CSCE611 Machine Problem 6 Simple Disk Device Driver

Gehao Yu 629008717

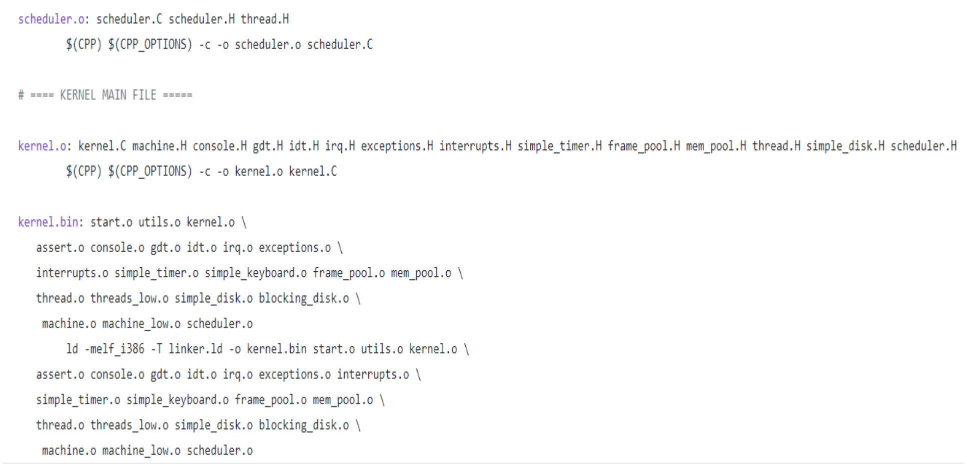
Apr 5 2021

1 Introduction

We have to design the simple disk device driver based on the requirement of the machine problem 6. In this machine problem, We have to handle the busy waiting of the block queue and when we're operating the read and write operations, we don't need to worry about either return prematurely or tie up the entire system waiting for the device to return.

2 Modification based on original makefile

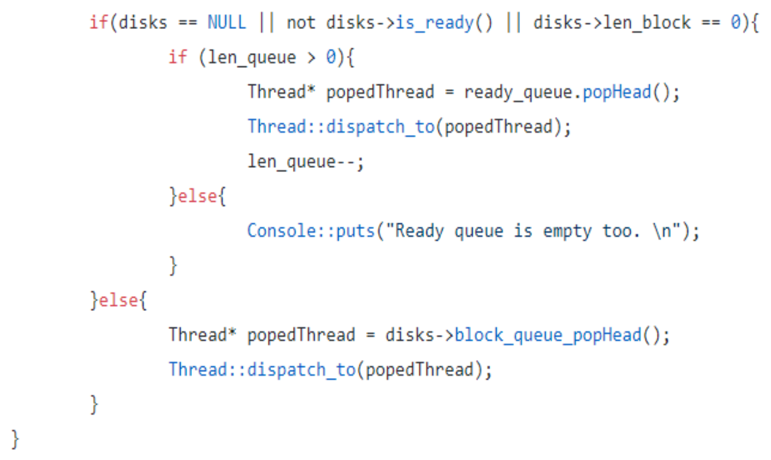
We bring the scheduler.C into the complied files because we need implement the blocking queue based on the ready queue. So we added required sentences to the makefile.



```
scheduler.o: scheduler.C scheduler.H thread.H
	$(CPP) $(CPP_OPTIONS) -c -o scheduler.o scheduler.C

# ==== KERNEL MAIN FILE =====

kernel.o: kernel.C machine.H console.H gdt.H idt.H irq.H exceptions.H interrupts.H simple_timer.H frame_pool.H mem_pool.H thread.H simple_disk.H scheduler.H
	$(CPP) $(CPP_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 simple_disk.o blocking_disk.o \
    machine.o machine_low.o scheduler.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 \
    machine.o machine_low.o scheduler.o
```

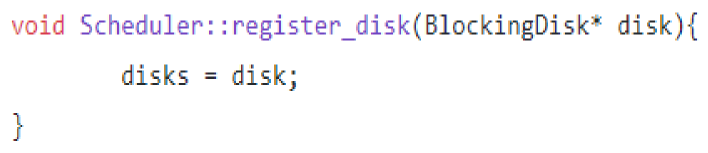
3 Modification based on original scheduler.C

Our old scheduler.C, it couldn't handle the circumstance that we now have disks and another blocking queue. So we have to make the decision that whether situation currently we are and do the corresponding things.



```
        if(disks == NULL || not disks->is_ready() || disks->len_block == 0){
                if (len_queue > 0){
                        Thread* popedThread = ready_queue.popHead();
                        Thread::dispatch_to(popedThread);
                        len_queue--;
                }else{
                        Console::puts("Ready queue is empty too. \n");
                }
        }else{
                Thread* popedThread = disks->block_queue_popHead();
                Thread::dispatch_to(popedThread);
        }
}
```

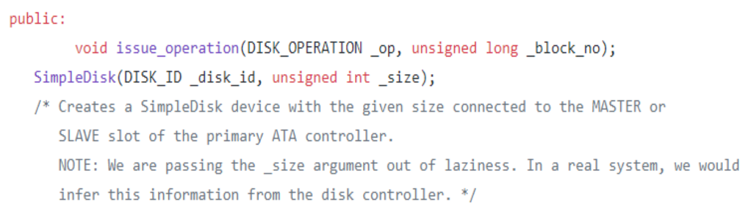
And also, we have to define a new function called register_disk() to register the disk.



```
void Scheduler::register_disk(BlockingDisk* disk){
        disks = disk;
}
```

4 Modification in simple_disk.H

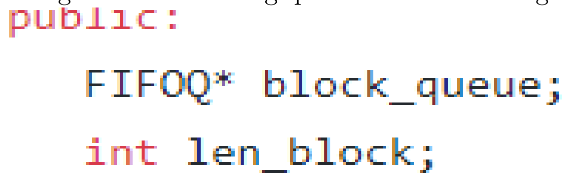
I moved the issue_operation() from private function to public function so that I could call it in my blocking_disk.C.



```
public:
        void issue_operation(DISK_OPERATION _op, unsigned long _block_no);
   SimpleDisk(DISK_ID _disk_id, unsigned int _size);
   /* Creates a SimpleDisk 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. */
```

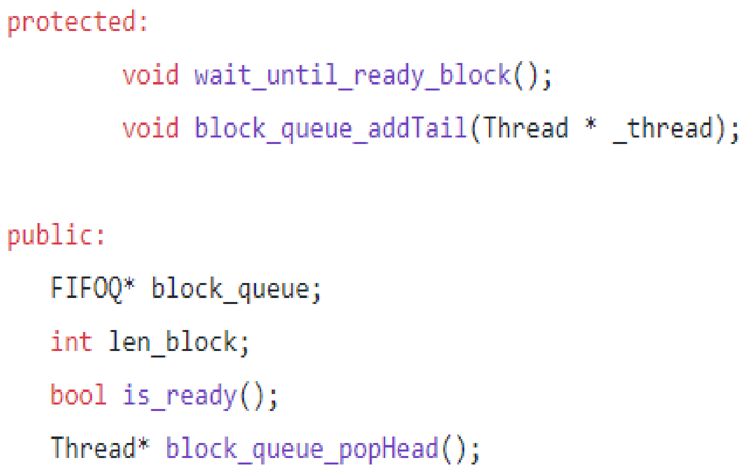
5 Modification in blocking_disk.H

1:New variables in the blocking_disk.H:
We defined two new variables blocking_queue and length of the blocking_queue. The blocking_queue is simply another FIFO queue we implemented before and the length of the blocking queue is to monitoring if it was an empty queue.



```
public:
    FIFOQ* block_queue;
    int len_block;
```

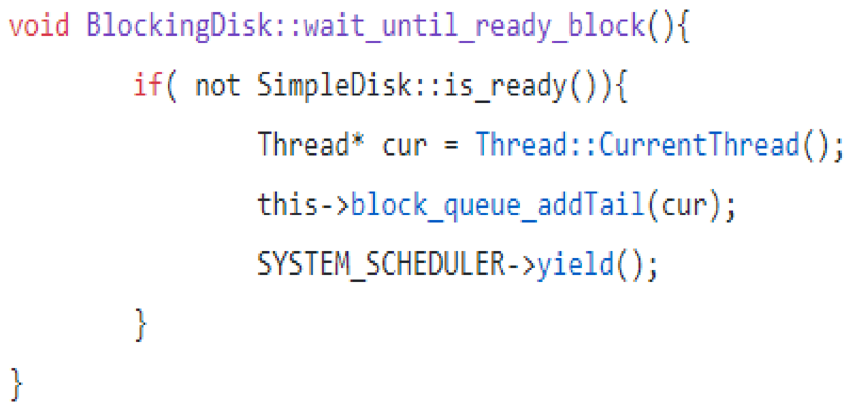
2:New functions in the blocking_disk.H:
wait_until_ready_block(), block_queue_addTail(), block_queue_popHead() and modified read() and write() functions. These functions helped us implemented the operations about what we did in the blocking queue.



```
protected:
        void wait_until_ready_block();
        void block_queue_addTail(Thread * _thread);


public:
    FIFOQ* block_queue;
    int len_block;
    bool is_ready();
    Thread* block_queue_popHead();
```

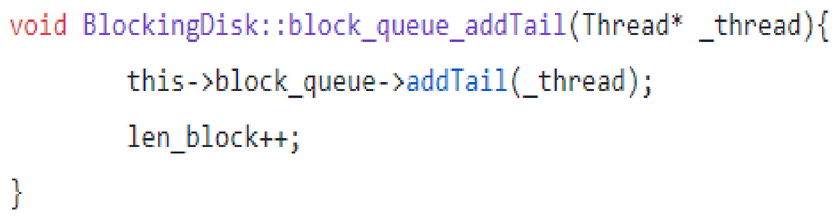
6 Modification in blocking_disk.C

In the blocking_disk.C, we implemented new functions including wait_until_ready_block(), block_queue_addTail(), block_queue_popHead() and filled the required functions read() and write() based on the simple_disk.C.
A: wait_until_ready_block(): Since we already created a block queue, every time we found out that the current disk is not ready, we shall put the current thread into the block queue and yield.

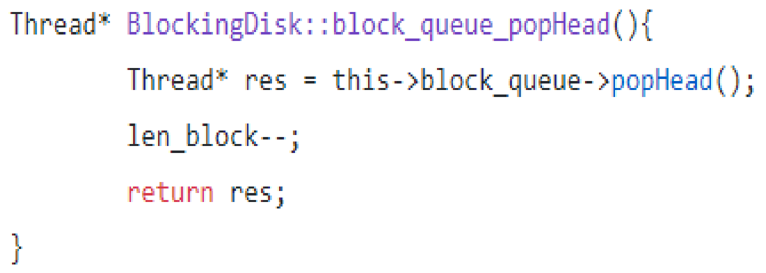


```
void BlockingDisk::wait_until_ready_block(){
        if( not SimpleDisk::is_ready()){
                Thread* cur = Thread::CurrentThread();
                this->block_queue_addTail(cur);
                SYSTEM_SCHEDULER->yield();
        }
}
```

B: block_queue_addTail() & block_queue_popHead(): Simply adding the thread in the end of block queue or popping the head thread of the block queue.



```
void BlockingDisk::block_queue_addTail(Thread* _thread){
        this->block_queue->addTail(_thread);
        len_block++;
}
```



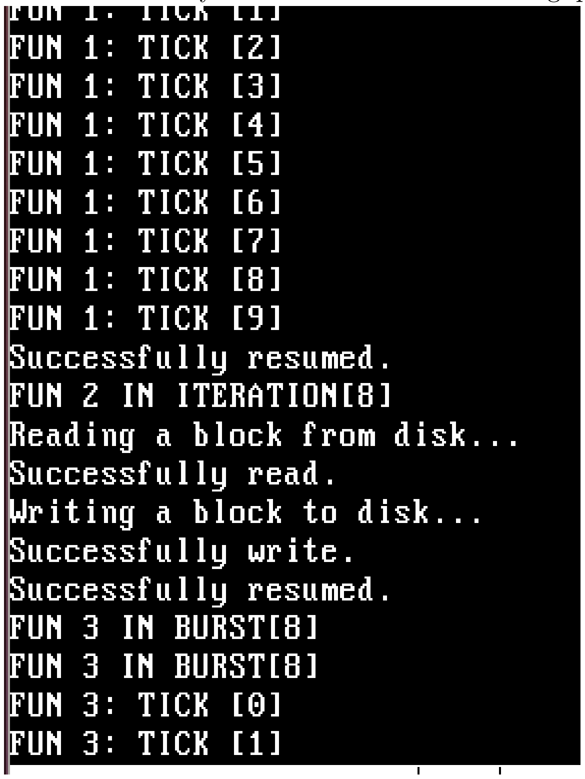
```
Thread* BlockingDisk::block_queue_popHead(){
        Thread* res = this->block_queue->popHead();
        len_block--;
        return res;
}
```

C: read() & write(): The function is similar to the read() and write() in the simple_disk.C, but in here we call our own wait_until_ready_block() function instead of the wait_until_ready() function.

```
void BlockingDisk::read(unsigned long _block_no, unsigned char * _buf) {

  //SimpleDisk::read(_block_no, _buf);
      issue_operation(READ, _block_no);

  wait_until_ready_block();

  /* 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);
  }
  Console::puts("Successfully read. \n");

}


void BlockingDisk::write(unsigned long _block_no, unsigned char * _buf) {
      issue_operation(WRITE, _block_no);

  wait_until_ready_block();

  /* 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);
  }
  //SimpleDisk::write(_block_no, _buf);
  Console::puts("Successfully write. \n");
}
```

7 Result of new blocking disk

We can see that, after we finished all the parts, once we met a blocking problem, we could correctly add threads into the blocking queue with read() and write():



```
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 [7]
FUN 1: TICK [8]
FUN 1: TICK [9]
Successfully resumed.
FUN 2 IN ITERATION[8]
Reading a block from disk...
Successfully read.
Writing a block to disk...
Successfully write.
Successfully resumed.
FUN 3 IN BURST[8]
FUN 3 IN BURST[8]
FUN 3: TICK [0]
FUN 3: TICK [1]
```