::WRITE, _block_no);

  wait_until_ready();

  /* write data to port */
  int i;
  unsigned short tmpw;
  for (i = 0; i < 256; i++) {
    tmpw = _buf[2*i] | (_buf[2*i+1] << 8);
    Machine::outportw(0x1F0, tmpw);
  }
}
```

makefile

I added the scheduler.o file to the makefile in order to correctly compile the scheduler to or kernel file

```
kernel.o: kernel.C machine.H console.H gdt.H idt.H irq.H exceptions.H interrupts.H simple_timer.H fra
    $(GCC) $(GCC_OPTIONS) -c -o kernel.o kernel.C

kernel.bin: start.o utils.o kernel.o \
   assert.o console.o gdt.o idt.o irq.o exceptions.o \
   interrupts.o simple_timer.o simple_keyboard.o frame_pool.o mem_pool.o \
   thread.o threads_low.o scheduler.o simple_disk.o blocking_disk.o \
    machine.o machine_low.o
    $(LD) -melf_i386 -T linker.ld -o kernel.bin start.o utils.o kernel.o \
   assert.o console.o gdt.o idt.o irq.o exceptions.o interrupts.o \
   simple_timer.o simple_keyboard.o frame_pool.o mem_pool.o \
   thread.o threads_low.o simple_disk.o blocking_disk.o \
   scheduler.o machine.o machine_low.o
```

Testing

I did not add additional test data in this MP6, since I believe the provided test set is sufficient to check if the page table system works as expected. The test result is shown as follows.

Following screenshot shows that the threads can terminate appropriately. The thread first tries to read the disk then found it not available then give up the CPU. Therefore, we can see there is no writing messgage printed.

```
Next thread to run: 1
THREAD: 1
FUN 2 INVOKED!
FUN 2 IN ITERATION[0]
Reading a block from disk...
The disk is not ready! Giving up CPU to another thread...
Resume thread to ready queue: 1
Next thread to run: 2
THREAD: 2
FUN 3 INVOKED!
FUN 3 IN BURST[0]
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]
```