MP6 design

In kernel.C, we create thread 1, 2, 3, 4 and add thread 2, 3, 4 to scheduler. Dispatch thread 1 and scheduler is managing running of threads. Thread 1, 2, 3, 4 will take turns. However unlike mp5, thread2 calls I/O operation using disk. Since it takes long time, move it to the diskQueue in BlockingDisk and pass CPU when until it is ready. Before that, thread 3 runs and then, since disk is ready, thread2 runs.

To implement and test **thread safe disk design(OPTION 3, 4)** I changed thread 3 to call I/O operation. Thus kernel.C is slightly modified.

Main modification code of MP6 is blocking_device.H and blocking_device.C. Also, scheduler.H and scheduler.C are modified to support BlockingDisk operation.

BlockingDisk
blocking_disk.H

```
/*--------------------------------------------------------------------------*/
/* B l o c k i n g D i s k  */
/*--------------------------------------------------------------------------*/
class Queue;
class BlockingDisk : public SimpleDisk {
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();


   void pushToQueue(Thread* thread);
   Queue* getDiskQueue() { return diskQueue; };
private:
   Queue* diskQueue;
};
```

BlockingDisk class is derived from SimpleDisk. It has member variable diskQueue which is used for store the thread called disk I/O operation. And four methods are overrided (read, write, is_ready, wait_until_ready)

Blocking_disk.C

```
extern Scheduler* SYSTEM_SCHEDULER;

/*--------------------------------------------------------------------------*/
/* CONSTRUCTOR */
/*--------------------------------------------------------------------------*/

BlockingDisk::BlockingDisk(DISK_ID _disk_id, unsigned int _size)
  : SimpleDisk(_disk_id, _size) {
    diskQueue = new Queue();
}
```

At first, to use scheduler, declare extern variable SYSTEM_SCHEDULER.
At constructor, allocate queue to diskQueue

```
void BlockingDisk::read(unsigned long _block_no, unsigned char * _buf) {
  SimpleDisk::read(_block_no, _buf);
  SYSTEM_SCHEDULER->setDiskInUse(false);
}


void BlockingDisk::write(unsigned long _block_no, unsigned char * _buf) {
  SimpleDisk::write(_block_no, _buf);
  SYSTEM_SCHEDULER->setDiskInUse(false);
}
```

Read and write method are almost similar with base class SimpleDisk. But after calling it, set boolean variable of scheduler (diskInUse) to indicate operation is done now. This Boolean value will be keypoint of design and implement thread safe disk.

```
bool BlockingDisk::is_ready() {
    return SimpleDisk::is_ready();
}

void BlockingDisk::wait_until_ready() {
    if(!is_ready()) {
        Thread* thread = Thread::CurrentThread();
        if(!SYSTEM_SCHEDULER->getDiskInUse()) {//if disk is not in use, push to disk block queue to use later
            pushToQueue(thread);
            SYSTEM_SCHEDULER->setDiskInUse(true);
        } else {//if disk is not ready and disk already in use, push to normal ready queue to call later
            SYSTEM_SCHEDULER->resume(thread);
        }
        SYSTEM_SCHEDULER->yield();
    }
}
```

Is_ready() method is the same with base class
Wait_until_ready() method is keypoint of BlockingDisk. If the disk is not ready and disk is not in use, it pushed current thread(which calls I/O operation) to diskQueue to separate it from readyQueue of scheduler. And then set disk in use to provide thread safe condition. If other thread access to disk and try to call disk I/O operation (SimpleDisk::read, write calls from BlockingDisk::read, write)both calls wait_until_ready(). If the disk is already in use, it push thread to normal ready queue to be called later.
And after push thread to diskQueue, scheduler->yield() is called. Based on the is_ready condition, yield determine whether dispatch to I/O thread or other thread.

Scheduler

Scheduler.H

```
class BlockingDisk;
class Scheduler {

  /* The scheduler may need private members... */
private:
   Queue* readyQ;
   BlockingDisk* disk;
   bool isDiskInUse;
public:
```

To support blockingDisk operation, scheduler code is also slightly modified.
Member variable disk, isDiskInUse are added and getter, setter for member variables are added. isDiskInUse variable is used for thread-safe design. As mentioned earlier, if it is set as false, that means disk is not currently in use now, so I/O thread can use it. However if it is set as true, that means disk is currently in use, so I/O thread should called later after previous I/O thread handle its operation.

Scheduler.C

```
Scheduler::Scheduler() {
  readyQ = new Queue();
  disk = nullptr;
  isDiskInUse = false;
  Console::puts("Constructed Scheduler.\n");
}
```

In constructor, disk and isDiskInUse is initialized as nullptr and false. Disk of scheduler is set in kernel.C(after allocating BlockingDisk in heap)

```
void Scheduler::yield() {
    if(disk && disk->is_ready() && disk->getDiskQueue()->size() != 0) {
        Thread* diskThread = disk->getDiskQueue()->pop();
        Console::puts("dispatch to disk thread\n");
        isDiskInUse = true;
        if(diskThread) Thread::dispatch_to(diskThread);
    } else {
        Thread* target = readyQ->pop();
        Console::puts("dispatch to ready queue thread\n");
        if(target) Thread::dispatch_to(target);
    }
}
```

As mentioned above, in scheduler yield method, dispatch to whether I/O thread or other thread is determined. If disk is ready and diskQueue has element, dispatch to I/O thread and set diskInUse for true. Else dispatch to next target in ready Q.