

# Project 4: cpmFS - A Simple File System

Satyam P. Todkar

[sz0064@auburn.edu](mailto:sz0064@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 need 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 in order 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:

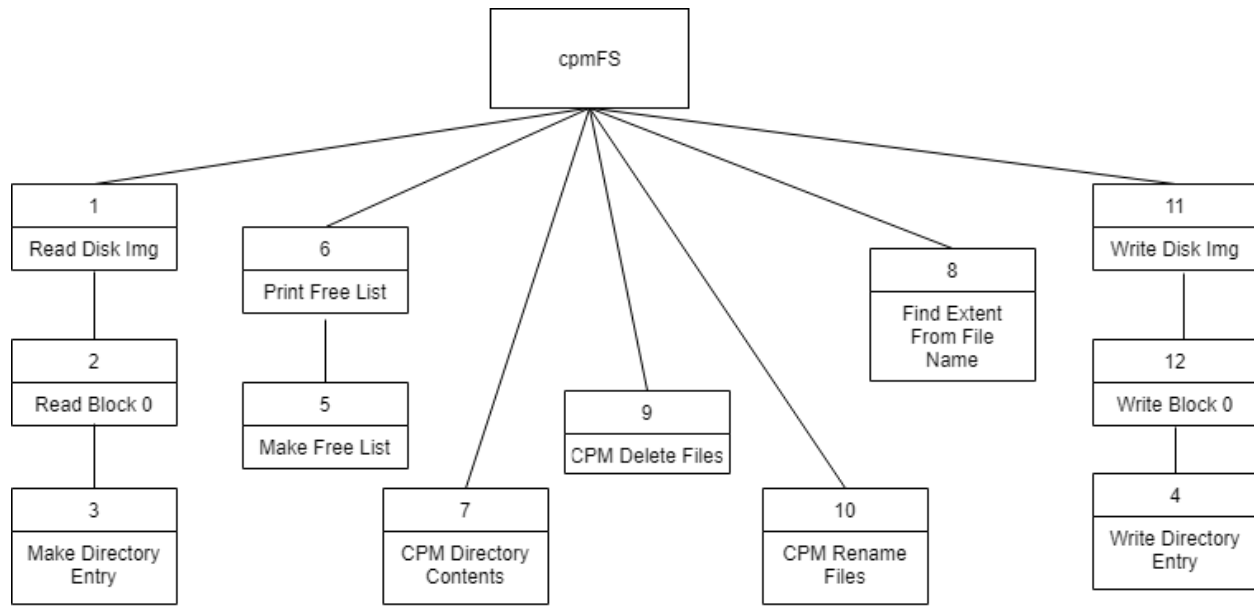
The design of the system is pretty straightforward. However, one needs to get familiar with the design of the system. The following figure 1 can be helpful in visualizing the system that was implemented. Figure 2 shows the program structure for the cpmFS that is implemented. The directory entries in 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 algorithms 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.

- 
- The flowchart illustrates the sequence of operations for the CPM File System, organized into three main sections: Disk Image Management, File Creation/Modification, and File Deletion/Rename.
- Disk Image Management:**
- Read Disk Image:** Operation 1, which takes **Disk Blocks** as input and outputs **Disk Block 0** to **Read Block 0** (Operation 2).
  - Write Disk Image:** Operation 11, which takes an **Image1.img** file and an **Updated Disk State** as input and outputs **Disk Blocks** back to **Read Disk Image**.
  - Write Block 0:** Operation 12, which takes **Disk Block 0** as input and outputs **Updated Disk State** to **Write Disk Image**.
- File Creation/Modification:**
- Make Directory Entry:** Operation 3, which takes **Block 0** (from **Read Block 0**) and **Index, Size Block 0 address** as input and outputs **File Information** to **Directory Entries** (D4).
  - Write Directory Entry:** Operation 4, which takes **Block 0** and **Directory Entry** as input and outputs **Entry for that File** to **CPM Directory Contents** (7).
  - Find Extent From File Name:** Operation 8, which takes **File Name** (D5) and **File Information** as input and outputs **Extent of File** (D6).
  - CPM Rename File:** Operation 10, which takes **Old File Name** (D7), **New File Name** (D8), and **Extent of File** as input and outputs **Entry for that File** to **CPM Delete File** (9).
- File Deletion/Rename:**
- CPM Delete File:** Operation 9, which takes **Entry for that File** as input and outputs **CPM Directory Contents** (7).
  - CPM Rename File:** Operation 10, which takes **Old File Name** (D7), **New File Name** (D8), and **Extent of File** as input and outputs **Entry for that File** to **CPM Delete File** (9).
- Free Blocks Management:**
- Make Free List:** Operation 5, which takes **Free Blocks** as input and outputs **Free Blocks** to **BitMap** (D3).
  - Print Free List:** Operation 6, which takes **Free Blocks** as input and outputs **Free Blocks** to **Make Free List** (5).
- Other Operations:**
- Fetch Block 0:** Operation 2, which takes **Disk Block 0** as input and outputs **Block 0** to **Make Directory Entry** (3).
  - Block 0:** A central data structure that receives **Disk Block 0** and **Block 0** and outputs to **Write Block 0** (12) and **Make Directory Entry** (3).
  - Directory Entry:** A central data structure that receives **Directory Entry** and **Entry for that File** and outputs to **Write Directory Entry** (4) and **CPM Directory Contents** (7).
  - File Information:** A central data structure that receives **File Information** and **File Name** (D5) and outputs to **Find Extent From File Name** (8).
  - Extent of File:** A central data structure that receives **Extent of File** and **File Name** (D5) and outputs to **CPM Rename File** (10).

### 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 in 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 need 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
  
```



```
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.