COMP 7500/7506 Advanced Operating Systems

Project 4: cpmFS - A Simple File System

Frequently Asked Questions

Document History: Created on April 25, 2019. Revised on April 12, 2021. Version 1.5

1. In cpmfsys.h file there are 5 functions at the bottom:
   - `cpmCopy`
   - `cpmOpen`
   - `cpmClose`
   - `cpmRead`
   - `cpmWrite`

   which says that need not to be implemented for project 4. Are they need to be implemented in project 4?

Answer: You don't need to implement the above five functions in project 4. Your source code file `cpmfsys.c` should include the implementations of the following nine functions:
- `mkDirStruct()`
- `writeDirStruct()`
- `makeFreeList()`
- `printFreeList()`
- `cpmDir()`
- `checkLegalName()`
- `findExtentWithName()`
- `cpmDelete()`
- `cpmRename()`

2. Are we supposed to use `diskSimulator.c` and other files from the course repository or are we supposed to build our own version of all of those files?

Answer: You only need to implement `cpmfsys.c` by making use of the existing `diskSimulator.c` and the other files from the course repository.

3. Do we ignore the actual execution time and use the execution time given by the run command? If the last sector is fully used, would it be save to assume that RC would have the

value of 8? This would indicate that all 1024 (8*128) bytes have been used. However, so far I do not see any RC with the value of 8.

Answer: RC is the number of sectors in the last partially-filled block. The maximum possible RC should be 7.

4. Regular expression aren't part of ANSI C. But I found that regex.h can do regular expression which can help us check the file name more easily. Can we use this header file?

Answer: You are allowed to include `regex.h` in your implementation.

5. When I try to inspect the disk image, it appears the only block with contents is block 0. Is this correct? Since block 0 refers to locations for file contents, I expected to find that actual content elsewhere in the disk image. It seems the project can be completed without having this content, but I was just checking to see if I was missing something.

Answer: You must exam the extents in block 0 to determine if there are allocated and used blocks (i.e., block address of 1-255). The information can be found in the last 16 bytes of each extent. I agree with you that this project might be completed without having the data content (i.e., blocks are allocated without containing any content).

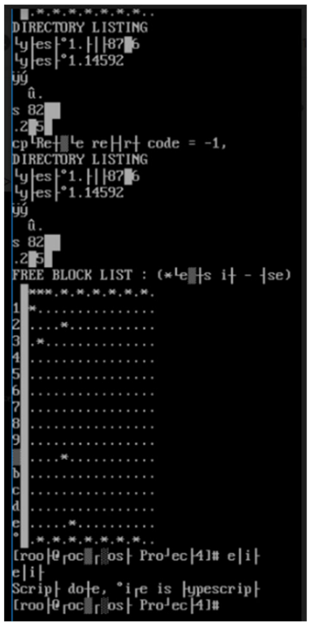
6. Why users must specify a job's CPU time priori to the job submission? Function cpmDir prints weired output. It also caused the centOS system a lot of errored displays. The solutions I tried as follows:
   1) I tried to download the image1.img again, and the error still shows.
   2) If I only print the first 5 lines of image1.img (or the first two files), no error in the output.
   3) If I use script to save the output, centOS error display are correct in typescript file, but the weired output for the file names are still there.

   Does anyone have the same issue?

   In saved typescript:

```
DIRECTORY LISTING
mytestf1.txt 8706
mytestf1. tx 14592
ÿýû.
s 8200
01 . 2050
```

Direct captured from centOS:



```
 █.*.*.*.*.*.*.*..
DIRECTORY LISTING
└y|es├°1.├||87█6
└y|es├°1.14592
ÿý
  û.
s 82█
.2█5█
cp└Re┼█└e re├┤r┼ code = -1,
DIRECTORY LISTING
└y|es├°1.├||87█6
└y|es├°1.14592
ÿý
  û.
s 82█
.2█5█
FREE BLOCK LIST : (*└e█┼s i┼ - ┤se)
 █***.*.*.*.*.*.*.
1█*...............
2█....*...........
3█.*..............
4█................
5█................
6█................
7█................
8█................
9█................
██....*...........
b█................
c█................
d█................
e█.....*..........
°█.*.*.*.*.*.*.*..
[roo|@┌oc█┌█os├ Pro┘ec|4]# e|i├
e|i├
Scrip├ do┼e, °i┌e is |ypescrip├
[roo|@┌oc█┌█os├ Pro┘ec|4]#
```

Answer:
(1) Your `cpmDir()` function might have a bug. Prior to printing an extent (say `d`) in block 0, your `cpmDir` must check the `status` of the extend (i.e., `d->status != 0xe5`). If `d->status == 0xe5`, then this extent is invalid. Don't print the extent if it is invalid.

(2) How did you display file names? A sample statement to print out file name and size is given below. We assume that you correctly calculate file size (i.e., file_size) prior to this statement.

```
printf("%s.%s %d\n", d->name, d->extension, file_size);
```

(3) Peijie Chen (Spring 2019) shared his programming experience with us as follows.
"I have encountered similar issues with you when I tried to use the tux machine (which is also a cent OS). But the problem disappears when I switched to windows and used cygwin64 to comply it. (I used Visual studio code to write my code) It may or may not solve your problem, but if you think your implementation is correct, I recommend you give a try."

7. How can I make use of `blockRead()` to retrieve a block from image1.img? (Spring'20)

Answer: A sample usage below demonstrates how to use the `blockRead()` to read a block.

```
uint8_t dirBlock[BLOCK_SIZE];

/* read the directory block from the disk */
blockRead(dirBlock,(uint8_t) 0);
```

8. Initialization of disk. In project4->disksimulator.c-> for the initialization of disk[][], it is described in comments that the 1st index is the sector number but it has been told in the class that a block contains 8 sectors of 128 bytes each. Also, the disk is considered as 256 blocks of 1024 bytes each. Kindly confirm if the following initialization means that the disk has 256 blocks of 1024 bytes each? (Spring'20)

Answer: Yes, this is a static data structure of two-dimentional array of 256 blocks, each of which has 1024 bytes.

9. **Printblock.** In disksimulator.c, in printblock(), why are we checking after 16 bytes and why do we use %4x and %2x, when we don't know the type of information(datatype) that image contains? (Spring'20)

Answer: The "16 bytes" mean that the function prints out 16 bytes per line. Please test this function and sense the usage of printblock().

10. **File System Size.** If each image contains 256 blocks in total. Block 0 has 1024/32=32 extents with 16 block information each. That means 16(blocks per file extent) * 32(no of file extents in block 0) = 512 blocks are needed to fill the data whereas only 255 blocks are available in the image. Hence, I would like to ask how does a file system deal with this? (Spring'20)

Answer: You are right. If all the extents are fully utilized and all files contains 16 blocks, the file system will be running out of space. To fix this problem, we must extend the file system size by increasing the number of blocks from 256 to at least 513. You don't have to address this concern in this project.

11. **Freelist.** In freelist, it was mentioned that a bit is reserved in every block to know if it is used or not. Hence, I am a little confused about how to convert it into a freelist[]. (Spring'20)

Answer: You may define a boolean data type first. The data structure of all the element in the freelist[ ] array is boolean (e.g., True = free, False = allocated).

12. **File Names.** I am confused with some instructions of bool checkLegalName(char *name) function. In instruction "there does not have to be any valid character in the extension – it can all be blanks" But in checkLegalName function it says "returns true for legal name (8.3 format)". (Spring'20)

Answer: Let me give you a few examples below:

mytestf1 (legal or true)

mytestf1.txt (legal or true)
mytestf11 (illegal or false)
mytestf1.txtt (illegal or false)

13. **DirStructType.** In the Project Specification it says “If the name is less than 8 bytes, the remaining bytes are padded with blanks (ASCII 32). There is no terminating ‘\0’ after the last valid character of the filename, just blanks”

    In lecture 33/handout it says “

    ```
    If name length < 8, filled by ' '
    obtain dir->name from in-memory block0
    add '\0' at the end of dir->name
    ```

    The project specification also says that the file extension can be all blanks, but the lecture says to add a '\0' at the end. Which should I do? (Spring’20)

Answer: File extents (i.e., direcotry entries) are represented in two different formats. In block 0, each extent is 32 bytes in length. However, when you read an extent from bloc 0 into an DirStructType variable using the mkDirStruct() function, the DirStructType variable has 34 bytes rather than 32 bytes. This is because we must add '\0' into the DirStructType variable for the file name and extension name, respectively.

In a nutshell, there are the two formats:

- Format 1: All 32-byte extents in block 0 have no ''\0" in file names and extension names.
- Format 2: All 34-byte DirStructType variables have "\0" attached at the end of file names and extension names.

14. **DirStructType.** The dirStruct definition is in cpmfsys.h. It has name defined as char name[9] and extension is defined as char extension[4], but the spec shows the name as being 8 characters long and the extension as being 3 characters long. The disk image matches the definition from the spec. I'm assuming there's an issue with the header file.

Answer: dirStruct is a data structure defined for in-memory block 0, which is slightly different from the in-disk block 0. The char array name[9] in dirStruct contains up to 8 chars and the terminator $\0$, which turns the char array into a string. Please refer to FAQ question 13 for more details.

15. **Makefile.** The makefile doesn't specify the c99 flag necessary to declare a variable inside of a for loop. Are we able to submit our own makefile with the c99 flag specified?

Answer: Yes, you should modify the makefile by specifying the c99 flag. Please submit a makefile that results in successful compilation of your code on the machines in the Linux lab: http://www.eng.auburn.edu/admin/ens/helpdesk/off-campus/access-information.html

16. **Image1.img.** What method should be used to view image1.img file? (Spring'21. Contributed by Mahmoud Abdalkarim)

Answer: "`hexdump -C image1.img`" in the command line of your Linux terminal.