Document

Zhiren Zhou

November 2022

1 Options Implemented

Beyond the blocking_disk driver, I implemented the **OPTION1 Support for Disk Mirroring** and **OPTION3 Design of a thread-safe disk system.**.

2 Design Concept

1. Modified the **SimpleDisk** class:

   Moved is_ready() to public so that it can be used in the following MirroredDisk class.

2. Implemented the **BlockingDisk** class:

   BlockingDisk is the subclass of class **SimpleDisk**. Most function in class **Scheduler** is virtual function which can be redefined by subclass such as wait_until_ready.

3. Implemented the **BlockingDisk::BlockingDisk()** function:

   This is the constructor of BlockingDisk. It prints one piece of construction information to the console and specifies the disk_no and disk size of the controlled disk.

4. Implemented the **BlockingDisk::wait_until_ready()** function:

   This function does not use busy_waiting any more. Instead, it asks the disk controller for just one time. If the response is not_ready, the thread will voluntarily yield the CPU and wait for its next turn. It is quite possible that when it resumes, the disk is ready and it can keep reading/write operation.

5. Implemented the **BlockingDisk::read()** function:

   This function calls wait_until_ready() function first. If the disk is not

ready, this thread yields the CPU and waits for next turn. Otherwise, it reads 512bytes data from disk to a buffer.

6. Implemented the **BlockingDisk::write()** function:

   This function calls wait_until_ready() function first. If the disk is not ready, this thread yields the CPU and waits for next turn. Otherwise, it writes 512bytes data from buffer to the disk.

7. **OPTION1**:Implemented the **MirroredDisk** class:

   This class is a driver responding to a mirror disk system. It controls two disks. When the user uses read() function through this driver, it will try to read data from the fast-prepared disk. When the user uses write() function through this driver, it will write data to both two disks.

8. **OPTION1** Implemented the **MirroredDisk::MirroredDisk()** function:

   This is the constructor of MirroreDisk class. Since this driver has to manage two disks. It has two SimpleDisk object, one for master disk and one for dependent disk.

9. **OPTION1** Implemented the **MirroredDisk::issue_mirrored_operation()** function:

   This function adds the disk_no as the param. So it can query the read/write operation to the controller, which controls the master disk and dependent disk by passing through disk_no=MASTER or disk_no=DEPENDENT.

10. **OPTION1** Implemented the **MirroredDisk::read()** class:

    This function first calls twice issue_mirrored_operation() function to tell the controller that it wants to read information from master disk or dependent disk. If both disks are not ready, this thread voluntarily yields the CPU. Otherwise, it reads data from the port and it does not care about which disk provides the data.

11. **OPTION1** Implemented the **MirroredDisk::write()** function:

    This function first calls issue_mirrored_operation() function to tell the controller that it wants to read information to master disk. If master disk is not ready, this thread voluntarily yields the CPU. Otherwise, it writes data to master disk through port. Then it calls issue_mirrored_operation() function for the second time to tell the controller that it wants to read information to dependent disk. If the dependent disk is not ready, this thread voluntarily yields the CPU. Otherwise, it writes data to the dependent disk through port.

12. **OPTION3** Design **a thread-safe disk system.**:

    In order to design a safe disk system. We need to (1)Create a request queue to store requests from multiple threads. Add provision to store the thread info too. (2)Enable interrupts (3)Implement a handler that will process the actual io transfer once device is ready. (4)Keep a mutex like mechanism (a pair of enable_interrupts & disable_interrupts) handy. (5)Add every new request to this queue. (6)Issue the commands & yield the thread. We need not wait for the device ready status; it will be interrupted. (7)Interrupt 14 got triggered & handler caught the interrupt. Process the io transfer (8)Pop the head io request that just got handled & push the owner thread to the ready queue again. (9)Apply the mutex in both push & pop operations to synchronize the threads. This way multiple threads can operate concurrently without having to bother about the requests being lost.

3 Files Modified

1. **scheduler.H** (Implemented in MP5)
2. **scheduler.C** (Implemented in MP5)
3. **simple_disk.H** (for **OPTION1**)
4. **blocking_disk.H**
5. **blocking_disk.C**
6. **mirrored_disk.H** (OPTION1)
7. **mirrored_disk.C** (OPTION1)
8. **kernel.C** (for Test:Comment the macro _USES_SCHEDULER_ in kernel.C to select the FIFOcheduler. Comment the macro _DISK_MIRRORING in kernel.C to select mirrored disk.)
9. **makefile** (for Compiling)

4 Testing Results

1. Testing BlockingDisk with disk is not ready

```
69 ----FIFOScheduler::resume()----
70 Resume a thread:1
71 ----FIFOScheduler::resume() Successfully----
72 ----FIFOScheduler::yield()----
73 Disable Interrupts....
74 Dispatching Thread to:2
75 Enable Interrupts....
76 THREAD: 1
77 FUN 2 INVOKED!
78 FUN 2 IN ITERATION[0]
79 Reading a block from disk...
80 ----FIFOScheduler::resume()----
81 Resume a thread:2
82 ----FIFOScheduler::resume() Successfully----
83 Device is not ready, voluntarily yielding thread
84  ----FIFOScheduler::yield()----
85 Disable Interrupts....
86 Dispatching Thread to:3
87 Enable Interrupts....
88 THREAD: 2
89 FUN 3 INVOKED!
90 FUN 3 IN BURST[0]
91 FUN 3: TICK [0]
92 FUN 3: TICK [1]
93 FUN 3: TICK [2]
94 FUN 3: TICK [3]
95 FUN 3: TICK [4]
96 FUN 3: TICK [5]
97 FUN 3: TICK [6]
98 FUN 3: TICK [7]
99 FUN 3: TICK [8]
100 FUN 3: TICK [9]
101 ----FIFOScheduler::resume()----
102 Resume a thread:3
```

2. Testing BlockingDisk with disk is ready

```
128 FUN 1 IN ITERATION[1]
129 FUN 1: TICK [0]
130 FUN 1: TICK [1]
131 FUN 1: TICK [2]
132 FUN 1: TICK [3]
133 FUN 1: TICK [4]
134 FUN 1: TICK [5]
135 FUN 1: TICK [6]
136 FUN 1: TICK [7]
137 FUN 1: TICK [8]
138 FUN 1: TICK [9]
139 ----FIFOScheduler::resume()----
140 Resume a thread:1
141 ----FIFOScheduler::resume() Successfully----
142 ----FIFOScheduler::yield()----
143 Disable Interrupts....
144 Dispatching Thread to:2
145 Enable Interrupts....
146 ----FIFOScheduler::yield() Successfully----
147 Device is ready, performing the read operation
148  Writing a block to disk...
149 Device is ready, performing the write operation
150  ----FIFOScheduler::resume()----
151 Resume a thread:2
152 ----FIFOScheduler::resume() Successfully----
153 ----FIFOScheduler::yield()----
154 Disable Interrupts....
155 Dispatching Thread to:3
156 Enable Interrupts....
157 ----FIFOScheduler::yield() Successfully----
158 FUN 3 IN BURST[1]
159 FUN 3: TICK [0]
160 FUN 3: TICK [1]
161 FUN 3: TICK [2]
162 FUN 3: TICK [3]
163 FUN 3: TICK [4]
164 FUN 3: TICK [5]
165 FUN 3: TICK [6]
166 FUN 3: TICK [7]
167 FUN 3: TICK [8]
168 FUN 3: TICK [9]
169 ----FIFOScheduler::resume()----
170 Resume a thread:3
171 ----FIFOScheduler::resume() Successfully----
```

3. Testing MirroedDisk(**OPTION!**)

```
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]
----FIFOScheduler::resume()----
Resume a thread:1
----FIFOScheduler::resume() Successfully----
----FIFOScheduler::yield()----
Disable Interrupts....
Dispatching Thread to:2
Enable Interrupts....
----FIFOScheduler::yield() Successfully----
FUN 2 IN ITERATION[27]
Reading a block from disk...
Issuing mirrored operation
 Writing a block to disk...
Issuing mirrored operation
 Master Disk is ready, write to it
 Dependent Disk is ready, write to it
 ----FIFOScheduler::resume()----
Resume a thread:2
----FIFOScheduler::resume() Successfully----
----FIFOScheduler::yield()----
Disable Interrupts....
Dispatching Thread to:3
Enable Interrupts....
----FIFOScheduler::yield() Successfully----
FUN 3 IN BURST[28]
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]
----FIFOScheduler::resume()----
Resume a thread:3
----FIFOScheduler::resume() Successfully----
```