CSCE 611

Gautham Srinivasan (UIN: 927008557)

MP6 Design Document

Objective:

The objective of machine problem 6 is to add a layer on top of simple device driver to support the same blocking read and write operations as the basic implementation, but without busy waiting in the device driver code and bonus points for **options 1,2,3,4** have been done.

Following are the files modified:

Scheduler.C

Scheduler.H

Kernel.C

Blocking_disk.C

Blocking_disk.H

Mirroring_disk.C

Mirroring_disk.H

makefile

Blocking Disk Design Implementation:

In this MP, we need to implement blocking disk driver which inherits simple disk and supports reads and write without busy waiting. This MP makes use of scheduler where the thread yields the CPU if the disk is not ready.

I have added new wait_until_ready() which checks where the driver is ready or not. If the driver is not ready, it yields the CPU so that other threads can be executed in the meantime. The scheduler put this thread in its waiting queue and executes other threads. After some time, it once again resumes this thread and this ready again checks if the driver is ready or not and this process continues until the driver is ready.

The read, write and is_ready methods are used from the simple disk as the blocking disk inherits the methods of simple disk.

Mirroring Disk [Option 1]:

I have implemented mirroring disk by inheriting blocking disk where the data is written to both master and slave disk and it is ready whichever is ready first.

Design Implementation:

1. Mirroring disk inherits blocking disk
2. It creates two instances of blocking disk for master and slave in the constructor.
3. When write is requested by the user, mirroring disk writes to both master and slave.
4. When read is requested by the user, mirroring disk issues command to master and slave disk and it wait for either of the disk to complete and gives the data to the user.

Interrupts [Option 2]:

I have implemented interrupts where the interrupt is invoked once the disk is ready and the data can be read/written to the disk.

Design Implementation:

1. Register the interrupt handler function
2. If the disk is not ready, push the thread id into a local ready_queue and yield the CPU
3. Once the interrupt handler function is called, pop the thread from the local ready queue and resume the thread.

Design of thread safe system [Option 3]:

As multiple threads can access the disk, we can protect the disk when the user calls read and write API. As we cannot use POSIX mutex or semaphore, I have implemented test and set function which protect the disk when read and write API is called by the user. This function can be made atomic by masking the interrupts.

Implementation of thread safe system [Option 4]:

I have implemented the above stated design for mirroring_disk.

Note:

To test the code, do the following:

1. To test blocking disk, uncomment ENABLE_BLOCKING_DISK macro if commented in kernel.C
2. To test mirroing disk (**option 1**), comment ENABLE_BLOCKING_DISK macro if uncommented kernel.C
3. To test interrupts feature (**option 2)**, uncomment INTERRUPTS_ENABLED macro if commented blocking_disk.H
4. To test thread sync feature, uncomment (**option 3 & 4)** ENABLE_THREAD_SYNC macro if commented blocking_disk.H