CS3026 2018

OPERATING SYSTEMS ASSESSMENT

51660004

Task-Brief:

Implement a simple file system that allows you to manage files and directories in a virtual in-memory disk. The file system is to be based on simplified concepts of a File Allocation Table (FAT). Your implementation will allow the creation of files and directories within this virtual hard disk and the performance of simple read and write operations on files.

CGS D3_D1

Implementation of method **format()** to create structure for virtual disk.

The block 0, among other blocks is used for metadata of the disk. The block is created with Course Assessment information.

```
/* prepare block 0 : fill it with '\0',
* use strcpy() to copy some text to it for test purposes
* write block 0 to virtual disk
*/
for (int i = 0; i < BLOCKSIZE; i++) block.data[i] = '\0';
strcpy(block.data, "CS3026 Operating Systems Assessment");
writeblock(&block, 0);
```

Block 1 and 2 has details about data in other blocks, i.e File Allocation Table (FAT.) We implement this before creating root directory. The entries of FAT is set to UNUSED initially and then details about Block 1 and 2 chaining is added.

```
/* prepare FAT table
* write FAT blocks to virtual disk
*/
for (int i = 0; i < BLOCKSIZE; i++) FAT[i] = UNUSED;
FAT[0] = ENDOFCHAIN;
FAT[1] = 2;
FAT[2] = ENDOFCHAIN;

FAT[3] = ENDOFCHAIN;

// Copies he FAT to Virtual Disk blocks
copyFAT();
```

Before we create the root directory, we need to write FAT changes to the memory.

copyFAT()

```
void copyFAT() {
  diskblock_t block;
  unsigned int numOfFatBlocks;
  numOfFatBlocks = (unsigned int)(MAXBLOCKS/FATENTRYCOUNT);
  int i, j;
  for (i = 0; i < numOfFatBlocks; i++) {
    for (j = 0; j < FATENTRYCOUNT; j++) {
      block.fat[j] = FAT[((i*FATENTRYCOUNT)+j)];
    }
    writeblock(&block, i + 1);
  }
}
```

Now, the root directory is created in Block 3 and details about root dir is added to the FAT.

```
/* prepare root directory
* write root directory block to virtual disk
*/
for (int i = 0; i < BLOCKSIZE; i++) block.data[i] = '\0';
block.dir.isdir = 1;
block.dir.nextEntry = 0;


for (int i = 0; i < DIRENTRYCOUNT; i++) block.dir.entrylist[0].unused = TRUE;

writeblock(&block, 3);

rootDirIndex = 3;
```

shell.c

We will use shell.c to test our file system we have created.

```
void main(int argc, char const *argv[]) {
  format();
  writedisk("virtualdiskD3_D1");

}
```

Output:

```
[hans@HanJaroD CGS_D3_D1]$ make
gcc -o shell shell.c filesys.c
[hans@HanJaroD CGS_D3_D1]$ ./shell
writedisk> virtualdisk[0] = CS3026 Operating Systems Assessment
[hans@HanJaroD CGS_D3_D1]$ hexdump -C virtualdiskD3_D1
00000000  43 53 33 30 32 36 20 4f  70 65 72 61 74 69 6e 67  |CS3026 Operating|
00000010  20 53 79 73 74 65 6d 73  20 41 73 73 65 73 73 6d  | Systems Assessm|
00000020  65 6e 74 00 00 00 00 00  00 00 00 00 00 00 00 00  |ent.............|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000400  00 00 02 00 00 00 00 00  ff ff ff ff ff ff ff ff  |................|
00000410  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00000c00  01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000c10  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00100000
```

00000000 to 00000400 is Block 0, contains the metadata.

00000400 to 00000c00 is Block 1,2 i.e FAT.

Block 3 starts at 00000c00, where root directory exists.

CGS_C3_C1:

Task-Brief:

Implementation of standard interface functions for file creation, and modification.

All the files are assumed to be created at root directory. The test file is filled with random text of 4*BLOCKSIZE data.

myfopen():

File is dynamically allocated to the type of MyFILE and of size MyFIle. The file I/O type is set to the filedescriptor mode variable. Block 3 is loaded to the memory, and entrylist is checked for any pre-existing files with same name, if found the position is stored in **pos** and returned true for next step.

```
diskblock_t block;
int pos = 0;
int f = FALSE;

// Allocate file space
MyFILE * file = malloc(sizeof(MyFILE));

// Copy the mode to file meta data
strcpy(file->mode, mode);

// load the block
block = virtualDisk[rootDirIndex];


// Loop in possible 3 DirEntries to see if file exists
for(int i = 0; i < DIRENTRYCOUNT; i++) {
  if (strcmp(block.dir.entrylist[i].name, filename) == 0) {
    f = TRUE;
    pos = i;
    break;
  }
}
```

Entrylist is checked for UNUSED entries, FAT is checked for UNUSED entries and then which the **pos** stored in filedescriptor. FAT is updated with position **pos.** Filename is stored in dir metadata, and block. The final details is written to disk.

```
// Create File metadata if file doesn't exist
if (f != TRUE) {
  int dir;
  for (dir = 0; dir < DIRENTRYCOUNT; dir++) {
    if (block.dir.entrylist[dir].unused == TRUE) break;
  }
  for (pos = 0; pos < MAXBLOCKS; pos++)
    if (FAT[pos] == UNUSED) break;
  FAT[pos] = ENDOFCHAIN;
  file->blockno = pos;
  block.dir.entrylist[dir].firstblock = pos;
  copyFAT();
  strcpy(block.dir.entrylist[dir].name, filename);
  block.dir.entrylist[dir].unused = FALSE;

  writeblock(&block, rootDirIndex);
}
else {
  file->blockno = block.dir.entrylist[pos].firstblock;
  file->pos = 0;
}

return file;
```

P.T.O.

myfputc()

Writes byte to file, if mode is in write mode.

```
if (strcmp(stream->mode, "r") == 0)
        return;

int zpos;
int i = 0;
int f = FALSE;

zpos = stream->blockno;

while(TRUE) {
  if (FAT[zpos] == ENDOFCHAIN) {
    f = TRUE;
    break;
  } else {
    zpos = FAT[zpos];
  }
}

stream->buffer = virtualDisk[zpos];

//finds end position of data in block
for(i=0; i<BLOCKSIZE; i++){
  if(stream->buffer.data[i] == '\0'){
    //pos is the position in the block that is free/where to start placing more data
    stream->pos = i;
    break;
  }
}
```

The variable **zpos**, consists the position of first block of the file.

ENDOFCHAIN is found. The character is inserted to the block unless it is full.

```
// add new data to the open file block
stream->buffer.data[stream->pos]= (Byte) b;

// write buffer block to the virtualDisk
writeblock(&stream->buffer, zpos);
// increment end position
stream->pos++;

/ looks to see if at the end of block and finds next free pos in FAT
if(stream->pos==BLOCKSIZE){
  stream->pos=0;
  for(i=0; i<BLOCKSIZE; i++)
    if(FAT[i]==UNUSED)
      break;

  //set next position in fat and write to virtual disk
  FAT[zpos] = i;
  FAT[i] = ENDOFCHAIN;
  copyFAT();
}

//clear the buffer block for new data
for(i=0; i<MAXBLOCKS; i++)
  stream->buffer.data[i] = '\0';
```

The stream is written to disk with respect to BLOCKSIZE.

myfgetc()

Returns the next byte of the open file, or EOF.

```
int myfgetc(MyFILE *stream) {
  if (stream->blockno == ENDOFCHAIN || strcmp(stream->mode, "r") != 0)
    return EOF;

  if (stream->pos % BLOCKSIZE == 0) {
    memcpy(&stream->buffer, &virtualDisk[stream->blockno], BLOCKSIZE);
    stream->blockno = FAT[stream->blockno];
    stream->pos = 0;
  }
  return stream->buffer.data[stream->pos++];
}
```

If the mode is read-only or EOF the function returns void, otherwise returns the buffer block by block in BLOCKSIZE.

myfclose()

```
void myfclose(MyFILE *stream) {
  int next;
  for (int i = rootDirIndex + 1; i < MAXBLOCKS; i++) {
    if (FAT[i] == UNUSED) {
      next = i;
      break;
    }
  }

  FAT[next] = ENDOFCHAIN;
  copyFAT();
  free(stream);

}
```

Closes the file, writes out any blocks not written to disk.

The ENDOFCHAIN is initiated in the respective place block and written to FAT.

Memory is freed.

Shell.c : testing

zfile is set of type MyFILE, myfopen() returns the type MyFILE with filedescriptor details. The file is open.

The myfputc is passed with character of **al,** which consists of alphabet of type char array. The char is written to buffer and the disk byte by byte of size **4*BLOCKSIZE**. The file is closed.

```
// CGS C
printf("virtualdiskC3_C1: \n");
MyFILE * zfile = myfopen("testfile.txt", "w");

char *al = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

int k = 0;
for (int i = 0; i < (4*BLOCKSIZE); i++) {
  if (k == 26)
    k = 0;
  myfputc(al[k], zfile);
  k++;
}

myfclose(zfile);

// Testing the written file using myfgetc()
FILE * txt = fopen("testfileC3_C1_copy.txt", "w");
MyFILE * file = myfopen("testfile.txt", "r");
char c;
while (c != EOF) {
  c = myfgetc(file);
  if (c != EOF) {
    fprintf(txt, "%c", c);
    printf("%c", c);
  } else printf("\n");
}
myfclose(file);
fclose(txt);

writedisk("virtualdiskC3_C1");
```

myfgetc() returns BLOCKSIZE of blocks, which is written to test file and printed to the screen and after which the file is closed and written to the disk.

OUTPUT:

```
[hans@HanJaroD CGS_C3_C1]$ hexdump -C virtualdiskC3_C1
00000000  43 53 33 30 32 36 20 4f  70 65 72 61 74 69 6e 67  |CS3026 Operating|
00000010  20 53 79 73 74 65 6d 73  20 41 73 73 65 73 73 6d  | Systems Assessm|
00000020  65 6e 74 00 00 00 00 00  00 00 00 00 00 00 00 00  |ent.............|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000400  00 00 02 00 00 00 00 00  05 00 06 00 07 00 08 00  |................|
00000410  00 00 00 00 00 00 ff ff  ff ff ff ff ff ff ff ff  |................|
00000420  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
*
00000c00  01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000c10  00 00 00 00 00 00 00 00  00 00 00 00 04 00 74 65  |..............te|
00000c20  73 74 66 69 6c 65 2e 74  78 74 00 00 00 00 00 00  |stfile.txt......|
00000c30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001000  41 42 43 44 45 46 47 48  49 4a 4b 4c 4d 4e 4f 50  |ABCDEFGHIJKLMNOP|
00001010  51 52 53 54 55 56 57 58  59 5a 41 42 43 44 45 46  |QRSTUVWXYZABCDEF|
00001020  47 48 49 4a 4b 4c 4d 4e  4f 50 51 52 53 54 55 56  |GHIJKLMNOPQRSTUV|
00001030  57 58 59 5a 41 42 43 44  45 46 47 48 49 4a 4b 4c  |WXYZABCDEFGHIJKL|
00001040  4d 4e 4f 50 51 52 53 54  55 56 57 58 59 5a 41 42  |MNOPQRSTUVWXYZAB|
00001050  43 44 45 46 47 48 49 4a  4b 4c 4d 4e 4f 50 51 52  |CDEFGHIJKLMNOPQR|
00001060  53 54 55 56 57 58 59 5a  41 42 43 44 45 46 47 48  |STUVWXYZABCDEFGH|
00001070  49 4a 4b 4c 4d 4e 4f 50  51 52 53 54 55 56 57 58  |IJKLMNOPQRSTUVWX|
00001080  59 5a 41 42 43 44 45 46  47 48 49 4a 4b 4c 4d 4e  |YZABCDEFGHIJKLMN|
00001090  4f 50 51 52 53 54 55 56  57 58 59 5a 41 42 43 44  |OPQRSTUVWXYZABCD|
000010a0  45 46 47 48 49 4a 4b 4c  4d 4e 4f 50 51 52 53 54  |EFGHIJKLMNOPQRST|
000010b0  55 56 57 58 59 5a 41 42  43 44 45 46 47 48 49 4a  |UVWXYZABCDEFGHIJ|
000010c0  4b 4c 4d 4e 4f 50 51 52  53 54 55 56 57 58 59 5a  |KLMNOPQRSTUVWXYZ|
000010d0  41 42 43 44 45 46 47 48  49 4a 4b 4c 4d 4e 4f 50  |ABCDEFGHIJKLMNOP|
000010e0  51 52 53 54 55 56 57 58  59 5a 41 42 43 44 45 46  |QRSTUVWXYZABCDEF|
000010f0  47 48 49 4a 4b 4c 4d 4e  4f 50 51 52 53 54 55 56  |GHIJKLMNOPQRSTUV|
00001100  57 58 59 5a 41 42 43 44  45 46 47 48 49 4a 4b 4c  |WXYZABCDEFGHIJKL|
00001110  4d 4e 4f 50 51 52 53 54  55 56 57 58 59 5a 41 42  |MNOPQRSTUVWXYZAB|
00001120  43 44 45 46 47 48 49 4a  4b 4c 4d 4e 4f 50 51 52  |CDEFGHIJKLMNOPQR|
00001130  53 54 55 56 57 58 59 5a  41 42 43 44 45 46 47 48  |STUVWXYZABCDEFGH|
00001140  49 4a 4b 4c 4d 4e 4f 50  51 52 53 54 55 56 57 58  |IJKLMNOPQRSTUVWX|
```

STUDENT ID: 51660004