CSCI131 Assignment 3

Mohamed Asjad Athick (4970512)

22 September 2015

Report

Table of Contents

[Introduction 3](#_Toc429165046)

[Evidence for Correct Operation 4](#_Toc429165047)

[C 4](#_Toc429165048)

[Assembly 5](#_Toc429165049)

[Code Explanation 6](#_Toc429165050)

[C code 6](#_Toc429165051)

[Explanation of Assembly functions 9](#_Toc429165052)

[Mean 9](#_Toc429165053)

[Mode 10](#_Toc429165054)

[Max and Min 12](#_Toc429165055)

[Assembly Code 13](#_Toc429165056)

# Introduction

The program simulates a simple continuous file system, as described in the assignment prompt. The file Disk.c implements the prototypes given in the assignment (Disk.h).

# Source code (Disk.c)

//

// Disk.c

// 131a3

//

// Created by Asjad Athick on 17/09/2015.

// Copyright (c) 2015 Asjad Athick. All rights reserved.

//

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include "Disk.h"

static int diskblockcontent[DISKBLOCKS];

static int diskblockbitmap[DISKBLOCKS];

static Direntry directory[MAXFILES];

//stats variables

static int dirEntryCount=0;

static int createFileCalls=0;

static int deleteFileCalls=0;

static int compactionCalls=0;

static int freeBlocksInMemory=0;

static int blocksAlloc=0;

void **initialiseFileSystem**(){

int i;

for (i=0; i<DISKBLOCKS; i++) {

diskblockcontent[i]=0;

diskblockbitmap[i]=0;

}

for (i=0; i<MAXFILES; i++) {

directory[i].filename[0]='**\0**';

directory[i].size=0;

directory[i].start=0;

directory[i].symbol='x';

}

dirEntryCount=0;

}

FileSysErrors **createFile**(const char\* filename, int size){

//find free block

if (dirEntryCount>MAXFILES) {

return DIRECTORY\_FULL;

}

int start=findFreeBlock(size);

if (start<0) {

//free space not found

return CREATE\_FAIL;

}

if((DISKBLOCKS-blocksAlloc)<size){

return CREATE\_FAIL;

}

//check for duplicate names

int i;

int found=0;

for (i=0; i<MAXFILES; i++) {

if (strcmp(directory[i].filename,filename)==0) {

//found

found=1;

break;

}

}

if (found==1) {

return DUPLICATE\_NAME;

}

if (size<=0) {

return ZERO\_SIZE;

}

//make direntry

if(dirEntryCount<MAXFILES){

strcpy(directory[dirEntryCount].filename,filename);

directory[dirEntryCount].start=start;

directory[dirEntryCount].size=size;

directory[dirEntryCount].symbol=(rand()%65)+48;

dirEntryCount++;

//update bit map

writeToBitMap(start, size, 1);

blocksAlloc+=size;

createFileCalls++;

}

return NO\_ERR;

}

FileSysErrors **deleteFile**(const char\* filename){

//find file, return if non existent

int i;

int found=0; //flag

for (i=0; i<MAXFILES; i++) {

if (strcmp(directory[i].filename, filename)==0) {

//found

found=1;

//delete

writeToBitMap(directory[i].start, directory[i].size, 0);

directory[i].filename[0]='**\0**';

dirEntryCount--;

deleteFileCalls++;

blocksAlloc-=directory[i].size;

}

}

if (found==0) {

return NON\_EXISTENT\_FILE;

}

for (i=0; i<MAXFILES; i++) {

if (directory[i].filename[0]=='**\0**') {

//find next item

int index;

int found=0; //flag

for (index=i; index<MAXFILES; index++) {

if (directory[index].filename[0]!='**\0**') {

found=1;

break;

}

}

if (found==1) {

int tmpDirIndexStart, tmpDirIndexSize,tmpDirIStart,tmpDirISize;

tmpDirIndexStart=directory[index].start;

tmpDirIndexSize=directory[index].size;

tmpDirIStart=directory[i-1].start;

tmpDirISize=directory[i-1].size;

directory[i]=directory[index];

directory[index].filename[0]='**\0**';

directory[index].start=0;

directory[index].size=0;

directory[index].symbol='x';

if (i==0) {

directory[i].start=0;

}else{

directory[i].start=(directory[i-1].start+directory[i-1].size);

}

moveBack(tmpDirIndexStart, tmpDirIndexSize, (tmpDirIndexStart-(tmpDirIStart+tmpDirISize)));

}

}

}

return NO\_ERR;

}

FileSysErrors **writeBlock**(const char\* filename, int block, int value){

//search for file

int i;

int found=0; //flag

for (i=0; i<MAXFILES; i++) {

if (strcmp(directory[i].filename,filename)==0) {

found=1;

break;

}

}

if (found==0) {

//filename does not exist

return NON\_EXISTENT\_FILE;

}

//check if block > size

if (block>=directory[i].size || block<0) {

return INVALID\_BLOCK;

}

diskblockcontent[(directory[i].start)+block]=value;

return NO\_ERR;

}

FileSysErrors **readBlock**(const char\* filename, int block, int\* vp){

//search for file

int i;

int found=0; //flag

for (i=0; i<MAXFILES; i++) {

if (strcmp(directory[i].filename,filename)==0) {

found=1;

break;

}

}

if (found==0) {

//filename does not exist

return NON\_EXISTENT\_FILE;

}

//check if block > size

if (block>=directory[i].size || block<0) {

return INVALID\_BLOCK;

}

\*vp=diskblockcontent[(directory[i].start)+block];

return NO\_ERR;

}

int **compactFiles**(){

//delete file keeps compacts up to date

int i;

//count number of freeblocks and return

int freeblocks=0;

int iscounting=0; //flag

for (i=0; i<DISKBLOCKS; i++) {

if (diskblockbitmap[i]==0) {

if (iscounting==0) {

iscounting=1;

}

freeblocks++;

}

else{

freeblocks=0;

iscounting=0;

}

}

compactionCalls++;

freeBlocksInMemory=freeblocks;

return freeblocks;

}

void **showHistory**(){

printf(**"**Number of file create operations**\t\t**:%d**\n"**,createFileCalls);

printf(**"**Number of file delete operations**\t\t**:%d**\n"**,deleteFileCalls);

printf(**"**Number of file compaction operations**\t\t**:%d**\n\n"**,compactionCalls);

printf(**"**Current number of directory entries**\t\t**:%d**\n"**,dirEntryCount);

printf(**"**Current number of disk blocks allocated**\t\t**:%d**\n"**,blocksAlloc);

printf(**"**Current number of disk blocks free**\t\t**:%d**\n"**,DISKBLOCKS-blocksAlloc);

}

void **moveBack**(int originalIndex, int size, int movebackBy){

int i;

for (i=originalIndex; i<originalIndex+size; i++) {

diskblockbitmap[(i-movebackBy)]=diskblockbitmap[i];

diskblockcontent[(i-movebackBy)]=diskblockcontent[i];

diskblockcontent[i]=0;

diskblockbitmap[i]=0;

}

}

void **displayDisk**(){

printf(**"**Directory listing:**\n"**);

printf(**"**# **\t** Filename **\t\t** Start **\t\t** Size **\t\t** (symbol)**\n"**);

int cnt=1;

int i;

//dir listing

for (i=0; i<MAXFILES; i++) {

if (directory[i].filename[0]!='**\0**') {

printf(**"**%d **\t** %s **\t\t** %d **\t\t** %d **\t\t** %c **\n"**,cnt,directory[i].filename,directory[i].start,directory[i].size,directory[i].symbol);

cnt++;

}

}

//disk map

printf(**"\n\n"**);

int filecnt=0;

for (i=0; i<DISKBLOCKS; i++) {

if (diskblockbitmap[i]==1) {

int filesize=directory[filecnt].size;

int j;

for (j=0; j<filesize; j++) {

printf(**"**%c**"**,directory[filecnt].symbol);

}

i+=filesize-1;

filecnt++;

}

else

printf(**"**.**"**);

}

printf(**"\n\n"**);

}

int **findFreeBlock**(int size){

int i;

int freecnt=0; //number of continuous blocks found

int freeIndex=0; //value of start of block

int freeset=0; //flag

for (i=0; i<DISKBLOCKS; i++) {

if (diskblockbitmap[i]==0) {

if (freeset==0) {

freeset=1;

freeIndex=i;

}

freecnt++;

if (freecnt>=size) {

return freeIndex;

}

}

else{

freeset=0;

}

}

return -1;

}

void **writeToBitMap**(int start, int size, int value){

int i;

for (i=start; i<(start+size); i++) {

diskblockbitmap[i]=value;

}

}

# Source code (Disk.h)

//

// Disk.h

// 131a3

//

// Created by Asjad Athick on 17/09/2015.

// Copyright (c) 2015 Asjad Athick. All rights reserved.

//

#ifndef DISK\_H

#define DISK\_H

#define MAXFILES 64

#define DISKBLOCKS 512

#define NAMESIZE 32

struct direntry{

char filename[NAMESIZE];

int start;

int size;

char symbol;

};

typedef struct direntry Direntry;

enum filesyserrors {NO\_ERR, DIRECTORY\_FULL, CREATE\_FAIL, NON\_EXISTENT\_FILE, DUPLICATE\_NAME, INVALID\_BLOCK, ZERO\_SIZE};

typedef enum filesyserrors FileSysErrors;

void **initialiseFileSystem**();

FileSysErrors **createFile**(const char\* filename, int size);

FileSysErrors **deleteFile**(const char\* filename);

FileSysErrors **writeBlock**(const char\* filename, int block, int value);

FileSysErrors **readBlock**(const char\* filename, int block, int\* vp);

int **compactFiles**();

void **displayDisk**();

void **showHistory**();

//helper functions

int **findFreeBlock**(int size);

void **writeToBitMap**(int start, int size, int value);

void **moveBack**(int originalIndex, int size, int movebackBy);

#endif