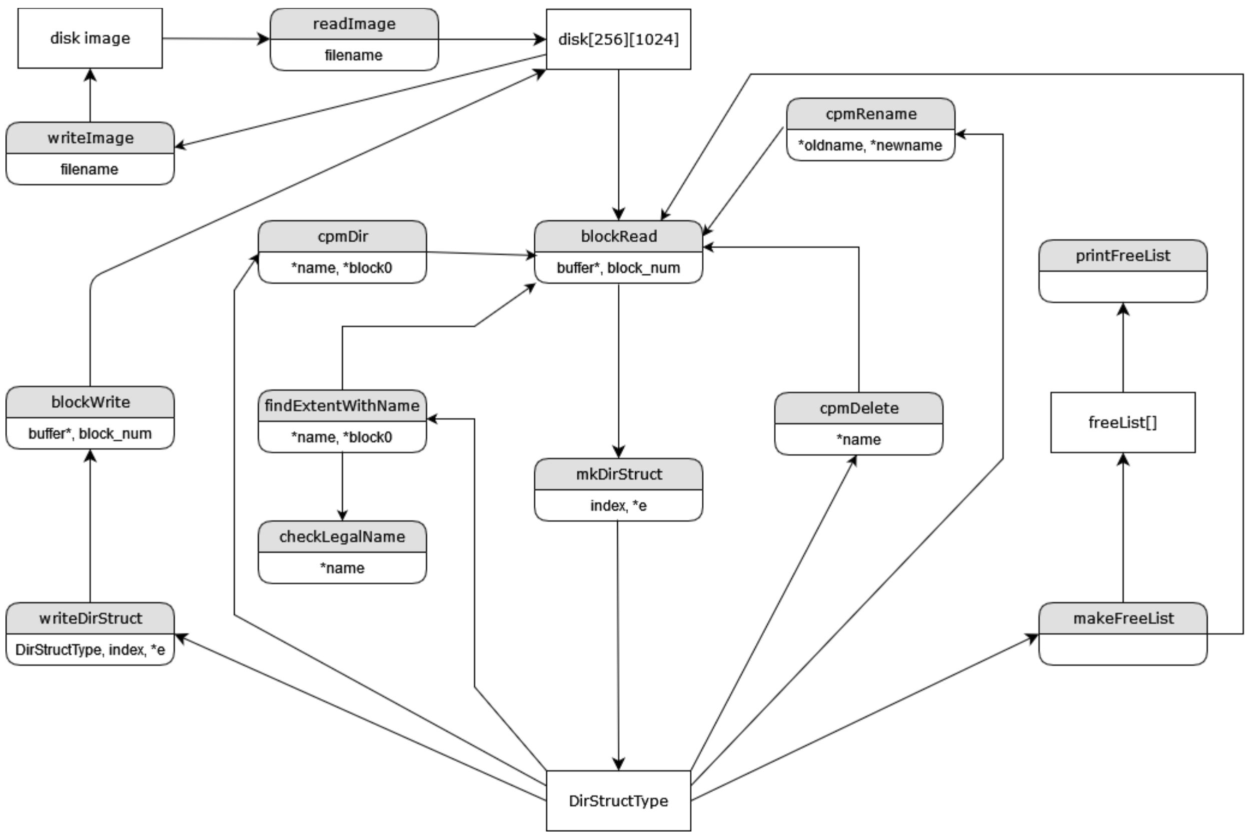
Matthew Shiplett (mss0033)
Comp-7500
Project 4 Report

Project 4: cpmFS - A Simple File System

Report

Design of cpmFS:

Dataflow Diagram:



Functions Design:

The majority of the cpmFS code was provided for this project, therefore any portion of the design directly ascribable to me is minimal. To be specific, I was provided all files except for the cpmfsys.c file. However, the provided files did include a cpmfsys.h file which defined useful constants, such as the extent size and the blocks per extent, as well as all the functions that are required to be implemented for this project. Therefore, my design is constrained to the specific implementations of the required functions. I will attempt to outline any 'design' work I did in order to implement the required functions. Please note that 'design' here often refers to how I decided to implement the function, rather than what I would consider a more traditional definition of design in regards to software.

- mkDirStruct():
    - For this function, little design was needed. The DirStruct is defined in the cpmfsys.h file, so this function simply needed to copy the information from a read in block (extent) to the struct. The largest contribution I feel I can say I made to the design of this function is that, for the implementation, a method to ensure that the copied file names and extensions do not contain superfluous leading and trailing white spaces is necessary. I therefore created a 'trim_string' function which trims leading and trailing whitespace.
- writeDirStruct():
    - Similar to mkDirStruct, I feel that little design for this function was required. The function simply needed to copy the information from a struct into a block on the simulated disk.
- makeFreeList():
    - For this function, the main design contribution I felt I made is deciding to have the FreeList be recomputed each time. This decision was made as a mitigation tool used to prevent the free list from being out of sync with post any operation. Each time makeFreeList is called, the entire free list is

recomputed with all blocks initialized as being free. Therefore, I do not need to track modifications made by other functions, or worry about calling makeFreeList or some sort of updateFreeList function each time a block is read from or written to.

- printFreeList():
  - I do not feel I can claim any 'design' contributions to this function. The function simply prints the global free list in the manner prescribed in the project 4 specification document.
- cpmDir():
  - I do not feel I can claim any 'design' contributions to this function. The function simply prints the directory in the manner prescribed in the project 4 specification document.
- checkLegalName():
  - For this function I feel the 'design' claim I can make is in regards to what constituted an illegal character. I was unable to locate and specific set of illegal characters in the project 4 specification. Therefore, I decided that only the follow character sets where valid: {a, z}, {A, Z}, {0, 9}, {_}. All other characters are determined to be invalid.
- findExtentWithName()
  - I do not feel I can claim any 'design' contributions to the function. The function simply returns the extent number of a file with a matching name, or -1 if the file is not found, in accordance with the project 4 specification document.

- cpmDelete():
    - I do not feel I can claim any 'design' contributions to the function. The function simply marks a file status as un-used, and marks the file's disk blocks as free, if a file with a matching name is found in the directory. This in accordance with the behavior specified in the project 4 specification document.
- cpmRename():
    - I feel that I can claim only a minor 'design' contribution here insofar as that I decided that when a valid new file name is provided, the previous file name is first 'deleted'. This is carried out by replacing the 9 characters of the dir struct name and 4 characters of the dir struct extension with the C termination character '\0'. This prevents portions of the old file name or extension from being present in the renamed file name, or extension, should the new name, or extension, be shorter than the old name, or extension.

Lessons Learned:

For this project, I feel I learned a few lessons. The first is that code which causes segmentation faults and accidentally modifying a string literal are easier than I would have thought to implement. Another lesson is that when copying data that has a fixed maximum size, but the exact size is not known, can be tricky. One has to do post-processing on the copied data to ensure it complies with the standard(s) set out in the design. For example, I spent considerable time with the mkDirStruct and cpmRename functions. For mkDirStruct I had to figure out how to prevent the file names and extensions from having trailing whitespace when the file name was shorter than the maximum 8 allowed name characters. For cpmRename I had to spend time ensuring that if the new name was shorter than the old name, portions of the old name where not present in the renamed file. Lastly, I learned that even being given the formula for how to calculate file size, I still had a small Off-by-One issue where I count an extra block for each file.

Conclusion:

Completing this project has given me newfound respect for the difficulties and intricacies of file systems.