CS232L Operating Systems Lab
Assignment 04 : A simply file system

Name: Muhammad Munawwar Anwar
ID: ma04289

December 8, 2020

1 filesystem.c

1.1 my_list.c

```c
#include<stdio.h>
#include<string.h>
#include<unistd.h>
#include<fcntl.h>
#include<stdlib.h>
#include <stdbool.h>
/*
 *   --- --- --- --- --- --- --- --- --- --- ---
 *  |   |   |   |   |                       |   |
 *  | 0 | 1 | 2 | 3 |      .....            |127|
 *  |---|---|---|---|-----------------------|---|
 *  |   \    <-----   data  blocks  ------->
 *  |     \
 *  |       \
 *  |         \
 *  |           \
 *  |             \
 *  |               \
 *  |                 \
 *  |                   \
 *  |                     \
 *  |                       \
 *  |                         \
 *  |                           \
 *  |                             \
 *  |                               \
 *  |                                 \
 *  |                                   \
 *  |                                     \
 *  |                                       \
 *  |                                         \
 *  |                                           \
 *  |     <--- super block --->                   \
 *  |----------------------------------------------|
 *  |               |      |      |        |       |
 *  |        free   |      |      |        |       |
 *  |       block   |inode0|inode1|   .... |inode15|
 *  |        list   |      |      |        |       |
 *  |---------------|------|------|--------|-------|
 *
 *
 */


#define FILENAME_MAXLEN 8  // including the NULL char
#define BLOCK_SIZE 1024
#define MAX_BLOCK  128
```

```c
int myfs;

/*
 * inode
 */

typedef struct inode {
  int  dir;  // boolean value. 1 if it's a directory.
  char name[FILENAME_MAXLEN];
  int  size;  // actual file/directory size in bytes.
  int  blockptrs [8];  // direct pointers to blocks containing file's content.
  int  used;  // boolean value. 1 if the entry is in use.
  int  rsvd;  // reserved for future use
} inode;


/*
 * directory entry
 */

typedef struct dirent {
  char name[FILENAME_MAXLEN];
  int  namelen;  // length of entry name
  int  inode;  // this entry inode index
} dirent;

/*
 * Data Block
 */

typedef struct DataBlock {
    char Data[1024];
}block;

/*
 * Directory Block
 */

typedef struct DirectoryBlock{
    struct dirent DirectoryTable[17];
    // One for Parent Directory  0th Position --> Empty in case of Root Directory
    // One for Current Directory 1th Position
    // 15 Directories at Max (2-16)th Position

} DirectoryBlock;


/*
 * functions
 */
// create file
// copy file
// remove/delete file
// move a file
// list file info
// create directory
// remove a directory


/*
 * Initialize File System
 *
 */

int initiliaze()
{
  myfs = open("myfs", O_CREAT | O_RDWR,0222);
  bool FreeBlockList[128]; // Free Block List
  struct inode InodeTable [16]; // Inode Table
  char Data [BLOCK_SIZE];
  int i = 0;

  for (i = 0; i<128;i++) // Free Block Initialize
  {
      FreeBlockList[i] = false;
  }
  for (i = 0; i<16;i++)  // Inode Table Initialize
  {

      strcpy(InodeTable[i].name,"");
      InodeTable[i].used = 0;
      InodeTable[i].rsvd = 0;
      InodeTable[i].size = 0;
      InodeTable[i].dir = 0;
      for (int j = 0; j < 8;j++){
          InodeTable[i].blockptrs[j] = -1;
      }
  }

  // Root Inode Initialize
  FreeBlockList[0] = true;
  FreeBlockList[1] = true;
  strcpy(InodeTable[0].name,"/");
  InodeTable[0].size = sizeof(DirectoryBlock);
  InodeTable[0].dir = 1;
  InodeTable[0].used = 1;
  InodeTable[0].blockptrs[0] = 1;

  // Root Directory Block Initialize
  struct DirectoryBlock RootBlock;
  strcpy(RootBlock.DirectoryTable[0].name,"NA");
  RootBlock.DirectoryTable[0].namelen = 2;
  RootBlock.DirectoryTable[0].inode = -2;
  strcpy(RootBlock.DirectoryTable[1].name,".");
  RootBlock.DirectoryTable[1].namelen = 1;
  RootBlock.DirectoryTable[1].inode = 0;
  for (int i = 2; i < 17;i++)
  {
      RootBlock.DirectoryTable[i].inode = -1;
      RootBlock.DirectoryTable[i].namelen = 0;
      strcpy(RootBlock.DirectoryTable[i].name,"");

  }

  write(myfs, (char*)&FreeBlockList, 128);
  write(myfs,(char*)&InodeTable,16*56);
  write(myfs,(char*)&RootBlock,BLOCK_SIZE);

  // Initialize the Data Region
  for ( i  = 1 ;  i < 127; i++){
      write(myfs,(char*)&Data,BLOCK_SIZE);
  }
  return myfs;
}

int getFreeInode(int flag, char *name)
{
  struct inode ReadTable[16];
  int i = 1;
  lseek(myfs,128,SEEK_SET);
  read(myfs,(char*)&ReadTable,16*56);

  if (flag == 1)
  {
    for (i = 1; i <16;i++)
    {
      if (strcmp(ReadTable[i].name,name) == 0)
      {
```

```c
        return -1;
      }
    }
  }
  for (i = 1; i <16;i++)
  {
    if (ReadTable[i].used == 0)
    {
      return i;
    }
  }

  return -2;
}
// Scan the inode table. Return the corresponding inode. If inode is not found, then return -1
int getInode(char*name){
  struct inode ReadTable[16];
  lseek(myfs,128,SEEK_SET);
  read(myfs,(char*)&ReadTable,16*56);
  int i = 0;
  for (i = 0 ; i < 16; i++ )
  {
    if (strcmp(ReadTable[i].name,name) == 0 && ReadTable[i].used == 1){
      return i;
    }
  }
  return -1;
}
// Scan the bit map and return the first free block
int getFreeBlock()
{
  lseek(myfs,0,SEEK_SET);
  bool ReadFreeBlockList[128];
  read(myfs,(char*)ReadFreeBlockList,128);
  int i = 0;
  for (i = 0 ; i < 128; i++)
  {
    if (ReadFreeBlockList[i] == false)
    {
      return i;
    }
  }
  return -1;
}

// CREATE A FILE

/*
  Start from the Root Directory
  Traverse Path while keeping directory inode.
  Check if the directory in the file path exsists
  Else return -1
  If inode available
  Check if the file does not exsist
  Write in directory
  Create directory entry

*/
int CR(char* filename, int size)
{
  if (size > 1024 * 8)
  {
    printf("error: File size exceeding maximum limit.\n");
    return -1;
  }
  if (strcmp(filename,"/") == 0 )
  {
    printf("error: Root directory is an invalid filename.\n");
    return -1;
  }
  char * token = strtok(filename,"/");
  char * ChildDirectory = token;
  char * ParentDirectory = "/";

  // Get Parent Inode & Parent Directory
  int address = getInode(ParentDirectory);
  struct inode ParentInode;
  struct DirectoryBlock ParentDirectoryBlock;

  int i = 0;
  lseek(myfs,(128+(address*56)),SEEK_SET);
  read(myfs,(char*)&ParentInode,56);
  int value = ParentInode.blockptrs[0];
  lseek(myfs,value*1024,SEEK_SET);
  read(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));
  while(token!=NULL)
  {
    token = strtok(NULL,"/");
    if (token !=NULL)
    {
      // Scan the child inode's entry in the Parent's directory table. If not found then return -1.
      // If found then the child is the new parent.
      int flag = 0;
      for (i = 2; i < 17;i++)
      {
        if(strcmp(ParentDirectoryBlock.DirectoryTable[i].name,ChildDirectory) == 0){
        address = ParentDirectoryBlock.DirectoryTable[i].inode;
        flag = 1;
        break;
        }
      }
      if (flag == 0)
      {
        printf("error: The %s directory in the given path does not exsist \n",ChildDirectory);
        return -1;
      }
      // Get the new parent's inode and directory
      lseek(myfs,(128+(address*56)),SEEK_SET);
      read(myfs,(char*)&ParentInode,56);
      value = ParentInode.blockptrs[0];
      lseek(myfs,value*1024,SEEK_SET);
      read(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));
      ParentDirectory = ChildDirectory;
      ChildDirectory = token;

    }
  }
  // Check if free inodes our availaible
  int freeInoode = getFreeInode(0,ChildDirectory);
  if (freeInoode == -2)
  {
    printf("error: All inodes our occupied\n");
    return -1;
  }
  // Check if the file exsists already
  for(int i = 2; i  < 17;i++)
  {
    if (ParentDirectoryBlock.DirectoryTable[i].inode != - 1 && strcmp(ParentDirectoryBlock.DirectoryTable[i].name,ChildDirectory) == 0 ){

      printf("error: The file %s already exsists .\n",ChildDirectory);
      return -1;

    }
```

```c
}
// Get Free Data Blocks & Fill them according to the size specified
struct inode ChildNode;
struct DataBlock ChildData[8];
int DataPtr[8];
for (int i = 0 ; i < 8 ; i ++)
{
  DataPtr[i] = -1;
}
char Buffer[1024];
char NewBuffer[1024];
int fullblocks;
int partialblock;
fullblocks = (size)/1024;
partialblock = (size%1024);
if (size > 1024)
{
  if(partialblock != 0)
  {
    fullblocks ++;
  }
}
int j =  0 ;
bool lflag = true;
for (i = 0 ; i < fullblocks; i++)
{
  DataPtr[i] = getFreeBlock();
  lseek(myfs,DataPtr[i],SEEK_SET);
  write(myfs,(char*)&lflag ,1);
  for (j = 0 ; j < 1024; j++)
  {
    Buffer[j] = (char) (97 + j % 26);
  }
  strcpy(ChildData[i].Data,Buffer);
}

if (partialblock >0)
{
  DataPtr[fullblocks] = getFreeBlock();
  lseek(myfs,DataPtr[fullblocks],SEEK_SET);
  write(myfs,(char*)&lflag ,1);
  for (i = 0; i < partialblock; i++)
  {
    NewBuffer[i] = (char) (97 + i % 26);
  }
  strcpy(ChildData[fullblocks].Data,NewBuffer);
}
// Create File's entry in the inode table
lseek(myfs,128 + (56*freeInoode),SEEK_SET);
read(myfs,(char*)&ChildNode,sizeof(inode));
ChildNode.dir = 0;
strcpy(ChildNode.name,ChildDirectory);
ChildNode.size = size;
ChildNode.used = 1;
ChildNode.rsvd = 0;
// Update's parents directory
for (int i = 0 ; i < 8; i++)
{
  ChildNode.blockptrs[i] = DataPtr[i];
}
for ( i = 2; i < 17; i++)
{
  if (ParentDirectoryBlock.DirectoryTable[i].inode == -1)
  {
    ParentDirectoryBlock.DirectoryTable[i].inode = freeInoode;
    ParentDirectoryBlock.DirectoryTable[i].namelen = strlen(ChildDirectory);
    strcpy(ParentDirectoryBlock.DirectoryTable[i].name,ChildDirectory);
    break;
  }
  }
  lseek(myfs,128 + (56*freeInoode),SEEK_SET);
  write(myfs,(char*)&ChildNode,sizeof(inode));
  lseek(myfs,value*1024,SEEK_SET);
  write(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));
  for (i = 0 ; i < 8;i++)
  {
    if (ChildNode.blockptrs[i] != -1)
    {
      lseek(myfs,1024 * ChildNode.blockptrs[i],SEEK_SET);
      write(myfs,(char*)&ChildData[i],1024);
    }
  }
  return 0;
}

// DELETE A FILE

/*
  Start from the Root Directory
  Traverse Path while keeping directory inode.
  Check if the directory in the file path exsists
  Else return -1
  Check if the file does exsist
  Delete directory entry
*/
int DL(char* filename)
{

  if (strcmp(filename,"/") == 0 )
  {
    printf("error: Cannot delete root directory \n");
    return -1;
  }
  char * token = strtok(filename,"/");
  char * ChildDirectory = token;
  char * ParentDirectory = "/";
  // Get Parent Inode & Parent Directory
  int address = getInode(ParentDirectory);
  struct inode ParentInode;
  struct DirectoryBlock ParentDirectoryBlock;

  int i = 0;

  lseek(myfs,(128+(address*56)),SEEK_SET);
  read(myfs,(char*)&ParentInode,56);
  int value = ParentInode.blockptrs[0];
  lseek(myfs,value*1024,SEEK_SET);
  read(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));
  while(token!=NULL)
  {
    token = strtok(NULL,"/");
    if (token !=NULL)
    {// Scan the child inode's entry in the Parent's directory table. If not
    found then return -1.
      // If found then the child is the new parent.
      int flag = 0;
      for (i = 2; i < 17;i++)
      {
        if(strcmp(ParentDirectoryBlock.DirectoryTable[i].name,ChildDirectory) ==
    0){
        address = ParentDirectoryBlock.DirectoryTable[i].inode;
        flag = 1;
        break;
        }
      }
      if (flag == 0)
      {
        printf("error: The directory %s in the given path does not exsist \n",
    ChildDirectory);
        return -1;
      }
      // Get the new parent's inode and directory
      lseek(myfs,(128+(address*56)),SEEK_SET);
      read(myfs,(char*)&ParentInode,56);
      value = ParentInode.blockptrs[0];
      lseek(myfs,value*1024,SEEK_SET);
      read(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));
      ParentDirectory = ChildDirectory;
      ChildDirectory = token;
    }
  }  // Check if the file exsists already
  int DeleteInode = -1;
  for (i = 2; i < 17;i++)
  {
      if(strcmp(ParentDirectoryBlock.DirectoryTable[i].name,ChildDirectory) == 0)
      {
        DeleteInode = ParentDirectoryBlock.DirectoryTable[i].inode;
        ParentDirectoryBlock.DirectoryTable[i].inode = -1;
        ParentDirectoryBlock.DirectoryTable[i].namelen = 0;
        strcpy(ParentDirectoryBlock.DirectoryTable[i].name,"");
        break;

      }
  }
  if (DeleteInode == -1)
  {
    printf("error: The file %s does not exsist.\n",ChildDirectory);
    return -1;
  }
  // Release all the blocks pointed by the file
  struct inode DelInode;
  lseek(myfs,128 + (56 * DeleteInode),SEEK_SET);
  read(myfs,(char*)&DelInode,sizeof(inode));

  bool lflag = false;
    for (i = 0 ; i < 8 ; i++)
    {
      if (DelInode.blockptrs[i]!=-1)
      {
        DelInode.blockptrs[i] = -1;
        lseek(myfs,DelInode.blockptrs[i],SEEK_SET);
        write(myfs,(char*)&lflag,1);
      }
    }
    // Release the inode
    strcpy(DelInode.name,"");
    DelInode.used = 0;
    DelInode.rsvd = 0;
    DelInode.size = 0;
    DelInode.dir = 0;

  lseek(myfs,128 + (56 * DeleteInode),SEEK_SET);
  write(myfs,(char*)&DelInode,sizeof(inode));

  lseek(myfs,value*1024,SEEK_SET);
  write(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));
  return 0;
}
/*
  Start from the Root Directory
  Traverse both paths while keeping track of directory inode.
  Check if the directory in the file path exsists
  Else return -1
  Check if the source file exsists in the destination path
  If exsists then overwrite contents
  Copy contents in directory
  Create directory entry
```

```
*/
int CP(char* srcname, char * dstname )
{
  if(strcmp(srcname,"/")==0)
  {
    printf("error: The Root Directory is an invalid source \n");
    return -1;
  }
  if(strcmp(dstname,"/")==0)
  {
    printf("error: The Root Directory is an invalid source \n");
    return -1;
  }
  char * token = strtok(srcname,"/");
  char * srcChildDirectory = token;
  char * srcParentDirectory = "/";
 // Get Source Parent Inode & Parent Directory
  int srcaddress = getInode(srcParentDirectory);

  struct inode srcParentInode;
  struct DirectoryBlock srcParentDirectoryBlock;
  lseek(myfs,(128+(srcaddress*56)),SEEK_SET);
  read(myfs,(char*)&srcParentInode,56);
  int i = 0;
  int value = srcParentInode.blockptrs[0];
  lseek(myfs,value*1024,SEEK_SET);
  read(myfs,(char*)&srcParentDirectoryBlock,sizeof(DirectoryBlock));
  while(token!=NULL)
  {
    token = strtok(NULL,"/");
    if (token !=NULL)
    {
      // Scan the child inode's entry in the Parent's directory table. If not
    found then return -1.
      // If found then the child is the new parent.
      int flag = 0;
      for (i = 2; i < 17;i++)
      {
        if(strcmp(srcParentDirectoryBlock.DirectoryTable[i].name,
    srcChildDirectory) == 0){
        srcaddress = srcParentDirectoryBlock.DirectoryTable[i].inode;
        flag = 1;
        break;
        }
      }
      if (flag == 0)
      {
        printf("error: The %s directory in the given path does not exsist \n",
    srcChildDirectory);
        return -1;
      }
       // Get the new parent's inode and directory
    lseek(myfs,(128+(srcaddress*56)),SEEK_SET);
    read(myfs,(char*)&srcParentInode,56);
    value = srcParentInode.blockptrs[0];
    lseek(myfs,value*1024,SEEK_SET);
    read(myfs,(char*)&srcParentDirectoryBlock,sizeof(DirectoryBlock));
    srcParentDirectory = srcChildDirectory;
    srcChildDirectory = token;
    }
  }
  token = strtok(dstname,"/");
  char * ChildDirectory = token;
  char * ParentDirectory = "/";
  // Destination Parent Inode & Parent Directory
  int address = getInode(ParentDirectory);

  struct inode ParentInode;
  struct DirectoryBlock ParentDirectoryBlock;
```

```

lseek(myfs,(128+(address*56)),SEEK_SET);
read(myfs,(char*)&ParentInode,56);
i = 0;
int value1 = ParentInode.blockptrs[0];
lseek(myfs,value1*1024,SEEK_SET);
read(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));

while(token!=NULL)
{
  token = strtok(NULL,"/");
  if (token !=NULL)
  {
    // Scan the child inode's entry in the Parent's directory table. If not
  found then return -1.
    // If found then the child is the new parent.
    int flag = 0;
    for (i = 2; i < 17;i++)
    {
      if(strcmp(ParentDirectoryBlock.DirectoryTable[i].name,ChildDirectory) ==
  0){
      address = ParentDirectoryBlock.DirectoryTable[i].inode;
      flag = 1;
      break;
      }
    }
    if (flag == 0)
    {
      printf("error: The %s directory in the given path does not exsist \n",
  ChildDirectory);
      return -1;
    }
   // Get the new parent's inode and directory
  lseek(myfs,(128+(address*56)),SEEK_SET);
  read(myfs,(char*)&ParentInode,56);
  value1 = ParentInode.blockptrs[0];
  lseek(myfs,value1*1024,SEEK_SET);
  read(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));
  ParentDirectory = ChildDirectory;
  ChildDirectory = token;

  }
}
struct inode SrcChildNode;
struct inode DstChildNode;
struct DataBlock SrcChildData[8];
struct DataBlock DstChildData[8];

srcaddress = getInode(srcChildDirectory);
lseek(myfs,128 + srcaddress * 56,SEEK_SET);
read(myfs,(char*)&SrcChildNode,sizeof(inode));
// Check if the src path is a directory
if (SrcChildNode.dir == 1)
{
  printf("error: cannot handle directories .\n");
  return -1;
}

// Check if the file already exsists in the dst directory
bool alreadypresent = false;
bool lflag  = true;
int foundat = -1;
int FreeInode;
for ( i = 2; i < 17; i++)
{
  if (ParentDirectoryBlock.DirectoryTable[i].inode != -1 && strcmp(
  ParentDirectoryBlock.DirectoryTable[i].name,ChildDirectory) == 0)
  {
    foundat = i;
```

```
      alreadypresent = true;
    }
  }

  if (alreadypresent == true)
  {
    // Check if the dst path is not a directory path . If yes then overwrite
    content
    FreeInode = ParentDirectoryBlock.DirectoryTable[foundat].inode;
    lseek(myfs,128 + 56 * FreeInode,SEEK_SET);
    read(myfs,(char*)&DstChildNode,sizeof(inode));

    if (DstChildNode.dir == 1)
    {
      printf("error: cannot handle directories \n");
      return -1;
    }
    bool tempflag = false;
    for (i = 0 ; i < 8 ; i++)
    {
      if (DstChildNode.blockptrs[i]!=-1)
      {
        lseek(myfs,DstChildNode.blockptrs[i],SEEK_SET);
        write(myfs,(char*)&tempflag,1);
      }
    }
  }
  else
  {
    // If not present then look for a free inode
    FreeInode = getFreeInode(0,"");
    if (FreeInode == -2){
      printf("error: All inodes our occupied\n");
      return -1;
    }
    lseek(myfs,128 + 56 * FreeInode,SEEK_SET);
    read(myfs,(char*)&DstChildNode,sizeof(inode));
  }
  // Get Data from the Data region
  for (i = 0 ; i < 8 ; i++)
  {
    if(SrcChildNode.blockptrs[i]!=-1)
    {
      lseek(myfs,SrcChildNode.blockptrs[i]*1024,SEEK_SET);
      read(myfs,(char*)&SrcChildData[i],1024);
    }
  }
    // Update Destination Child Inode
    strcpy(DstChildNode.name,ChildDirectory);
    DstChildNode.used = 1;
    DstChildNode.size = SrcChildNode.size;
    DstChildNode.dir = SrcChildNode.dir;
    for(int i = 0 ; i < 8 ; i++)
    {
      if(SrcChildNode.blockptrs[i]!=-1){
        strcpy(DstChildData[i].Data,SrcChildData[i].Data);
        DstChildNode.blockptrs[i] = getFreeBlock();
        lseek(myfs,DstChildNode.blockptrs[i],SEEK_SET);
        write(myfs,(char*)&lflag,1);
      }
    }
    // Update the entry in the Parent's directory
    if (alreadypresent == false)
    {
      for ( i = 2; i < 17; i++)
      {
        if (ParentDirectoryBlock.DirectoryTable[i].inode == -1)
        {
          ParentDirectoryBlock.DirectoryTable[i].inode = FreeInode;
          ParentDirectoryBlock.DirectoryTable[i].namelen = strlen(ChildDirectory)
    ;
          strcpy(ParentDirectoryBlock.DirectoryTable[i].name,ChildDirectory);
          break;
        }
      }
    }
    // Create new entry in the Parent's directory
    else
    {
      ParentDirectoryBlock.DirectoryTable[foundat].inode = FreeInode;
      ParentDirectoryBlock.DirectoryTable[foundat].namelen = strlen(
    ChildDirectory);
      strcpy(ParentDirectoryBlock.DirectoryTable[foundat].name,ChildDirectory);
    }

    lseek(myfs,128 + (56*FreeInode),SEEK_SET);
    write(myfs,(char*)&DstChildNode,sizeof(inode));
    lseek(myfs,value1*1024,SEEK_SET);
    write(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));
    for (i = 0 ; i < 8; i++)
    {
      if (DstChildNode.blockptrs[i]!=-1)
      {
        lseek(myfs,1024 * DstChildNode.blockptrs[i],SEEK_SET);
      }
    }
    return 0 ;
}
/*
  MOVE A FILE
  Start from the Root Directory
  Traverse both paths while keeping track of directory inode.
  Check if the directory in the file path exsists
  Else return -1
  Check if the source file exsists in the destination path
  If exsists then overwrite contents
  Else Write in directory
  Create directory entry in the destination directory
  Delete directory entry in the source directory
*/
int MV(char* srcname, char  * dstname)
{
  if(strcmp(srcname,"/")==0)
  {
    printf("error: Root directory is an invalid source name");
    return -1;
  }
  if(strcmp(dstname,"/")==0)
  {
    printf("error: Root directory is an invalid destinatio name");
    return -1;
  }
  char * token = strtok(srcname,"/");
  char * srcChildDirectory = token;
  char * srcParentDirectory = "/";
// Get Source Parent Inode & Parent Directory
  int srcaddress = getInode(srcParentDirectory);

  struct inode srcParentInode;
  struct DirectoryBlock srcParentDirectoryBlock;


  lseek(myfs,(128+(srcaddress*56)),SEEK_SET);
  read(myfs,(char*)&srcParentInode,56);
  int i = 0;
  int value = srcParentInode.blockptrs[0];
  lseek(myfs,value*1024,SEEK_SET);
  read(myfs,(char*)&srcParentDirectoryBlock,sizeof(DirectoryBlock));
```

```c
  while(token!=NULL)
  {
    token = strtok(NULL,"/");
    if (token !=NULL)
    { // Scan the child inode's entry in the Parent's directory table. If not found then return -1.
      // If found then the child is the new parent.
      int flag = 0;
      for (i = 2; i < 17;i++)
      {
        if(strcmp(srcParentDirectoryBlock.DirectoryTable[i].name, srcChildDirectory) == 0){
        srcaddress = srcParentDirectoryBlock.DirectoryTable[i].inode;
        flag = 1;
        break;
        }
      }
      if (flag == 0)
      {
        printf("error: The %s directory in the given path does not exsist \n", srcChildDirectory);
        return -1;
      }
       // Get the new parent's inode and directory
    lseek(myfs,(128+(srcaddress*56)),SEEK_SET);
    read(myfs,(char*)&srcParentInode,56);
    value = srcParentInode.blockptrs[0];
    lseek(myfs,value*1024,SEEK_SET);
    read(myfs,(char*)&srcParentDirectoryBlock,sizeof(DirectoryBlock));
    srcParentDirectory = srcChildDirectory;
    srcChildDirectory = token;

    }
  }
  token = strtok(dstname,"/");
  char * ChildDirectory = token;
  char * ParentDirectory = "/";
// Destination Parent Inode & Parent Directory
  int address = getInode(ParentDirectory);

  struct inode ParentInode;
  struct DirectoryBlock ParentDirectoryBlock;
  lseek(myfs,(128+(address*56)),SEEK_SET);
  read(myfs,(char*)&ParentInode,56);
  i = 0;
  int value1 = ParentInode.blockptrs[0];
  lseek(myfs,value1*1024,SEEK_SET);
  read(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));

  while(token!=NULL)
  {
    token = strtok(NULL,"/");
    if (token !=NULL)
    { // Scan the child inode's entry in the Parent's directory table. If not found then return -1.
      // If found then the child is the new parent.
      int flag = 0;
      for (i = 2; i < 17;i++)
      {
        if(strcmp(ParentDirectoryBlock.DirectoryTable[i].name,ChildDirectory) == 0){
        address = ParentDirectoryBlock.DirectoryTable[i].inode;
        flag = 1;
        break;
        }
      }
      if (flag == 0)
      {
        printf("error: The %s directory in the given path does not exsist \n",
    ChildDirectory);
        return -1;
      }
        // Get the new parent's inode and directory
    lseek(myfs,(128+(address*56)),SEEK_SET);
    read(myfs,(char*)&ParentInode,56);

    value1 = ParentInode.blockptrs[0];
    lseek(myfs,value1*1024,SEEK_SET);
    read(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));
    ParentDirectory = ChildDirectory;
    ChildDirectory = token;

    }
  }

  bool isFoundSrc = false;
  int foundatSrc = -1;
  for ( i = 2; i < 17; i++)
  {
    if (srcParentDirectoryBlock.DirectoryTable[i].inode != -1 && strcmp(
    srcParentDirectoryBlock.DirectoryTable[i].name,srcChildDirectory) == 0)
    {
      foundatSrc = i;
      isFoundSrc = true;
      break;
    }
  }
  if (isFoundSrc == false)
  {
    printf("error : file or directory with this name does not exist \n");
    return -1;
  }
  struct inode SrcChildNode;
  srcaddress = srcParentDirectoryBlock.DirectoryTable[foundatSrc].inode;
  lseek(myfs,128 + 56 * srcaddress, SEEK_SET);
  read(myfs,(char*)&SrcChildNode,sizeof(inode));
  // Check if the src path is a directory
  if (SrcChildNode.dir == 1)
  {
    printf("error : does not handle directories \n");
    return -1;
  }
  bool  isFound = false;
  int foundat = -1;
  for ( i = 2; i < 17; i++)
  {
    if (ParentDirectoryBlock.DirectoryTable[i].inode != -1 && strcmp(
    ParentDirectoryBlock.DirectoryTable[i].name,ChildDirectory) == 0)
    {
      foundat = i;
      isFound = true;
      break;
    }
  }
  struct inode DstChildNode;
  if (isFound == true)
  {
    // If already exsists, then overwrite old entry. The inode is same but the
    blk pts our updated
    address = ParentDirectoryBlock.DirectoryTable[foundat].inode;
    lseek(myfs,128 + 56 * address, SEEK_SET);
    read(myfs,(char*)&DstChildNode,sizeof(inode));
    if (DstChildNode.dir == 1)
    {
      printf("error : does not handle directories \n");
      return -1;
    }
    for (i = 0 ; i < 8 ; i++)
    {
      DstChildNode.blockptrs[i] = SrcChildNode.blockptrs[i];
      SrcChildNode.blockptrs[i] = -1;
    }
    DstChildNode.size = SrcChildNode.size;
    SrcChildNode.dir = 0;
    SrcChildNode.rsvd = 0;
    SrcChildNode.size = 0;
    SrcChildNode.used = 0;
    strcpy(SrcChildNode.name,"");

    srcParentDirectoryBlock.DirectoryTable[foundatSrc].inode = -1;
    srcParentDirectoryBlock.DirectoryTable[foundatSrc].namelen = 0;
    strcpy(srcParentDirectoryBlock.DirectoryTable[foundatSrc].name,"");

    lseek(myfs,128 + 56 * srcaddress, SEEK_SET);
    write(myfs,(char*)&SrcChildNode,sizeof(inode));
    lseek(myfs,128 + 56 * address, SEEK_SET);
    write(myfs,(char*)&DstChildNode,sizeof(inode));
    lseek(myfs,value*1024,SEEK_SET);
    write(myfs,(char*)&srcParentDirectoryBlock,sizeof(DirectoryBlock));
  }
  else
  {
    // Delete the entry in src Parent's directory table and add it in the dst
    Parent's directory table
    for (i = 2; i < 17; i++)
    {
      if (ParentDirectoryBlock.DirectoryTable[i].inode == -1 )
      {
        ParentDirectoryBlock.DirectoryTable[i].inode = srcaddress;
        strcpy(ParentDirectoryBlock.DirectoryTable[i].name , ChildDirectory);
        ParentDirectoryBlock.DirectoryTable[i].namelen = strlen(ChildDirectory);
        break;
      }
    }
    srcParentDirectoryBlock.DirectoryTable[foundatSrc].inode = -1;
    srcParentDirectoryBlock.DirectoryTable[foundatSrc].namelen = 0;
    strcpy(srcParentDirectoryBlock.DirectoryTable[foundatSrc].name,"");
    lseek(myfs,value*1024,SEEK_SET);
    write(myfs,(char*)&srcParentDirectoryBlock,sizeof(DirectoryBlock));
    lseek(myfs,value1*1024,SEEK_SET);
    write(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));


  }





  return 0;
}
/*
Start from the Root Directory
  Traverse Path while keeping directory inode.
  Check if the directory in the file path exsists
  Else return -1
  If inode available
  Check if the director does not exsist
  Create an empty directory
  Create directory entry
*/
// CREATE A DIRECTORY
int CD(char* dirname)
{

  if (strcmp(dirname,"/") == 0)
  {
    printf("error : Root directory already exsists \n");
    return -1;
  }
  char * token = strtok(dirname,"/");
  char * ChildDirectory = token;
  char * ParentDirectory = "/";

  int address = getInode(ParentDirectory);
  // Get Parent Inode & Parent Directory
  struct inode ParentInode;
  struct DirectoryBlock ParentDirectoryBlock;
  lseek(myfs,(128+(address*56)),SEEK_SET);
  read(myfs,(char*)&ParentInode,56);
  int i = 0;
  int value = ParentInode.blockptrs[0];
  lseek(myfs,value*1024,SEEK_SET);
  read(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));

  while(token!=NULL)
  {
    token = strtok(NULL,"/");
    if (token !=NULL)
    {// Scan the child inode's entry in the Parent's directory table. If not found then return -1.
      // If found then the child is the new parent.
      int flag = 0;
      for (i = 2; i < 17;i++)
      {
        if(strcmp(ParentDirectoryBlock.DirectoryTable[i].name,ChildDirectory) == 0){
        address = ParentDirectoryBlock.DirectoryTable[i].inode;
        flag = 1;
        break;
        }
      }
      if (flag == 0)
      {
        printf("error: The %s directory in the given path does not exsist \n",ChildDirectory);
        return -1;
      }
       // Get the new parent's inode and directory
    lseek(myfs,(128+(address*56)),SEEK_SET);
    read(myfs,(char*)&ParentInode,56);
    value = ParentInode.blockptrs[0];
    lseek(myfs,value*1024,SEEK_SET);
    read(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));
    ParentDirectory = ChildDirectory;
    ChildDirectory = token;

    }
  }
  struct inode ChildInode;
  struct DirectoryBlock ChildDirectoryBlock;
 // Check if free inodes our availaible  & Check if the file exsists already

  int freeInoode = getFreeInode(1,ChildDirectory);
  if (freeInoode == -1)
  {
    printf("error : file or directory with this name already exsists \n");
    return -1;
  }

  if (freeInoode == -2)
  {
    printf("error: No free Inodes are availaible \n");
    return -1;
  }
  int FreeBlock = getFreeBlock();
```

```

   // Update's parents directory & Create Directory entry in the inode table
  lseek(myfs,128 + (freeInoode * 56),SEEK_SET);
  read(myfs,(char*)&ChildInode,sizeof(inode));
  strcpy(ChildInode.name,ChildDirectory);
  ChildInode.used = 1;
  ChildInode.size = sizeof(DirectoryBlock);
  ChildInode.dir = 1;
  ChildInode.rsvd = 0;
  ChildInode.blockptrs[0] = FreeBlock;
  // Initiliaze the directory table for the new directory
  strcpy(ChildDirectoryBlock.DirectoryTable[0].name , "..");
  ChildDirectoryBlock.DirectoryTable[0].namelen = 2;
  ChildDirectoryBlock.DirectoryTable[0].inode = address;

  strcpy(ChildDirectoryBlock.DirectoryTable[1].name,".");
  ChildDirectoryBlock.DirectoryTable[1].namelen = 1;
  ChildDirectoryBlock.DirectoryTable[1].inode = freeInoode;

  for ( i = 2; i <17;i++)
  {
    ChildDirectoryBlock.DirectoryTable[i].namelen = 0;
    ChildDirectoryBlock.DirectoryTable[i].inode = -1;

  }
  for (int i = 2; i < 17;i++)
  {
    if (ParentDirectoryBlock.DirectoryTable[i].inode == -1)
    {
      strcpy(ParentDirectoryBlock.DirectoryTable[i].name,ChildDirectory);
      ParentDirectoryBlock.DirectoryTable[i].namelen = strlen(ChildDirectory);
      ParentDirectoryBlock.DirectoryTable[i].inode = freeInoode;
      break;
    }
  }

  lseek(myfs,128+(freeInoode*56),SEEK_SET);
  write(myfs,(char*)&ChildInode,sizeof(ChildInode));
  lseek(myfs,1024 * FreeBlock,SEEK_SET);
  write(myfs,(char *)&ChildDirectoryBlock,sizeof(DirectoryBlock));
  lseek(myfs,value*1024,SEEK_SET);
  write(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));
  bool temp = true;
  lseek(myfs,FreeBlock,SEEK_SET);
  write(myfs,(char*)&temp,1);
 return 0;
}

void Recurse(int inodeptr)
{
  struct inode currentnode;
  int i = 0;
  bool Delete = false;
  lseek(myfs,128 + inodeptr * 56,SEEK_SET);
  read(myfs,(char*)&currentnode,sizeof(inode));

  currentnode.used = 0;
  currentnode.rsvd = 0;
  currentnode.size = 0;
  strcpy(currentnode.name,"");

  lseek(myfs,128 + inodeptr * 56,SEEK_SET);
  write(myfs,(char*)&currentnode,sizeof(inode));
// File , So release all blocks
  if (currentnode.dir == 0)
  {

    for (i = 0 ; i < 8; i++)
    {
      if (currentnode.blockptrs[i]!= -1)
      {
        lseek(myfs,currentnode.blockptrs[i],SEEK_SET);
        write(myfs,(char*)&Delete,1);
      }
    }

  }
  else
  {
    // Directory so recursive call
    struct DirectoryBlock currentnodedirectory;
    lseek(myfs,currentnode.blockptrs[0]*1024,SEEK_SET);
    read(myfs,(char*)&currentnodedirectory,sizeof(DirectoryBlock));
    for ( i = 2; i<17;i++)
    {
      if (currentnodedirectory.DirectoryTable[i].inode!=-1)
      {
        Recurse(currentnodedirectory.DirectoryTable[i].inode);
      }
    }
    lseek(myfs,currentnode.blockptrs[0],SEEK_SET);
    write(myfs,(char*)&Delete,1);
  }


}

/*
Start from the Root Directory
  Traverse Path while keeping directory inode.
  Check if the directory in the file path exsists
  Else return -1
  If inode available
  Check if the directory exsist
  Recursively delete the content of the directory
*/
// DELETE A DIRECTORY
int DD(char* dirname)
{
  if (strcmp(dirname,"/")==0)
  {
    printf ("error: cannot delete root directory \n");
    return -1;
  }
  char * token = strtok(dirname,"/");
  char * ChildDirectory = token;
  char * ParentDirectory = "/";

  int address = getInode(ParentDirectory);

  int i = 0;
  struct inode ParentInode;
  struct DirectoryBlock ParentDirectoryBlock;


  lseek(myfs,(128+(address*56)),SEEK_SET);
  read(myfs,(char*)&ParentInode,56);

  int value = ParentInode.blockptrs[0];
  lseek(myfs,value*1024,SEEK_SET);
  read(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));

  while(token!=NULL)
  {
    token = strtok(NULL,"/");
    if (token !=NULL)
    {
      int flag = 0;
```

```c
      for (i = 2; i < 17;i++)
      {
        if(strcmp(ParentDirectoryBlock.DirectoryTable[i].name,ChildDirectory) ==
    0){
          address = ParentDirectoryBlock.DirectoryTable[i].inode;
          flag = 1;
          break;
          }
      }
      if (flag == 0)
      {
        printf("error: The %s directory in the given path does not exsist \n",
    ChildDirectory);
        return -1;
      }

    lseek(myfs,(128+(address*56)),SEEK_SET);
    read(myfs,(char*)&ParentInode,56);
    value = ParentInode.blockptrs[0];
    lseek(myfs,value*1024,SEEK_SET);
    read(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));
    ParentDirectory = ChildDirectory;
    ChildDirectory = token;

    }
  }

  int ChildInodeAddress = getInode(ChildDirectory);
  if (ChildInodeAddress == -1)
  {
    printf("error: the directory does not exsist \n");
    return -1;
  }
  Recurse(ChildInodeAddress);
  return 0;
}
/*

Traverse the Inode Table
Print all those inodes which our currently being used.
*/

// LIST ALL FILES
void LL()
{
  struct inode Table[16] ;
  lseek(myfs,128,SEEK_SET);
  read(myfs,(char*)&Table,(16*56));
  int i = 0;
  for(i = 0 ; i < 16; i++)
  {
    if (Table[i].used == 1)
    {
    printf("Name : %s , Dir : %d  , Size : %d \n",Table[i].name,Table[i].dir,
    Table[i].size);

    }
  }
}
/*
 * main
 *
 */
void printInodeTable(){
  struct inode Table[16] ;
  lseek(myfs,128,SEEK_SET);
  read(myfs,(char*)&Table,(16*56));
  int i = 0;
  for(i = 0 ; i < 16; i++)
  {
    if (Table[i].used == 1)
    {
    printf("Index : %d , Name : %s , Dir : %d  , Size : %d , Used : %d \n",i,
    Table[i].name,Table[i].dir,Table[i].size,Table[i].used);
    printf("BlkPtr[0] : %d BlkPtr[1] : %d BlkPtr[2] : %d BlkPtr[3] : %d BlkPtr[4]
     : %d BlkPtr[5] : %d BlkPtr[6] : %d BlkPtr[7] : %d \n",
    Table[i].blockptrs[0],Table[i].blockptrs[1],Table[i].blockptrs[2],Table[i].
    blockptrs[3],Table[i].blockptrs[4],Table[i].blockptrs[5],
    Table[i].blockptrs[6],Table[i].blockptrs[7]);
    }
  }

}
int main (int argc, char* argv[]) {

  // while not EOF
  // read command

  // parse command

  // call appropriate function
  myfs = open("myfs", O_RDWR); // read-write enabled
  if (myfs == -1)
  {
    printf("myfs does not exsist \n");
    initiliaze();
  }


  FILE * stream = fopen(argv[1],"r");
  char *Line = NULL;
  char Command[3];
  size_t len  = 0;
  while(getline(&Line,&len,stream)!=-1)
  {

    sscanf(Line,"%[^ \n] %[^\n]",Command,Line);
    // Create File
    if (strcmp(Command,"CR") == 0)
    {
      char * FileName = strtok(Line," ");
      int Size = atoi(strtok(NULL," "));
      CR(FileName,Size);
    }
    // Delete File
    else if (strcmp(Command,"DL") == 0)
    {
      DL(Line);
    }
    // Copy File
    else if (strcmp(Command,"CP") == 0)
    {
      char * srcname = strtok(Line," ");
      char * dstname = strtok(NULL," ");
      CP(srcname,dstname);

    }
    // Move a File
    else if (strcmp(Command,"MV") == 0)
    {
      printf("MV \n");
      char * srcname = strtok(Line," ");
      char * dstname = strtok(NULL," ");
      MV(srcname,dstname);

    }
    // Create Directory
    else if (strcmp(Command,"CD") == 0)
    {
      Line = strtok(Line,"\n");
      CD(Line);

    }
    // Remove Directory
    else if (strcmp(Command,"DD") == 0)
    {
      DD(Line);
    }
    // List all files
    else if (strcmp(Command,"LL") == 0)
    {
     LL();
    }
  }
  close(myfs);
  fclose(stream);
  free(Line);






  return 0;
}
```

Listing 1: filesystem.c