Project 4 Report

Tanner Bonds

COMP-7500

04/22/2021

Design:

The cpmFS program is designed to simulate a simple file system known as the CP/M file system. The program will allow users to perform actions of file systems such as creating, copying, deleting, and renaming files and directories. Other actions include reading, writing, opening, and closing files. Myself and the other students of COMP 7500 were given 7 files to work with. Six of those files were source code and a Makefile, and one was a sample of cpmFS' input and output. Ignoring the Makefile, Professor Qin provided us with five source code files relevant to cpmFS. These files are: "cpmfsys.h," "diskSimulator.h," "diskSimulator.c," "fsysdriver.c," and "image1.img." The "image1.img" file contains a "disk" image that we will use to simulate a data storage device. The "diskSimulator.c/h" files provide the functions for interacting with simulated hardware. The "fsysdriver.c" file contains the primary execution code and the "main" function for cpmFS. Our goal in Project 4 is to implement functions to complete the functionality of cpmFS. The nine functions we were required to implement are prototyped in "cpmfsys.h."

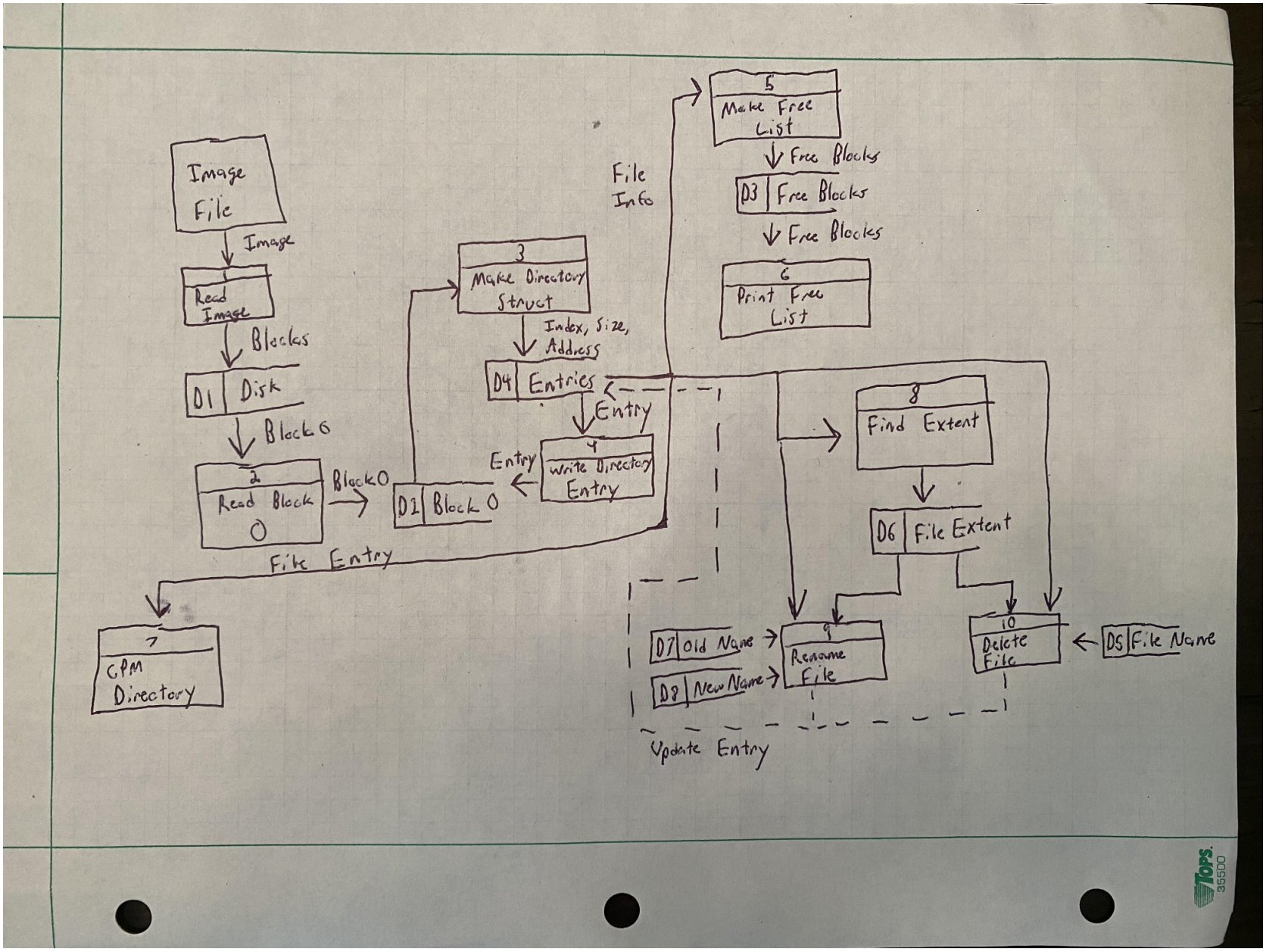
Image File
Image
1
Read Image
Blocks
D1 Disk
Block 0
2
Read Block 0
Block 0
D1 Block 0
3
Make Directory Struct
Index, Size, Address
D4 Entries
Entry
4
Write Directory Entry
Entry
File Entry
7
CPM Directory
File Info
5
Make Free List
Free Blocks
D3 Free Blocks
Free Blocks
6
Print Free List
8
Find Extent
D6 File Extent
D7 Old Name
D8 New Name
9
Rename File
10
Delete File
D5 File Name
Update Entry

Implementation:

Within the “cpmfsys.c” file, there were nine functions we were required to implement. These nine functions are titled “mkDirStruct,” “writeDirStruct,” “makeFreeList,” “printFreeList,” “cpmDir,” “findExtentWithName,” “checkLegalName,” “cpmRename,” and “cpmDelete.” The “mkDirStruct” function takes as input an index and an address and determines which extent from block zero to use to create a DirStructType. The function “writeDirStruct” will take the contents of a DirStructType and then write it to memory. The function “makeFreeList” will fill the FreeList data structure, titled “free_blk” in this program. This data structure shows the current blocks of memory that are free and which blocks are in use. The function “printFreeList” simply prints the FreeList data structure to stdout. The function “cpmDir” will print the current directory of files to stdout. The filenames for files have specific sets of requirements, such as limits on the number of characters. The function “checkILegalName” checks these parameters, as well as others, such as making sure a file name is not blank or has illegal characters, and returns a boolean value indicating if a file has a legal name. The function “findExtentWithName” checks to see if a passed string matches a directories filenames and extensions. Finally, the function “cpmRename” renames a file by modifying its extent and then writes it to the disk. Finally, the function “cpmDelete” deletes a file and frees disk blocks from the FreeList.

Output:

```
tjbonds  ~/comp_7500/project4  ./cpmRun
DIRECTORY
mytestf1.txt 15874
holefile.txt 1152
shortf.ps 1032
mytestf. 1026

BLOCK LIST (* = IN USE)
 0: * . * . * . * . * . * . * . * .
10: * . . . . . . . . . . . . . . .
20: * . . . * . . . . . . . . . . .
30: * * . . . . . . . . . . . . . .
40: . . . . . . . . . . . . . . . .
50: . . . . . . . . . . . . . . . .
60: . . . . . . . . . . . . . . . .
70: . . . . . . . . . . . . . . . .
80: . . . . . . . . . . . . . . . .
90: . . . . . . . . . . . . . . . .
a0: * . . . * . . . . . . . . . . .
b0: . . . . . . . . . . . . . . . .
c0: . . . . . . . . . . . . . . . .
d0: . . . . . . . . . . . . . . . .
```

```
e0: . . . . . .
f0: . * . * . * . * . * . * . * . *

DIRECTORY
mytestf1.txt 15874
holefile.txt 1152
mytestf. 1026
cpmRename return code = 0,
DIRECTORY
mytest2.tx 15874
holefile.txt 1152
mytestv2.x 1026

BLOCK LIST (* = IN USE)
 0: * . * . * . * . * . * . * . * .
10: * . . . . . .
20: * . . . * . . . . . .
30: . . . . . .
40: . . . . . .
50: . . . . . .
60: . . . . . .
70: . . . . . .
80: . . . . . .
90: . . . . . .
a0: * . . . * . . . . . .
b0: . . . . . .
c0: . . . . . .
d0: . . . . . .
e0: . . . . . .
f0: . * . * . * . * . * . * . * . *

tjbonds  ~/comp_7500/project4
```

Lessons Learned:

Interestingly, I found this project easier to implement than Project 3, the AUBatch Project. I am unsure if this is due to understanding the course material and assignment better, or if I was simply less overwhelmed by the scope of the project; as this project required significantly fewer work hours and lines of code than Project 3. I also find that I am becoming more proficient with separate compilation and Makefiles, and I correlate this directly to learning their usage in class. Unfortunately, I still have issues with working with GDB. I have yet to fully understand all the functionalities of GDB and have difficulty tracing errors. Finally, the implementation of functions in C, specifically related to memory management and pointers, remains a topic that I have yet to fully grasp.

Source Code:

```
/* Name: Tanner Bonds
 * Date: 04/11/2022
 * COMP 7500
 * Project 4
 * Referenced Dr. Qin's Example Code.
 */

#include "cpmfsys.h"
#include <ctype.h>

bool free_blk[NUM_BLOCKS];
uint8_t buf[BLOCK_SIZE];

DirStructType *mkDirStruct(int index, uint8_t *e)
{
   DirStructType *cpm;
   uint8_t *addr;

   cpm = malloc(sizeof(DirStructType));
   if (cpm == NULL)
   {
      fprintf(stderr, "Space Allocation Failed\n");
      exit(-1);
   }

   addr = e + index * EXTENT_SIZE;
   cpm->status = addr[0];

   for (int i = 0; i < 9; i++)
   {
      if (i != 8)
      {
         cpm->name[i] = ' ';
      }
      else
      {
         cpm->name[i] = '\0';
      }
   }

   for (int i = 1; i < 9; i++)
   {
```

```
    if (addr[i] != ' ' && addr[i])
    {
        cpm->name[i - 1] = addr[i];
    }
    else
    {
        cpm->name[i - 1] = '\0';
        break;
    }
}

if (strlen(cpm->name) == 0)
{
    cpm->name[0] = '\0';
}

for (int i = 0; i < 4; i++)
{
    if (i != 3)
    {
        cpm->extension[i] = ' ';
    }
    else
    {
        cpm->extension[i] = '\0';
    }
}

for (int i = 9; i < 12; i++)
{
    if (addr[i] != ' ' && addr[i])
    {
        cpm->extension[i - 9] = addr[i];
    }
    else
    {
        cpm->extension[i - 9] = '\0';
        break;
    }
}

if (strlen(cpm->extension) == 0)
{
    cpm->extension[0] = '\0';
```

```
    }

    cpm->XL = addr[12];
    cpm->BC = addr[13];
    cpm->XH = addr[14];
    cpm->RC = addr[15];

    for (int i = 16; i < 32; i++)
    {
        cpm->blocks[i - 16] = addr[i];
    }
    return cpm;
}

void writeDirStruct(DirStructType *d, uint8_t index, uint8_t *e)
{
    uint8_t *addr;
    addr = e + index * EXTENT_SIZE;
    addr[0] = d->status;

    for (int i = 1; i < 9; i++)
    {
        addr[i] = ' ';
    }
    for (int i = 1; i < 9; i++)
    {
        addr[i] = d->name[i - 1];
    }
    for (int i = 9; i < 12; i++)
    {
        addr[i] = ' ';
    }
    for (int i = 9; i < 12; i++)
    {
        addr[i] = d->extension[i - 9];
    }
    addr[12] = d->XL;
    addr[13] = d->BC;
    addr[14] = d->XH;
    addr[15] = d->RC;

    for (int i = 16; i < 32; i++)
    {
        addr[i] = d->blocks[i - 16];
```

```
    }

    blockWrite(e, 0);
}

void makeFreeList()
{
    DirStructType *cpm;

    blockRead(buf, 0);
    free_blk[0] = false;
    for (int i = 1; i < NUM_BLOCKS; i++)
    {
        free_blk[i] = true;
    }

    for (int i = 0; i < EXTENT_SIZE; i++)
    {
        cpm = mkDirStruct(i, buf);
        if (cpm->status != 0xe5)
        {
            for (int j = 0; j < 16; j++)
            {
                if ((int)cpm->blocks[j] != 0)
                {
                    free_blk[(int)cpm->blocks[j]] = false;
                }
            }
        }
    }
}

void printFreeList()
{
    fprintf(stdout, "\nBLOCK LIST (* = IN USE)\n");
    for (int i = 0; i < NUM_BLOCKS; i++)
    {
        if (i % 16 == 0)
        {
            fprintf(stdout, "%2x: ", i);
        }
        if (free_blk[i] == true)
        {
            fprintf(stdout, "%s ", ".");
```

```
        }
        else
        {
            fprintf(stdout, "%s ", "*");
        }
        if (i % 16 == 15)
        {
            fprintf(stdout, "\n");
        }
    }
    fprintf(stdout, "\n");
}

int findExtentWithName(char *name, uint8_t *block0)
{
    DirStructType *cpm;
    char fname[9];
    char fextent[4];
    int i = 0;

    char *tok = strtok(name, ".");
    while (tok != NULL)
    {
        i++;
        switch (i)
        {
        case 1:
            if (strlen(tok) <= 8)
            {
                strcpy(fname, tok);
            }
            else
            {
                return -1;
            }
            break;

        case 2:
            if (strlen(tok) <= 3)
            {
                strcpy(fextent, tok);
            }
            else
            {
```

```
            return -1;
        }
        break;
    }
    tok = strtok(NULL, ".");
}

if (checkLegalName(fname))
{
    for (int i = 0; i < 32; i++)
    {
        cpm = mkDirStruct(i, buf);
        if ((strcmp(fname, cpm->name) == 0 && strcmp(fextent, cpm->extension) == 0) ||
(strcmp(fname, cpm->name) == 0))
        {
            if (cpm->status == 0xe5)
            {
                return -1;
            }
            else
            {
                return i;
            }
        }
    }
}
else
{
    fprintf(stderr, "Invalid Filename or Extension");
}

//    {
//        return -1;
//    }
//    for (int i = 0; i < EXTENT_SIZE; i++)
//    {
//        cpm = mkDirStruct(i, block0);
//        if (strcmp(cpm->name, name) == 0)
//        {
//            return i;
//        }
//    }
return -1;
}
```

```
bool checkLegalName(char *name)
{

   for (int i = 0; i < strlen(name); i++)
   {
      if (isspace(name[i]) || ispunct(name[i]) || iscntrl(name[i]))
         return false;
   }

   return true;
}

void cpmDir()
{
   DirStructType *cpm;
   int block, len;

   blockRead(buf, 0);
   printf("DIRECTORY\n");

   for (int i = 0; i < EXTENT_SIZE; i++)
   {
      cpm = mkDirStruct(i, buf);
      if (cpm->status != 0xe5)
      {
         block = 0;
         len = 0;
         for (int j = 0; j < 16; j++)
         {
            if (cpm->blocks[j] != 0)
            {
               block++;
            }
         }
         len = ((block - 1) * 1024) + ((int)cpm->RC * 128) + (int)cpm->BC;
         printf("%s.%s %i\n", cpm->name, cpm->extension, len);
      }
   }
}

int cpmRename(char *oldName, char *newName)
{
   char old[20];
```

```
char new[20];
strcpy(old, oldName);
strcpy(new, newName);
blockRead(buf, 0);
int extent = findExtentWithName(old, buf);

char fname[9];
char fextent[4];
DirStructType *cpm;
int i = 0;

if (extent >= 0)
{
   char *tok = strtok(new, ".");
   while (tok != NULL)
   {
      i++;
      switch (i)
      {
      case 1:
         if (strlen(tok) < 9)
         {
            strcpy(fname, tok);
         }
         else
         {
            return -1;
         }
         break;
      case 2:
         if (strlen(tok) < 4)
         {
            strcpy(fextent, tok);
         }
         else
         {
            return -1;
         }
         break;
      }
      tok = strtok(NULL, ".");
   }
   cpm = mkDirStruct(extent, buf);
   strcpy(cpm->name, fname);
```

```
        strcpy(cpm->extension, fextent);
        writeDirStruct(cpm, extent, buf);
    }
    else
    {
        return -1;
    }

    return 0;
}

int cpmDelete(char *name)
{
    char cname[20];
    strcpy(cname, name);
    int extent = findExtentWithName(cname, 0);
    char fname[9];
    char fextent[4];
    DirStructType *cpm;
    int i = 0;
    // printf("%i\n", extent);
    if (extent >= 0)
    {
        char *tok = strtok(cname, ".");
        while (tok != NULL)
        {
            i++;
            switch (i)
            {
            case 1:
                if (strlen(tok) < 9)
                {
                    strcpy(fname, tok);
                }
                else
                {
                    return -1;
                }
                break;
            case 2:
                if (strlen(tok) < 4)
                {
                    strcpy(fextent, tok);
                }
```

```
            else
            {
                return -1;
            }
            break;
        }
        tok = strtok(NULL, ".");
    }
    cpm = mkDirStruct(extent, buf);
    cpm->status = 0xe5;
    strcpy(cpm->name, " ");
    strcpy(cpm->extension, " ");
    cpm->XL = 0;
    cpm->BC = 0;
    cpm->XL = 0;
    cpm->RC = 0;

    uint8_t del = 0;
    for (int i = 0; i < 16; i++)
    {
        del = cpm->blocks[i];
        if (del > 0)
        {
            free_blk[del] = true;
        }
    }
    writeDirStruct(cpm, extent, buf);
  }
  return 0;
}

// int cpmCopy(char *oldName, char *newName)
// {
//     return 0;
// }

// int cpmOpen(char *fileName, char mode)
// {
//     return 0;
// }

// int cpmClose(int filePointer)
// {
//     return 0;
```

```
// }

// int cpmRead(int pointer, uint8_t *buffer, int size)
// {
//     return 0;
// }

// int cpmWrite(int pointer, uint8_t *buffer, int size)
// {
//     return 0;
// }
```