MP6: Primitive Disk Device Driver

Vaibhav Pundir
UIN: 734004197
CSCE611: Operating System

Assigned Tasks

Main: Completed.
Bonus Option 1: Did not attempt.
Bonus Option 2: Did not attempt.
Bonus Option 3: Did not attempt.
Bonus Option 4: Did not attempt.

System Design

In this machine problem, the aim is to implement a blocking read and write layer to improve the simple disk driver without using the busy waiting logic.

The implementation is further explained below:

Code Description

For this machine problem, I've made changes to the following code files:

1. blocking_disk.H
2. blocking_disk.C
3. queue.H
4. scheduler.H
5. scheduler.C
6. simple_disk.C
7. kernel.C
8. makefile

blocking_disk.H

BlockingDisk inherits from SimpleDisk and declares the blocking_q data structure and additional methods to manage blocking disk tasks.

```
class BlockingDisk : public SimpleDisk {

   Queue blocking_q;

public:
   ...

   Thread * top();
   /* This method returns the top thread from the blocking_q.
      peek() the queue & then dequeue. */
```

```
11
12    void wait_until_ready();
13    /* Add the thread to the blocking_q and give up CPU to the next thread. */
14
15    bool has_blocking_thread();
16    /* Returns true if the disk is ready for data transfer and blocking_q
17       is not empty, false otherwise.*/
18
19    ...
20
21 };
```

Listing 1: C++ code snippet

blocking_disk.C

BlockingDisk::top() : Retrieve the thread from the front of the blocking queue.

```
Thread * BlockingDisk::top() {
  Thread *t = blocking_q.peek();  // get thread from blocking queue
  blocking_q.dequeue();           // remove the thread from queue
  return t;
}
```

Listing 2: C++ code snippet

BlockingDisk::wait_until_ready() : Adds the thread to the blocking queue and gives up CPU control to the next thread.

```
void BlockingDisk::wait_until_ready() {
  blocking_q.enqueue(Thread::CurrentThread());  // insert thread to the queue
  SYSTEM_SCHEDULER->yield();                    // give up CPU
}
```

Listing 3: C++ code snippet

BlockingDisk::has_blocking_thread() : Returns true if the disk is ready for data transfer and the blocking queue is not empty, otherwise false.

```
bool BlockingDisk::has_blocking_thread() {
  // if disk is ready and blocking queue is not empty
  return (SimpleDisk::is_ready() && !blocking_q.isEmpty());
}
```

Listing 4: C++ code snippet

queue.H

The below code defines a struct queue that the scheduler and blocking disk use for the ready and blocking queue. The queue implementation is the same as MP5 and the code is moved to the queue.H for efficient management.

```
// Define the struct for a queue node
struct Node {
   Thread *th;
   Node* next;

   Node(Thread *t) : th(t), next(nullptr) {}
};

// Define the struct for the queue itself
struct Queue {
   Node* front;
   Node* rear;

   Queue() : front(nullptr), rear(nullptr) {}

```

```
    // Function to check if the queue is empty
    bool isEmpty() {
        return front == nullptr;
    }

    // Function to enqueue (insert) an element into the queue
    void enqueue(Thread *t) {
        Node* newNode = new Node(t);
        if (isEmpty()) {
            front = rear = newNode;
        } else {
            rear->next = newNode;
            rear = newNode;
        }
    }

    // Function to dequeue (remove) an element from the queue
    void dequeue() {
        if (isEmpty()) {
            return;
        }
        Node* temp = front;
        front = front->next;
        if (front == nullptr) {
            rear = nullptr;
        }
        delete temp;
    }

    // Function to delete a specific element from the queue
    void remove(Thread *t) {
        Node *curr = front;
        Node *prev = nullptr;

        while(curr != nullptr) {
            if(t->ThreadId() != curr->th->ThreadId()) {
                if(curr == front) {
                    dequeue();
                    return;
                }
                prev->next = curr->next;
                if(curr == rear)
                    rear = prev;

                delete curr;
                return;
            }
            prev = curr;
            curr = curr->next;
        }
    }

    // Function to get the front element of the queue without removing it
    Thread * peek() {
        if (isEmpty()) {
            // std::cout << "Queue is empty. No element to peek.\n";
            return nullptr; // Return some default value indicating failure
        }
        return front->th;
    }
};
```

Listing 5: C++ code snippet

scheduler.H

The struct for the queue was moved to the header file queue.H.

```
#include "queue.H"
```

Listing 6: C++ code snippet

scheduler.C

Scheduler::yield(): This method is modified to first check if the disk is ready for data transfer and has a thread waiting for CPU in the blocking_q, then give CPU to this blocked thread, otherwise give CPU to the thread in the ready_q.

```
void Scheduler::yield() {
  // assert(false);
  // disabling the interrupts before performing operation on ready queue
  if(Machine::interrupts_enabled())
    Machine::disable_interrupts();

  // if disk is ready and has threads waiting in blocking queue
  // give CPU to the blocked thread
  if(SYSTEM_DISK->has_blocking_thread()) {
    // enabling the interrupts
    if(!Machine::interrupts_enabled())
      Machine::enable_interrupts();

    // dispatch the thread from blocking queue to the CPU
    Thread::dispatch_to(SYSTEM_DISK->top());

  } else if(!ready_q.isEmpty()) {  // if ready queue is not empty
    Thread *t = ready_q.peek(); // fetching thread from the front of queue
    ready_q.dequeue();          // remove the thread

    // enabling the interrupts
    if(!Machine::interrupts_enabled())
      Machine::enable_interrupts();

    // dispatch the thread to the CPU
    Thread::dispatch_to(t);
  }
}
```

Listing 7: C++ code snippet

simple_disk.C

SimpleDisk::is_ready(): Updated the method to mimic disk i/o delay.

```
static int delay = 1;
bool SimpleDisk::is_ready() {
  if(delay == 0) {
    ++delay;
    return false;
  }
  --delay;
  return ((Machine::inportb(0x1F7) & 0x08) != 0);
}
```

Listing 8: C++ code snippet

kernel.C

- Uncommented macro _USES_SCHEDULER_ to use the scheduler.
- Update SYSTEM_DISK object type to BlockingDisk.
- Increase stack size to 4096 to fix the stack overflow issue.

```
...
#define _USES_SCHEDULER_
...
BlockingDisk * SYSTEM_DISK;
...
SYSTEM_DISK = new BlockingDisk(DISK_ID::MASTER, SYSTEM_DISK_SIZE);
...
Console::puts("CREATING THREAD 2...");
char * stack2 = new char[1024 << 2];
thread2 = new Thread(fun2, stack2, 1024 << 2);
Console::puts("DONE\n");
...
```

Listing 9: C++ code snippet

makefile

Adding scheduler and queue files

```
scheduler.o: scheduler.C scheduler.H thread.H
  $(GCC) $(GCC_OPTIONS) -c -o scheduler.o scheduler.C

queue.o: queue.H thread.H
  $(GCC) $(GCC_OPTIONS) -c -o queue.o

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
  $(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 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
```

Listing 10: C++ code snippet

Testing

To test the implementation, I used the test provided in kernel.C.

TEST

Uses FIFO Scheduler with non-terminating threads along with Thread 2 issuing disk i/o operations. Each thread except for Thread 2 executes for 10 ticks and hands over the CPU to the next thread.
- On starting disk i/o, Thread 2 gives up CPU to the next thread in the scheduler's queue instead of busy waiting.
- When the disk read operation starts, the CPU is given to another thread.
- Once the control comes back to Thread 2, the disk write operation is executed.

```
<bochs:1> c
Installed exception handler at ISR <0>
Allocating Memory Pool... done
Installed interrupt handler at IRQ <0>
Constructed Scheduler.
Hello World!
CREATING THREAD 1...
esp = <2098128>
done
DONE
CREATING THREAD 2...esp = <2102248>
done
DONE
CREATING THREAD 3...esp = <2103296>
done
DONE
CREATING THREAD 4...esp = <2104344>
done
DONE
STARTING THREAD 1 ...
THREAD: 0
FUN 1 INVOKED!
FUN 1 IN ITERATION[0]
FUN 1: TICK [0]
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]
THREAD: 1
FUN 2 INVOKED!
FUN 2 IN ITERATION[0]
Reading a block from disk...
THREAD: 2
INTERRUPT NO: 14
NO DEFAULT INTERRUPT HANDLER REGISTERED
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]
FUN 3: TICK [8]
FUN 3: TICK [9]
```

```
FUN 3 IN BURST[3]
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]
FUN 3: TICK [8]
FUN 3: TICK [9]
FUN 2 IN ITERATION[2]
Reading a block from disk...
INTERRUPT NO: 14
NO DEFAULT INTERRUPT HANDLER REGISTERED
FUN 4 IN BURST[3]
FUN 4: TICK [0]
FUN 4: TICK [1]
FUN 4: TICK [2]
FUN 4: TICK [3]
FUN 4: TICK [4]
FUN 4: TICK [5]
FUN 4: TICK [6]
FUN 4: TICK [7]
FUN 4: TICK [8]
FUN 4: TICK [9]
Writing a block to disk...
FUN 1 IN ITERATION[4]
FUN 1: TICK [0]
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]
INTERRUPT NO: 14
NO DEFAULT INTERRUPT HANDLER REGISTERED
FUN 3 IN BURST[4]
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]
FUN 3: TICK [8]
FUN 3: TICK [9]
FUN 4 IN BURST[4]
```

Figure 1: Test