MP6: Primitive Disk Device Driver  
Kai-Chih Huang  
UIN: 333000021   
CSCE611: Operating System

Assigned Tasks

Main Part: Finished  
Bonus Option 1: Finished  
Bonus Option 2: Finished

System Design

The goal of this machine problem is to implement a simple file system. I this file system, I used 1 disk block to store the inodes, and 1 disk block to keep track of the free blocks. For the bonus option 1, I designed a 2-level block id structure to store files that requires multiple blocks. To be more specific, I change the block\_id attribute in the Inode data structure to store the ids of the blocks used by the file, instead of storing a single block.

Code Description

I implemented file\_system.H/C and file.H/C. In addition, I changed some codes in kernel.C for testing.   
  
To compile code, simply run following command lines under MP6\_Sources directory:  
  
$ make clean // clean the old compile files before we compile

$ make // compile files

$ ./copykernel.sh (if permission denied, try chmod u+x ./copykernel.sh then do it again) //copy kernel

$ bochs -f bochsrc.bxrc // run bochs  
  
I will walk **through the** functions/methods defined in above files as follows.  
  
**file\_system.H**

I defined Inode class and file\_system class in this file. Inode class contains 4 attributes, which is id(file name), block\_id(list of block ids that store the file for bonus option task), size(file size in bytes) and a pointer to the FileSystem class.  
A screenshot of a computer program

Description automatically generated

As for the FileSystem class, it has the disk size attribute, the inode list, a freeblock list, and a pointer to the SimpleDisk class. I added a GetFreeBlock function to simply find and return the id of a free block.  
A screenshot of a computer program

Description automatically generatedA computer screen shot of text

Description automatically generated

**file\_system.C**

FileSystem::FileSystem()  
This is the constructor that simply initialize the attributes, nothing special  
A screen shot of a computer code

Description automatically generated

FileSystem::~FileSystem()  
The destructor will first write back the inode list and free block list to disk block 0 and disk block 1 correspondingly, then free the memory.  
A computer screen shot of text

Description automatically generated

bool FileSystem::Mount(SimpleDisk \* \_disk)  
This function connect the file system to the disk. It will read the first disk block to inode list, and read the second block as the free block list.  
A computer screen shot of text

Description automatically generated  
  
bool FileSystem::Format(SimpleDisk \* \_disk, unsigned int \_size)  
It set the inodes list with size MAX\_INODES to 0 and write to block 0, then m ark the 1st and 2nd position as used and write to the disk at block 1.  
A computer screen shot of text

Description automatically generated

bool FileSystem::CreateFile(int \_file\_id)  
Before creating a new file, we first check if the file name is already existed, and if there is free block. Then, we initialize a new inode for the file.  
A computer screen shot of text

Description automatically generated

bool FileSystem::DeleteFile(int \_file\_id)  
Same as creating a file, we also need to check if the file exists before we delete the file. Then we free all the blocks in the block\_id list and the block of block list itself. Finally we invalidate inode and returnA computer screen shot of a program code

Description automatically generated

int FileSystem::GetFreeBlock()  
The function iterates through the free\_block list and find a free block, wipe it clean then return the id of that block. If there is no free block, it will return -1.  
A computer screen shot of text

Description automatically generated

**file.H**I added some attributes to the File class. The curr\_pos is the current position of the file indicates which position in the file that the will be read or written next. In addition, I added the block\_ids to store the content of inode.block\_id, which a list of blocks used by the file. Finally, I set the MAX\_FILE\_SIZE to indicate the maximum file size allowed, which is 64KB.  
A screenshot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generated

**file.C**

File::File(FileSystem \*\_fs, int \_id)  
The constructor simply initiate the attributes in the class.  
A computer code on a dark background

Description automatically generated  
  
File::~File()  
The destructor simply frees the memory allocated when creating the file instance. Note that we don’t write back any cached data since we already write back everything needed when reading or writing.**A screen shot of a computer code

Description automatically generated**

int File::Read(unsigned int \_n, char \*\_buf)  
It will iterate given times, each time read 1 byte to the buffer. Every time when curr\_pos % SimpleDisk::BLOCK\_SIZE == 0, that means it is at the beginning of a disk block, so we load that block to the block\_buffer, then read the content in that buffer.  
A computer screen shot of text

Description automatically generated

int File::Write(unsigned int \_n, const char \*\_buf)  
It will iterate given times, each time write 1 byte to the buffer. Every time when curr\_pos % SimpleDisk::BLOCK\_SIZE == 0, that means the buffer is full. Then we write back the buffer, find a new free block and wipe out the buffer and start writing again.A computer screen shot of text

Description automatically generatedA computer screen shot of text

Description automatically generated

bool File::EoF()  
This function simply returns whether the current position of the file reaches the end of the file.  
A computer screen shot of a computer code

Description automatically generated

**kernel.C**

void exercise\_file\_system(FileSystem \* \_file\_system)  
I changed the STRING1, STRING2 variable to be larger than 512B to test if the file system successfully store files that requires multiple disk blocks.

**A computer code on a dark background

Description automatically generated**

Testing

As mentioned above, I changed the STRING1, STRING2 variable to be larger than 512B to test if the file system successfully stores files that requires multiple disk blocks. Following screenshot shows that the file system works successfully for several iterations.A screenshot of a computer program

Description automatically generated