CSCE 611 MP6 Hung-Hsueh Shih UIN:132006330
Primitive Disk Device Driver
Introduction:
Implemented a layer of device driver on top of a simple disk which without busy waiting and enable call read and write operation.
BLOCKING DISK IMPLEMENTATION
Modified files :
blocking_disk.H
blocking_disk.C
Scheduler.C
kernel.C
Makefile

In blocking disk header file I implement BlockingDisk class by inheriting from a simple disk class. The BlockingDisk class supports read-write operation without busy waiting.
In blocking_disk.C file I added several functions:

1. disk_is _ready: to check whether the blocking disk is ready to do I/O operation. This function inherits from a simple disk class.
2. Disk_dequeue: if the disk queue is not empty and the disk is ready, then we call this function in yield function, which will remove the first item in the disk queue and return the thread in disk queue.
3. disk_enqueue: add the thread in the disk queue
4. disk_wait_until_ready: first, I check whether the disk is ready, if it is not ready, I call enqueue function add thread into the disk queue and call the yield function to let thread give up CPU.
5. Read: in the read function, I issue the I/O operation which is inherited from a simple disk, and call disk_wait_until_ready function to check whether the disk is ready.
6. Write: basically it's the same as the read function instead, I used the read function inherited from simple disk.

In scheduler.C in order to improve the waiting time, instead of using the resume function in the wait until ready, which checks the status of the disk when the scheduler executes the same thread which issued the I/O request, I check the status of the disk queue and disk at yield function. That is, every time a thread decides to give up CPU and calls a yield function, I'll check whether the disk is ready, if it's ready the I dequeue the disk queue and dispatch that thread. Otherwise, I remove the first item from scheduler ready queue and dispatch that thread.

BOBUS OPTION1 Support for disk mirroring
Modified file :
Mirrored_disk.H
Mirrored_disk.C

Kernal.C
Scheduler.C
In Mirrored_disk.H I implemented mirrored disk through using to blocking disk. One is MASTER_DISK one is DEPENDENT_DISK.
In Mirrored_disk.C I add several functions:
issue_operation: in order to let the machine to know which disk issue signal, I add a disk_id to separate the master disk and dependent disk.
Wait until ready, in this function I need to check the status of the master disk and dependent disk at the same time. For ready operation, as long as one thread is ready then I return to the caller. For the write operation, both disks need to be ready then I can write data into both disks.
Read: both master and dependent disk issue operation and see which one is faster, then return do the read operation.
Write: I call blocking disk to let both disks do write operation.
In scheduler.C I still check the status of both disks in the yield function.

OPTION 2: Using Interrupts for Concurrency.
I have implemented interrupts when the interrupt is invoked at the time the disk is ready to read or write.

1. I register the interrupt handler function at 14.
2. If the disk is not ready then I add the thread into the disk queue to wait
3. Once the disk is ready the interrupt occurs ad I dequeue the thread from the disk queue and dispatch that thread to the CPU.
4.

OPTION 3: Design of a thread-safe disk system.
In order to support a multithread system and keep the concurrency. The easiest way is using a lock to control access the disk. That is to say, no matter which thread is going to read or write disk. It needs to check whether the disk is not accessed by another thread if it is accessed by another thread, which means it needs to wait. Therefore, we can simply add the thread into the disk waiting queue. Then, as soon as the previous job is done, the previous thread will release the lock and at the next interval time, the thread in the disk waiting queue will access the lock and enable it to access the disk.

OPTION 4: Implementation of a thread-safe disk system.
In order to keep the lock, I add a variable in the scheduler which is a flag. Every time the thread calls the read/to write function then it will check the status of the disk ad the lock, if the lock is accessed by another thread the thread will resume waiting for the queue, otherwise, it will access the lock and execute the operation.
In scheduler, I had a flag variable as a lock.
In scheduler, I check the lock inside the yield function to decide a thread can access the disk or not. If the disk is ready and the thread possesses the lock it can dispatch to CPU otherwise I resume it.