Project 4: cpmFS - A Simple File System

Satyam P. Todkar

szt0064@auburn.edu

Department of Computer Science & Software Engineering, Auburn University

Introduction:

In this project, we develop a system called cpmFS. The file system is a simulated one and does not needs rebuilding of the kernel. The idea is to create a system which reads the image of a disk, creates blocks from the disk and finally performs the operation. Initially, the disk which is present on the secondary hard disk is read into the system i.e. main memory, there are 256 blocks in the system and each block is of 1024 bytes. A special structure called directory entry is created and they exist only in block 0. Later this special structure is exploited inorder to perform different tasks like creating a free list, deleting a file, renaming a file, enlisting contents of the directory, etc. Whenever an update is made or block 0 is changed the same should be reflected in the secondary storage. This is done by writing these directory entries back to the disk block 0.

Design:

The design of the system is pretty straight forward. However, one needs to get familiar with the design of the system. The following figure 1 can be helpful in visualizing the system that was implemented. Figure 2 shows the program structure for the cpmFS that is implemented. The directory entries in cpmFS form the heart of the design and implementation as the entire system is based upon the information stored in these directory entries.

Lessons Learned:

1. The file system designed is a simple yet elegant solution and after implementing it one can really understand why it was significant in the past
2. Though there are many block allocation and free block algorithm the Bit Map that is used in the current implementation is easy and suitable for our simple file system.
3. The directory entry structure is comprehensive, and one can find how the concept of Inode would have been evolved.
4. The blocks in the directory entries if occupied are unique to the file that holds it and cannot be shared.
5. The way free space, length of the file, the way name and extensions are stored in both directory entry and on the actual physical blocks is insightful as some development needs like string ending in a \0 is not taken care at the system level, but dealt by the file system.

6. The functions of a file system may not be limited to the one implemented as a file system takes care of a lot of important tasks and provides vibrant functionalities to the end user.

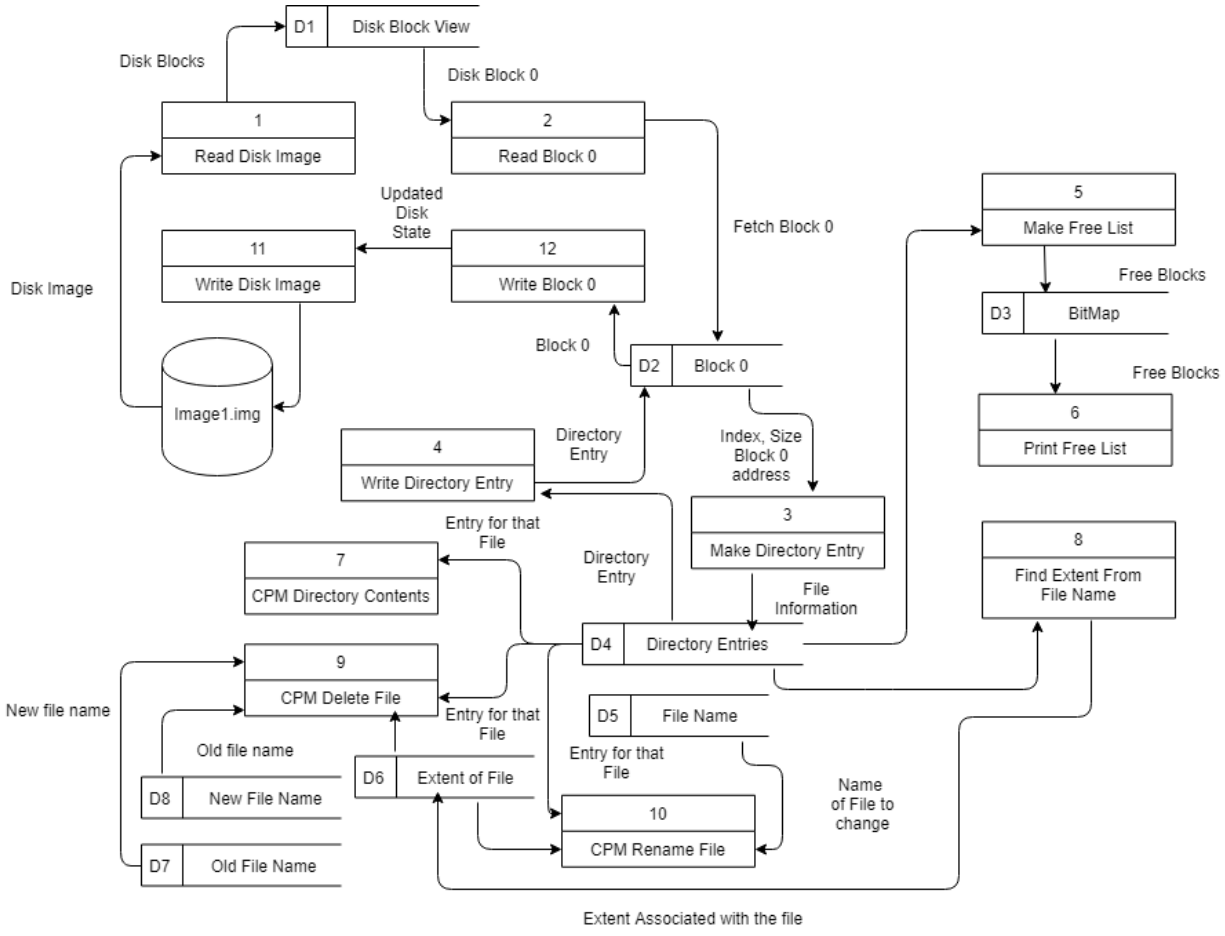

Figure 1: cpmFS Data Flow Diagram

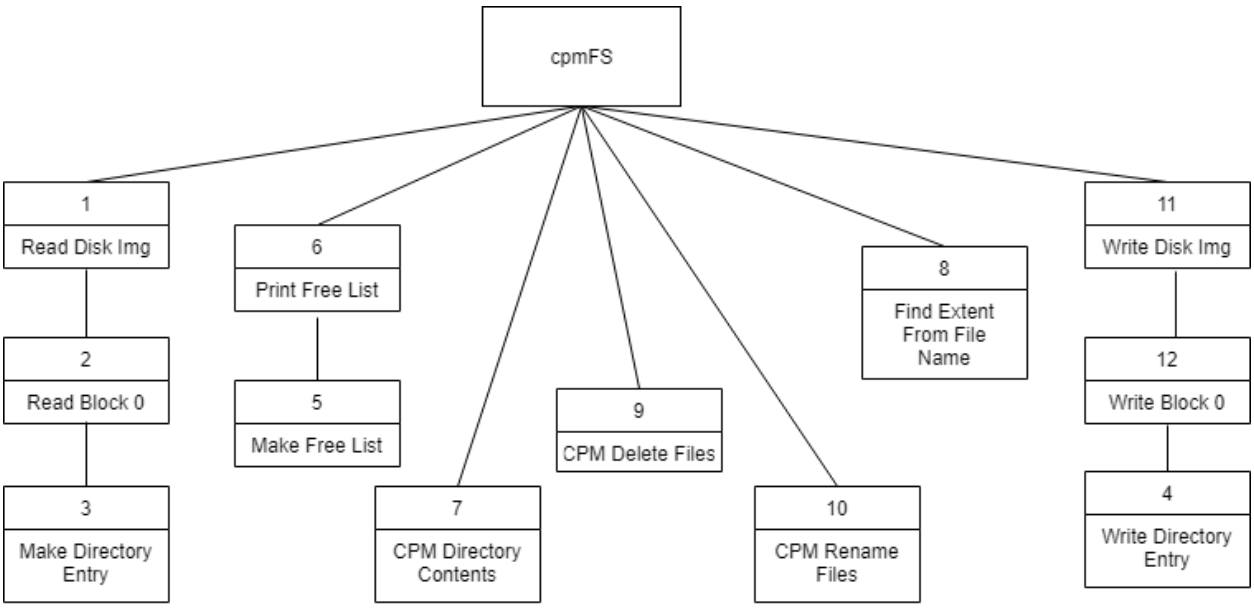

Figure 2: cpmFS Program Structure

Implementation:

The file cpmfsys.c contains the implementation for all of the above-enlisted modules. All the modules in the project are working and produce the desired results. Besides, an extensive amount of effort is taken to implement the deliverables in accordance with their strict declaration. Every block is of 1024 bytes and there are 256 such blocks. The Block 0 holds the metadata and must be kept consistent at all times, a corruption of Block 0 may lead to corrupting the entire system and thus failure of the cpmFS. In order to avoid this Block 0 is always assumed to be an assigned block and thus cannot be used by any other file. The directory entry is an important part of the file system and there are 32 such structures n the file system which can be found in the cpmFS Block 0.

The blocks are arranged as a matrix. Blocks are read into the system using read image function and can be stored using the write image function. Block 0 can be retrieved using the read block function that takes in two parameters (block number and buffer). The buffer supplied gives the content of the respective block. Similarly, a write can be performed using a write block function. A write to a Block 0 is necessary when some contents change for instance renaming a file, deleting a file and so on. Also, there is a function that provides the respective directory entry when we supply a file name. This comes in handy when other functions needs to retrieve whether a file name exists in the directory.

The output of the program is in accordance with the one supplied.

```
[stodkar@localhost Project 4]$ make
gcc -c fsysdriver.c
gcc -o cpmRun diskSimulator.o cpmfsys.o fsysdriver.o
[stodkar@localhost Project 4]$ ./cpmRun
```

```
DIRECTORY LISTING
mytestf1.txt 15874
holefile.txt 1152
shortf.ps 1032
mytestf. 1026
FREE BLOCK LIST: (* means in-use)
   0: * . * . * . * . * . * . * . * .
  10: * . . . . . . . . . . . . . . .
  20: * . . . * . . . . . . . . . . .
  30: * * . . . . . . . . . . . . . .
  40: . . . . . . . . . . . . . . . .
  50: . . . . . . . . . . . . . . . .
  60: . . . . . . . . . . . . . . . .
  70: . . . . . . . . . . . . . . . .
  80: . . . . . . . . . . . . . . . .
  90: . . . . . . . . . . . . . . . .
  a0: * . . . * . . . . . . . . . . .
  b0: . . . . . . . . . . . . . . . .
  c0: . . . . . . . . . . . . . . . .
  d0: . . . . . . . . . . . . . . . .
  e0: . . . . . . . . . . . . . . . .
  f0: . * . * . * . * . * . * . * . *
DIRECTORY LISTING
mytestf1.txt 15874
holefile.txt 1152
mytestf. 1026
cpmRename return code = 0,
DIRECTORY LISTING
mytest2.tx 15874
holefile.txt 1152
mytestv2.x 1026
FREE BLOCK LIST: (* means in-use)
   0: * . * . * . * . * . * . * . * .
  10: * . . . . . . . . . . . . . . .
  20: * . . . * . . . . . . . . . . .
  30: . . . . . . . . . . . . . . . .
  40: . . . . . . . . . . . . . . . .
  50: . . . . . . . . . . . . . . . .
  60: . . . . . . . . . . . . . . . .
  70: . . . . . . . . . . . . . . . .
  80: . . . . . . . . . . . . . . . .
  90: . . . . . . . . . . . . . . . .
  a0: * . . . * . . . . . . . . . . .
```

```
  b0: . . . . . . . . . . . . . . . .
  c0: . . . . . . . . . . . . . . . .
  d0: . . . . . . . . . . . . . . . .
  e0: . . . . . . . . . . . . . . . .
  f0: . * . * . * . * . * . * . * . *
[stodkar@localhost Project 4]$
```

Key Ideas Implemented:

- The idea of string tokenization implemented in Project 3 comes in handy when we need to extract file name from its extension
- Although there are many ways in which one could do the project I made use of global variables for bitmap and the block 0.

Running the Code

- Make use of the make file to generate the executable by issuing the command make
- Run the executable using ./cpmRun

Conclusion

Thus, we successfully implemented the cpmFS file system in C. All the modules of the project were implemented correctly, and the design document was investigated thoroughly in order to come up with the system. The lesson objectives were accomplished and thoroughly helped in understanding the cpmFS file system.