Design notes

Mian Qin UIN:725006574

1 Overview

In this machine problem, I implemented a blocking disk IO driver. Also finished the bonus **task 3&4**, i.e. make it thread safe. The main idea of blocking disk IO is maintaining a disk request queue in the BlockingDisk class. Every time a read/write request comes to disk, it first enqueue and then yield the CPU. In the scheduler, every time it calls resume, it will traverse all the disks in the system and see if the disk is ready, if ready, it will add the thread in the head of the request queue and adds it to the scheduler ready queue to make it run.
In this way, it complete both serializing the requests (make it thread-safe) and also not blocked by read/write operation. Since in the scheduler:resume() it will check whether disk is ready and then scheduler the thread into ready queue.

2 Implementation

2.1 Scheduler

The Scheduler class is basically the same with MP5, only modify resume()

2.1.1 Scheduler::Scheduler ()

In Scheduler constructor, it will initialize the head/ tail of the two linked-list.

2.1.2 Scheduler::yield()

For the yield() function, it need to do two jobs. First it will scan the clean up queue that stores the terminated threads and as long as is not in the terminated threads' context, it will perform the clean up (delete the Thread class instance, release the thread stack memory).
Then, it will do the scheduling job, by grab the tail node of the FIFO queue and switch to the thread in the tail of the FIFO queue. Also, it need to delete the thread node in the FIFO queue to prevent memory leakage.

2.1.3 Scheduler::add(Thread * _thread)

For the add function, it will first check whether the Thread class has registered the scheduler, if not, it will pass the scheduler pointer to the static member of the Thread class for termination use.
Then it will put the _thread information to the front of the FIFO queue, basically perform a insert to head of the double linked-list.

2.1.4 Scheduler::resume(Thread * _thread)

For resume() it first check each disk and if disk is ready, add the thread in the head of the request queue and add it to ready queue. Then add thread itself to ready queue..

2.1.5 Scheduler::terminate(Thread * _thread)

In the terminate() function, since it's called by the thread_shutdown() function in the terminated thread's context, it cannot directly release the thread memory resources. So, it will put the thread information into the clean up list. And the call the yield() to do a thread switch to release the terminated thread in other threads' context. (Done in the yield() call)

2.3 BlockingDisk

2.3.1 Scheduler::read()

First enqueue request to the request queue in Blockingdisk. Then yield CPU for scheduler. We resume issue the IO request to disk and then yield CPU again. Finally, we disk is ready for the data and thread will resume again and copy buffer.

2.3.2 Scheduler::write()

Basically the same as read.