Description of disk manager

Name: Ruiyuan Jiang
Course Number: CS6456
Title: Designing Mini File System

This program has simulated a file system by creating a file on the disk and using that file as a virtual disk. The maximum length of this virtual disk is 64 blocks and each block has 16 bytes. As this file system is very small, I use a simple layout without the inode to manage the disk. It has a super block at the first block of the system. It also holds a File Allocation Table(FAT) and Directory Table (DIR). Also, whenever the file system is mounted, an Open File Table (OFT) is created in the memory of the system. FAT, DIR and super block is the meta data of the file system.

In my layout, block 31-63 are used as the data segmentation and the previous 32 blocks are used for metadata. I stored the FAT in block 3-6, and stored the DIR in block 7-10. The allocation information of these tables is stored in the superblock. All tables are stored as unsigned char array.

My FAT has 32 entries which corresponds to the number of the data block and every entry needs two bytes. The first byte of every entry in FAT represents the state of the block (busy or not) and the second byte stands for the allocation (block index) of the next block needed by the file.

My DIR Table has 8 entries as there are at most 8 files. Every entry needs 8 bytes and 64 bytes in total. DIR Tables records the state, file length, name of the file, first block for the file. I distributed 1 byte to state, two bytes to file length, 4 bytes for the name of file , and one byte to first block for the file.

As for the OFT table, I build a two dimensional array on the memory. It has four entries as there at most 4 open file simultaneously and every entry has 3 attributes: state, file descriptor, offset. File descriptor is similar to the handle in the Linux system. And offset is similar to the file pointer.

This program has 12 sub-functions: make_fs, mount_fs , dismount_fs, fs_open, fs_close, fs_create, fs_delete, fs_read, fs_write, fs_get_filesize, fs_lseek, and fs_truncate.

The first three functions programmed the operation of the file system. Make_fs function creates a file and initialized the content by all 0s.

In the mount_fs function, the metablocks of the virtual disk are read into the memory. And I stored that information on a global char array.

In the dismount_fs function, the metablock is written back to virtual disk by the block_write function in disk.c

The rest nine functions is used to describe the operation on the file on the disk. The routine operation contains opening file, closing file, deleting file, and modifying file. First, we create a new file by calling fs_create, it will update the DIR table, and the FAT table. When this file is opend, fs_open is called and the function will updated the OFT table in the memory. The offset is set to zero, when the file is newly created. The fs_write function will modify the content of the file. And the distribution of the block required by the file follows the FIFO principle. The routine is that at first it will look into the OFT table , find the file descriptor and fetch the allocation of the first block of that file according to the file descriptor. Secondly, it will find the free block for this file

according to the FAT Table. The rest of these functions are very similar, and the most important task of these functions is to maintain and update the metadata of the file system.

Appendix:

```
  #include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include "disk.h"

int make_fs(char *disk_name);
int mount_fs(char *disk_name);
int dismount_fs(char *disk_name);
int fs_open(char *name);
int fs_close(int fildes);
int fs_create(char *name);
int fs_delete(char *name);
int fs_read(int fildes, void *buf, size_t nbyte);
int fs_write(int fildes, void *buf, size_t nbyte);
int fs_get_filesize(int fildes);
int fs_lseek(int fildes, off_t offset);
int fs_truncate(int fildes, off_t length);

size_t nbyte;
off_t offset;
off_t length;

#define BUFFER_SIZE 160
#define BLOCK_SIZE 16

int i;
int fileNum=0;

char metaBuf[BUFFER_SIZE];// supber block ,FAT, DIR

char tempBuf[BUFFER_SIZE];// read buffer


int OFT[4][3]; //1.status 2.fildes 3.offset
// FAT 1.status, 2. subsequent block allocation
// DIR 1.status, 2-3 length, 4-7 name, 8. first block allocation
```

```
int fda, fdb, fdc, fdd, fde; // 5 file descriptors


int make_fs(char *disk_name){
            char buf[BLOCK_SIZE];
      i=0;
      if(make_disk(disk_name)!=0)
      return -1 ;
      if(open_disk(disk_name)==-1)
      return -1;
      for(i=0;i<sizeof(disk_name);i++){
      buf[i]=disk_name[i];
      }
      buf[4]=(unsigned char)3;
      buf[5]=(unsigned char)6;
      buf[6]=(unsigned char)7;
      buf[7]=(unsigned char)10;
      buf[8]=(unsigned char)32;
      buf[9]=(unsigned char)64;
      if(block_write(0, buf)==-1)
      return -1;
      close_disk();
      return 0;
      }

int mount_fs(char *disk_name){
      int i=0;
      int j=0;
      char buf[BLOCK_SIZE];
      if(open_disk(disk_name)==-1){
      printf("mount\n");
      return -1;}
      for (i=0;i<10;i++){

            if (block_read(i, buf)==-1)
            return -1;
            for (j=0;j<16;j++){
               metaBuf[i*16+j]=buf[j];
            }
      // intialize the oft table
      for (i=0;i<4;i++){
            for(j=0;j<3;j++)
            OFT[i][j]=0;
```

```
	}

	return 0;
	}
}

int dismount_fs(char *disk_name){
	int i=0;
	int j=0;
	char buf[BLOCK_SIZE];
	//if(open_disk(disk_name)==-1){
	//printf("dismount\n");
	//return -1;}
	for (i=0;i<10;i++){
		for (j=0;j<16;j++)
		buf[j]=metaBuf[i*16+j];

		if (block_write(i, buf)==-1)
		return -1;
	}

	close_disk();
	return 0;
}

int fs_open(char *name){
	int namelen=0;
	int nameptr=0;
	int offset=0;
	char string[4]; // file name
	int fildes=0;
	int i;
	int m=0;
		int found=0;
	// judge valid name or not
	namelen=strlen(name);
	if (namelen>4)
	return -1;
	// judge OFT full or not
	for(i=0;i<4;i++){
		if (OFT[i][0]==1)
				m++;

		}
```

```
    if (m>=4){
    printf("full OFT\n");
    return -1;
    }
    // judge whether the file is on the disk
    for (nameptr=6*16+3;nameptr<16*10;nameptr=nameptr+8){
                fildes++;
        for (offset=0;offset<4;offset++)
            string[offset]=metaBuf[nameptr+offset];
        if (strcmp(string,name)==0){
            found=1;
            break;
        }

    }
    if (found!=1){
             printf("no such file\n");
    return -1;
        }
    // judge whether the file has been opend
    for(i=0;i<4;i++){
        if (OFT[i][1]==fildes){
            printf("Has been opened");
                        return -1;
            }
    }
    // open the file write the oft
    for(i=0;i<4;i++){
        if (OFT[i][0]==0){
            OFT[i][0]=1;
            OFT[i][1]=fildes;
                        OFT[i][2]=0;
            break;
            }
    }
    return fildes;

}



int fs_close(int fildes){
    int namelen=0;
    int offset=0;
```

```
    int i;
        int found=0;

    // judge whether the file has been opend
    for(i=0;i<4;i++){
        if (OFT[i][1]==fildes){
            found=1;
            OFT[i][0]=0;
            OFT[i][1]=0;
                    OFT[i][2]=0;
              break;
            }
    }
    if (found!=1){
            printf("no such opened file4\n");
    return -1;
    }
    return 0;
}

int fs_create(char *name){
    int namelen=0;
    int nameptr=0;
    int offset=0;
    char *string; // file name
    string = (char *) malloc(4);
    int fatptr;

    namelen=strlen(name);
    if (namelen>4){
      printf("Invalid file name\n");
    return -1;}
    if (fileNum>=8){
      printf("Can't create more than 8 files\n");
    return -1;}
    // judge dup
    for (nameptr=6*16+3;nameptr<16*10;nameptr=nameptr+8){

        for (offset=0;offset<4;offset++)
            string[offset]=metaBuf[nameptr+offset];
        if (strcmp(string,name)==0){
          printf("Duplicate files\n");
          return -1;
```

```
        }

    }

    // judge FAT
    for(fatptr=16*2;fatptr<16*6;fatptr=fatptr+2){
        if((int)metaBuf[fatptr]==0){
            metaBuf[fatptr]=(unsigned char)1;
            break;
        }
    }
    if (fatptr>16*6){
    printf("Full disk\n");
    return -1;
    }
    // write DIR
    fileNum++;
    for (nameptr=6*16+3;nameptr<16*10;nameptr=nameptr+8){


            if((int)metaBuf[nameptr-3]==0){
                    metaBuf[nameptr-3]=(unsigned char)1;
                    metaBuf[nameptr-2]=(unsigned char)0;
                metaBuf[nameptr-1]=(unsigned char)0;
                for (offset=0;offset<namelen;offset++)
                    metaBuf[nameptr+offset]=name[offset];
                metaBuf[nameptr+4]=(unsigned char)((fatptr-32)/2+32);
                break;
            }

        }

    return 0;
}

int fs_delete(char *name){
    int namelen=0;
    int nameptr=0;
    int offset=0;
    int fildes=0;
    char string[4]; // file name
    int i;
        int found=0;
    int delptr=0;
```

```
int fatptr=0;
        char *tempbuffer;
tempbuffer=(char *) malloc(BLOCK_SIZE);
int h=1;//length of the block series
int m=0;// iteration
int blockloc[10];
int linkptr=0;
// judge valid name or not
namelen=strlen(name);
if (namelen>4)
return -1;
// judge whether the file is on the disk
for (nameptr=6*16+3;nameptr<16*10;nameptr=nameptr+8){
              fildes++;
      for (offset=0;offset<4;offset++)
            string[offset]=metaBuf[nameptr+offset];
      if (strcmp(string,name)==0){
            found=1;
            delptr=nameptr-3;
            break;
      }

}
if (found!=1){
             printf("no such file\n");
return -1;
      }
// judge whether the file has been opend
for(i=0;i<4;i++){
      if (OFT[i][1]==fildes){
           printf("Has been opened cannot be deleted");
                        return -1;
            }
}
// delete the file;

// delete the FAT and the data block;
// initialized the tempbuffer
tempbuffer="";
// first block location
  blockloc[0]=(int)metaBuf[delptr+7];
      // find all the blocks
linkptr=(blockloc[0]-32)*2+32+1;
while((int)metaBuf[linkptr]!=0){
```

```
            blockloc[h]=(int)metaBuf[linkptr];
            linkptr=(blockloc[h]-32)*2+32+1;

            h=h+1;
            }
      // clean the file 's data block
      for(m=0;m<=h-1;m++){
            block_write(blockloc[m],tempbuffer);
            }
      // update FAT
             for(m=0;m<h;m++){
               linkptr=(blockloc[m]-32)*2+32+1;
               metaBuf[linkptr]=(unsigned char)0;
               metaBuf[linkptr-1]=(unsigned char)0;
              //        // delete the dir;
        fileNum--;
      for (i=delptr;i<delptr+8;i++){
            metaBuf[i]=(unsigned char)0;
            }
            }
}

int fs_write(int fildes, void *buf, size_t nbyte){
      char buffer[BLOCK_SIZE];
      int i=1;
      int foundfile=0;
      int foundspace=0;
      int j=0;
      int m=0;
      int n=0;// length base
      int l=0; //length offset
      int baseptr=6*16+(fildes-1)*8;
      int blen=0;
      int fatptr=0;
      int linkptr=0;
      int off_set=0;



      // judge the length for link list
      if (nbyte%16==0)
            blen=nbyte/BLOCK_SIZE;
              else
            blen=nbyte/BLOCK_SIZE+1;
```

```
int blockloc[blen];
        // initialize blockloc
for(j=0;j<blen;j++){
  blockloc[j]=0;
}

// judge whether the file has been opend
if (fildes==0){
printf("Invalid file name\n");
return -1;
}
for(j=0;j<4;j++){
      if (OFT[j][1]==fildes){
            off_set=OFT[j][2];
            foundfile=1;
              break;
            }
}
if (foundfile!=1){
            printf("No such opened file1\n");
return -1;
}

m= nbyte/BLOCK_SIZE;    // Num of blocks for current write
n= (nbyte+off_set)/256; // length base
l= (nbyte+off_set)%256; //length offset
// update the file length
metaBuf[baseptr+1]=(unsigned char)n;
metaBuf[baseptr+2]=(unsigned char)l;
 // judge what block to write
   if (off_set==0){
// first block location
blockloc[0]=(int)metaBuf[baseptr+7];
  if (i<=m){
// find all the block series needed and update the FAT entry
  for(fatptr=16*2;fatptr<16*6;fatptr=fatptr+2){

      if((int)metaBuf[fatptr]==0){
            metaBuf[fatptr]=(unsigned char)1;
            blockloc[i]=((fatptr-32)/2+32);
            linkptr=(blockloc[i-1]-32)*2+32+1;
            metaBuf[linkptr]=(unsigned char)blockloc[i];
            i=i+1;
            foundspace=1;
```

```
                //printf("%d \n",i);
                        if (i>m){
                //printf("%d,%d\n",blockloc[0],blockloc[1]);
                break;}
            }
        }

        // no any space for write
        if (foundspace==0){
        printf("Full disk\n");
        return 0;
        }
        }
        // write the file as long as possible
        for(i=0;i<blen;i++){
            strncpy(buffer,buf+16*i,16);
            if (blockloc[i]==0){
            printf("No enough space for full file, the actual write length is %d byte\n",
i*16 );
            return ((i-1)*16);
              }
            block_write(blockloc[i],buffer);

            }

        // find the file in OFT and update the offset
        for(j=0;j<4;j++){
            if (OFT[j][1]==fildes){
                  OFT[j][2]=nbyte;
                    break;
                }
            }
         }
       else{
        // iterate the FAT for block
                    int lastOffset=off_set%16;
            int lastBlockNum=off_set/16;
            int currFAT=(int)metaBuf[baseptr+7];

            for(j=0;j<lastBlockNum;j++){
                int fat_ptr=currFAT;
                currFAT=(metaBuf[currFAT+1]-32)*2+32;

            }
```

```
        // clean FAT//
        int dummyCurrFAT=currFAT;

        if(metaBuf[dummyCurrFAT+1]!=0){
              dummyCurrFAT=toFAT(metaBuf[dummyCurrFAT+1]);
              metaBuf[dummyCurrFAT]=0;

        }
        while(metaBuf[dummyCurrFAT+1]!=0){
                    metaBuf[dummyCurrFAT]=0;
              dummyCurrFAT=toFAT(metaBuf[dummyCurrFAT+1]);
        }
        // truncate block
        char *buffer1 = (char *) malloc(16);
        char *buffercut = (char *) malloc(16);
        block_read((currFAT-32)/2+32,buffer1);
        printf("%d\n",(currFAT-32)/2+32);
        strncpy(buffercut,buffer1,lastOffset);

        strcat(buffercut,buf);
        printf(buffercut);
        printf("\n");
    // first block location
    blockloc[0]=(currFAT-32)/2+32;

      if (i<=m){
    // find all the block series needed and update the FAT entry
      for(fatptr=16*2;fatptr<16*6;fatptr=fatptr+2){

        if((int)metaBuf[fatptr]==0){
              metaBuf[fatptr]=(unsigned char)1;
              blockloc[i]=((fatptr-32)/2+32);
              linkptr=(blockloc[i-1]-32)*2+32+1;
              metaBuf[linkptr]=(unsigned char)blockloc[i];
              i=i+1;
              foundspace=1;
              //printf("%d \n",i);
                    if (i>m){
              //printf("%d,%d\n",blockloc[0],blockloc[1]);
              break;}
             }
              }
    // no any space for write
```

```
      if (foundspace==0){
      printf("Full disk\n");
      return 0;
       }
         }
      // write the file as long as possible

      for(i=0;i<blen;i++){
            strncpy(buffer,buffercut+16*i,16);
            if (blockloc[i]==0){
            printf("No enough space for full file, the actual write length is %d byte\n",
i*16 );
            return ((i-1)*16);
              }
            block_write(blockloc[i],buffer);

            }

      // find the file in OFT and update the offset
      for(j=0;j<4;j++){
            if (OFT[j][1]==fildes){
                  OFT[j][2]=nbyte;
                    break;
                  }
            }



      }
      return nbyte;
}

int fs_read(int fildes, void *buf, size_t nbyte){
      char *buffer;
      buffer=(char *) malloc(BUFFER_SIZE);
      char *tempbuffer;
      tempbuffer=(char *) malloc(BLOCK_SIZE);
      int i=1;//length of the block series
      int j=0;
      int foundfile=0;
      int baseptr=6*16+(fildes-1)*8;
      int blockloc[10];
      int flength=0;
      int linkptr=0;
```

```
int off_set=0;

// judge whether the file has been opend
if (fildes==0){
printf("Invalid file name\n");
return -1;
}
for(j=0;j<4;j++){
      if (OFT[j][1]==fildes){
            foundfile=1;
            off_set=OFT[j][2];
              break;
            }
}
if (foundfile!=1){
            printf("No such opened file\n");
return -1;
}
// judge the filepointer
flength=fs_get_filesize(fildes);
if (off_set==flength){
printf("At the end of file\n");
return 0;
}
// update OFT offset
if ((off_set+nbyte)<=flength){
     OFT[j][2]=off_set+nbyte;}
else
     OFT[j][2]=flength;
// read the whole file out;
      // first block location
blockloc[0]=(int)metaBuf[baseptr+7];
//printf("%d\n",i);
      // find all the blocks
linkptr=(blockloc[i-1]-32)*2+32+1;
while((int)metaBuf[linkptr]!=0){
      blockloc[i]=(int)metaBuf[linkptr];
      linkptr=(blockloc[i]-32)*2+32+1;
      i=i+1;
      }
      // read the file in buffer
for(j=0;j<=i-1;j++){
      block_read(blockloc[j],tempbuffer);
      strncpy(buffer+16*j,tempbuffer,16);
```

```
        }
        // using offset and nbyte to write the tempBuf
        strncpy(tempBuf,buffer+off_set,nbyte);


}

int fs_get_filesize(int fildes){
    int baseptr=6*16+(fildes-1)*8;
    int flength;

        flength=(int)metaBuf[baseptr+1]*256+(int)metaBuf[baseptr+2];
    return flength;

}

int fs_lseek(int fildes, off_t offset){
    int foundfile=0;
    int flength=0;

        flength=fs_get_filesize(fildes);
    // judge whether the file has been opend
    if (fildes==0){
    printf("Invalid file name\n");
    return -1;
    }
    for(i=0;i<4;i++){
        if (OFT[i][1]==fildes){
            foundfile=1;
              break;
            }
    }
    if (foundfile!=1){
            printf("Invalid file name\n");
    return -1;}
        // judge whether the file is out of bound
    if(OFT[i][2]+offset<0||OFT[i][2]+offset>flength){
      printf("File pointer out of bound\n");
          return -1;
    }
    OFT[i][2]=OFT[i][2]+offset;
    return 0;
    }
```

```
int fs_truncate(int fildes, off_t length){
    char *tempbuffer;
    tempbuffer=(char *) malloc(BLOCK_SIZE);
    int i=1;//length of the block series
    int j=0;// use to track the OFT
    int m=0;// iteration
    int n=0;// length base
    int l=0;// length offset
    int foundfile=0;
    int cutBlock=length/16;
    int baseptr=6*16+(fildes-1)*8;
    int blockloc[10];
    int flength=0;
    int linkptr=0;
    int off_set=0;

    // initialized the tempbuffer

            tempbuffer="";

    // judge whether the file has been opend
    if (fildes==0){
    printf("Invalid file name\n");
    return -1;
    }
    for(j=0;j<4;j++){
        if (OFT[j][1]==fildes){
            foundfile=1;
            off_set=OFT[j][2];
            OFT[j][2]=0;//after the trucate the offset should be zero
              break;
            }
    }
    if (foundfile!=1){
            printf("no such opened file3\n");
    return -1;
    }
    // judge the length of the file
    flength=fs_get_filesize(fildes);
    if(length>flength){
        printf("cannot extend file by this command\n");
        return -1;
    }
    // read the truncated length message into a buffer
```

```
        fs_read(fildes, tempBuf, length);

        // first block location
    blockloc[0]=(int)metaBuf[baseptr+7];
    //printf("%d\n",i);
        // find all the blocks
    linkptr=(blockloc[i-1]-32)*2+32+1;
    while((int)metaBuf[linkptr]!=0){
        blockloc[i]=(int)metaBuf[linkptr];
        linkptr=(blockloc[i]-32)*2+32+1;

        i=i+1;
        }
    //printf("%d\n",i);
    // clean the file 's data block
    for(m=0;m<=i-1;m++){
        block_write(blockloc[m],tempbuffer);
        }
    // write back the truncate message
    for(m=0;m<=i-1;m++){
        block_write(blockloc[m],tempBuf+16*m);
        }
    // update OFT
    OFT[j][2]=0;
    // update DIR; update the file length
    n= length/256; // length base
    l= length%256; //length offset
    metaBuf[baseptr+1]=(unsigned char)n;
    metaBuf[baseptr+2]=(unsigned char)l;
    // update FAT
         for(m=cutBlock;m<i;m++){
           linkptr=(blockloc[m]-32)*2+32+1;
           metaBuf[linkptr]=(unsigned char)0;
           if(m>cutBlock){
           metaBuf[linkptr-1]=(unsigned char)0;}
        }

}

int toBlock(int fat){
    int block=(fat-32)/2+32;
    return block;
    }
```

```
int toFAT(int block){
	int fat=(block-32)*2+32;
	return fat;
}
```