ECE 421 Assignment 3

Brandon Rubio C23727769

FILE: “t.b” in /home/students/bur2/OS/Assignments/test

I have been able to implement the following:

* Mounting, formatting, and dismounting the disc
* A way to enter and remove entries from a directory
  + Each entry has its own header block
    - Header block includes:
      * Type of file
      * Date created/modified/accessed
      * permissions
* Block management system
  + Allows for finding free blocks and marking blocks as used
    - This is done through the stack system:
      * Stack.pop()
      * Stack.push()
      * Stack.top();
    - Every time the stack system is used it updates the superblock stored in cache
      * Everytime the superblock is updated the system checks if stackWindow needs to be updated and updates it accordingly

I have been unable to complete the following:

* The higher level read/write functions for the file system
* The iosb that ties in with those higher level functions

I am in the process of debugging and solidifying the code that I have been able to complete so I do not have screenshots of a working program. However, I will submit what I have done so far. Sometime soon I will resubmit a more complete version of the program.

**Code:**

import "io"

manifest {

sbPos = 0, //super block is located in block zero

sb.rootdir = 0,

sb.topBlock = 1,

sb.endBlock = 2,

sb.numberOfFreeBlocks = 3,

sb.top = 4,

sb.end = 5,

sb.topPosWindow = 6

}

//these vectors are copied from the mounted disk

let superblock = vec(128);

let rootdir = vec(128);

let stackWindow = vec(256);

//max refers to maximum index to iterate to

let strncpy(dest, src, max) be{

let i = 0, ended = false;

while i <= max do

{ test ended then

byte i of dest := 0

else

{ let c = byte i of src;

byte i of dest := c;

if c = 0 then

ended := true }

i +:= 1 }

}

let strncmp(a, b, max) be {

for i = 0 to max do

{ if byte i of a <> byte i of b then

resultis false;

if byte i of a = 0 then

resultis true }

resultis true

}

// checks if two strings are equal

let equals(s, t) be {

let i = 0;

while byte i of s = byte i of t do

{ if byte i of s = 0 then

resultis true;

i +:= 1 }

resultis false

}

let outsn(s, max) be { //prints a slected amount of characters from a string

for i = 0 to max do

test byte i of s = 0 then

return

else

outch(byte i of s)

}

let format() be { //clears and formats the entire disc

let buffer = vec(128), r; //512 byte buffer, r for result of devctl functions

let stackBlocksUsed, discSize, numFreeBlocks; // in blocks

let rootdirPos;

// used to initialize superblock

let top, topBlock, end, endBlock; // topBlock reffers to the block number that top is located in, similarly for endBlock

let topPosWindow; // top position in window

// used for creating the free list

let storageBlock, blockNumToStore, posRem;

discSize := devctl(DC\_DISC\_CHECK, 1);

if discSize <= 0 then{

out("format error, disc unavailable\n");

return;

}

out("format the disc\n");

// clear the disc

for i = 0 to 127 do { // set buffer to 0

buffer ! i := 0;

}

r := devctl(DC\_DISC\_WRITE, 1, 0, discSize, buffer);

if r < 0 then {

out("format error %d, unable to clear disc\n", r);

return;

}

stackBlocksUsed := (discSize + 127) / 128;

numFreeBlocks := discSize - stackBlocksUsed - 2;

rootdirPos := stackBlocksUsed + 1;

top := 0; //set to zero when formatiing

topBlock := 1; // set to one when formatting

end := numFreeBlocks - 1;

endBlock := end / 128 + 1;

topPosWindow := 0; //set to zero when formatting

// initialize superblock

buffer ! sb.rootdir := rootdirPos;

buffer ! sb.topBlock := topBlock;

buffer ! sb.endBlock := endBlock;

buffer ! sb.numberOfFreeBlocks := numFreeBlocks;

buffer ! sb.top := top; //top pos on disc

buffer ! sb.end := end;

buffer ! sb.topPosWindow := topPosWindow;

r := devctl(DC\_DISC\_WRITE, 1, sbPos, 1, buffer);

if r < 0 then {

out("format error %d, unable to initialize superblock\n", r);

return;

}

//create free list

blockNumToStore := stackBlocksUsed + 2;

storageBlock := 1;

for pos = 0 to end do {

posRem := pos rem 128;

buffer ! posRem := blockNumToStore;

if posRem = 127 then {

if storageBlock > stackBlocksUsed then {

out("format error, free list storage overflow\n");

return;

}

devctl(DC\_DISC\_WRITE, 1, storageBlock, 1, buffer);

storageBlock +:= 1;

}

blockNumToStore +:= 1;

}

// if there is not enough to fill the buffer then write the rest to final block

devctl(DC\_DISC\_WRITE, 1, storageBlock, 1, buffer);

// may cause difficulty debugging if value is read after "end" of stack since buffer my contain valuse from previous use

}

// mount copies the super block, root directory, and stack view from disc to their corresponding vectors

let mount() be {

let r; //result of devctl function

let buffer = vec(128);

let stackBlocksUsed, discSize;

let top, topBlock;

out("mount the disk\n");

//load superblock

r := devctl(DC\_DISC\_READ, 1, sbPos, 1, superblock);

if r < 0 then{

out("mount error %d, unable to load superblock\n", r);

return;

}

//load rootdir

r := devctl(DC\_DISC\_READ, 1, superblock ! sb.rootdir, 1, rootdir);

if r < 0 then{

out("mount error %d, unable to load rootdir\n", r);

return;

}

//load stackWindow

discSize := devctl(DC\_DISC\_CHECK, 1);

if discSize <= 0 then {

out("mount error, disc unavailable\n");

return;

}

stackBlocksUsed := (discSize + 127) / 128;

topBlock := superblock ! sb.topBlock;

top := superblock ! sb.top;

r := devctl(DC\_DISC\_READ, 1, topBlock, 1, buffer);

if r < 0 then {

out("mount error %d, unable to read block into stackWindow\n", r);

return;

}

for i = 0 to 127 do {

stackWindow ! i := buffer ! i;

}

if (topBlock + 1 <= stackBlocksUsed) then {

r := devctl(DC\_DISC\_READ, 1, topBlock + 1, 1, buffer);

if r < 0 then {

out("mount error %d, unable to read second block into stackWindow\n", r);

return;

}

for i = 0 to 127 do {

stackWindow ! (i + 128) := buffer ! i;

}

}

}

let dismount() be {

let r; // result of devctl

let buffer = vec(128);

let stackBlocksUsed, discSize; // in blocks

let topBlock, top, topPosWindow;

let temp = vec(128);

out("dismount the disc\n");

//write back to super block

r := devctl(DC\_DISC\_WRITE, 1, sbPos, 1, superblock);

if r < 0 then {

out("dismount error %d, unable to write to super block\n", r);

}

//write back to root directory

r := devctl(DC\_DISC\_WRITE, 1, superblock ! sb.rootdir, 1, rootdir);

if r < 0 then {

out("dismount error %d, unable to write to root directory\n", r);

}

//write back stack window to associated memory in disc

discSize := devctl(DC\_DISC\_CHECK, 1);

if discSize <= 0 then {

out("dismount error, disc unavailable\n");

return;

}

stackBlocksUsed := (discSize + 127) / 128;

topBlock := superblock ! sb.topBlock;

top := superblock ! sb.top;

topPosWindow := superblock ! sb.topPosWindow;

test topPosWindow <= 127 then {

for i = 0 to 127 do {

buffer ! i := stackWindow ! i;

}

r := devctl(DC\_DISC\_WRITE, 1, topBlock, 1, buffer);

if r < 0 then {

out("dismount error %d, unable to write first block (1)\n", r);

}

if topBlock + 1 <= stackBlocksUsed then {

for i = 0 to 127 do {

buffer ! i := stackWindow ! (i + 128);

}

r := devctl(DC\_DISC\_WRITE, 1, topBlock + 1, 1, buffer);

if r < 0 then {

out("dismount error %d, unable to write second block (1)\n", r);

}

}

}

else {

for i = 0 to 127 do {

buffer ! i := stackWindow ! (i + 128);

}

r := devctl(DC\_DISC\_WRITE, 1, topBlock, 1, buffer);

if r < 0 then {

out("dismount error %d, unable to write first block (2)\n", r);

}

for i = 0 to 127 do {

buffer ! i := stackWindow ! i;

}

r := devctl(DC\_DISC\_WRITE, 1, topBlock - 1, 1, buffer);

if r < 0 then {

out("dismount error %d, unable to write second block (2)\n", r);

}

}

}

let lookup(name, dirBlockNumber) be { // return block number of an entry to a given directory

let r;

let dir = vec(128);

r := devctl(DC\_DISC\_READ, 1, dirBlockNumber, 1, dir);

if r < 0 then {

out("lookup error %d, unable to read block %d to dir\n", r, dirBlockNumber);

resultis -1;

}

if dir ! 0 = 0 /\ dirBlockNumber <> superblock ! sb.rootdir then {

out("lookup error, block %d is not a directory\n", dirBlockNumber);

resultis -1;

}

out("lookup \"%s\" in the directory at block %d\n", name, dirBlockNumber);

test dirBlockNumber = superblock ! sb.rootdir then { //if rootdir

for ptr = 0 to 127 by 3 do { // rootdir has no metadata stored within it. By 3 since 2 words = file name, 1 word = pointer in dir.

if dir ! ptr <> 0 then {

if strncmp(name, dir + ptr, 15) then {

resultis dir ! (ptr + 2);

}

}

}

}

else {

for ptr = 8 to 127 by 3 do { // metadata = 8 words, therefore start at 8

if dir ! ptr <> 0 then {

if strncmp(name, dir + ptr, 15) then {

resultis dir ! (ptr + 2);

}

}

}

}

out("\"%s\" not found in block %d\n", name, dirBlockNumber);

resultis -1

}

let find.free.entry(dirBlockNumber) be { // returns position of next free space in select direcotry for an entry's header block, -1 otherwise

let r;

let dir = vec(128);

r := devctl(DC\_DISC\_READ, 1, dirBlockNumber, 1, dir);

if r < 0 then {

out("find.free.entry() error %d, unable to read block %d to dir\n", r, dirBlockNumber);

}

test superblock ! 0 = dirBlockNumber then { // if rootdir

for ptr = 0 to 127 by 3 do {

if dir ! ptr = 0 /\ dir ! (ptr + 1) = 0 /\ dir ! (ptr + 2) = 0 then {

resultis ptr;

}

}

}

else {

for ptr = 8 to 127 by 3 do { //check after metadata for every other directory

if dir ! ptr = 0 /\ dir ! (ptr + 1) = 0 /\ dir ! (ptr + 2) = 0 then {

resultis ptr;

}

}

}

resultis -1;

}

let stackWindow.update(newTopPosWindow) be { // goal of stackWindow is to keep top in the middle of the window

let r;

let buffer = vec(128);

test newTopPosWindow = 256 then { // shift window right

//save leftmost window back to disc

for i = 0 to 127 do {

buffer ! i := stackWindow ! i;

}

r := devctl(DC\_DISC\_WRITE, 1, (superblock ! sb.topBlock) - 2, 1, buffer);

if r < 0 then {

out("stackWindow.update error %d, unable to write stackWindow back to disc (1)\n", r);

resultis -1;

}

// right becomes left

for i = 0 to 127 do {

stackWindow ! i := stackWindow ! (i + 128);

}

// import new right

r := devctl(DC\_DISC\_READ, 1, superblock ! sb.topBlock, 1, buffer);

if r < 0 then {

out("stackWindow.update error %d, unable to read stack to stackWindow (1)\n", r);

resultis -1;

}

for i = 0 to 127 do {

stackWindow ! (i + 128) := buffer ! i;

}

superblock ! sb.topPosWindow := 128;

}

else { // shift window left

//save rightmost window back to disc

for i = 0 to 127 do {

buffer ! i := stackWindow ! (i + 128);

}

r := devctl(DC\_DISC\_WRITE, 1, (superblock ! sb.topBlock) + 2, 1, buffer);

if r < 0 then {

out("stackWindow.update error %d, unable to write stackWindow back to disc (2)\n", r);

resultis -1;

}

// left becomes right

for i = 0 to 127 do {

stackWindow ! (i + 128) := stackWindow ! i;

}

// import new left

r := devctl(DC\_DISC\_READ, 1, superblock ! sb.topBlock, 1, buffer);

if r < 0 then {

out("stackWindow.update error %d, unable to read stack to stackWindow (2)\n", r);

resultis -1;

}

for i = 0 to 127 do {

stackWindow ! i := buffer ! i;

}

superblock ! sb.topPosWindow := 127;

}

resultis 0;

}

let stackWindow.checkUpdate(newTopPosWindow) be { //to be only used in superblock.update();

test newTopPosWindow = -1 \/ newTopPosWindow = 256 then {

resultis stackWindow.update(newTopPosWindow);

}

else {

resultis 0;

}

}

let superblock.update(newTop, newTopPosWindow, newNumberOfFreeBlocks) be { //to be only used in stack functions

let newTopBlock;

newTopBlock := newTop / 128 + 1;

superblock ! sb.top := newTop;

superblock ! sb.topBlock := newTopBlock;

superblock ! sb.topPosWindow := newTopPosWindow;

superblock ! sb.numberOfFreeBlocks := newNumberOfFreeBlocks;

resultis stackWindow.checkUpdate(newTopPosWindow);

}

let stack.top() be {

if superblock ! sb.top > superblock ! sb.end then {

resultis -1;

}

resultis stackWindow ! (superblock ! sb.topPosWindow);

}

let stack.push(blocknumber) be { // pushing a new free block onto the stack

// returns -1 on error, 0 otherwise

let r;

let top, newTop;

let topPosWindow, newTopPosWindow;

let numberOfFreeBlocks, newNumberOfFreeBlocks;

top := superblock ! sb.top;

topPosWindow := superblock ! sb.topPosWindow;

numberOfFreeBlocks := superblock ! sb.numberOfFreeBlocks;

newTop := top - 1;

newTopPosWindow := topPosWindow - 1;

newNumberOfFreeBlocks := numberOfFreeBlocks + 1;

if newTop < 0 then {

out("stack.push() error: newTop = %d < 0\n", newTop);

resultis -1;

}

r := superblock.update(newTop, newTopPosWindow, newNumberOfFreeBlocks);

if r < 0 then {

resultis r;

}

stackWindow ! (superblock ! sb.topPosWindow) := blocknumber;

resultis 0;

}

let stack.pop() be { // popping a free block from the stack

// returns -1 on error, free block number otherwise

let r;

let top, newTop;

let topPosWindow, newTopPosWindow;

let numberOfFreeBlocks, newNumberOfFreeBlocks;

top := superblock ! sb.top;

topPosWindow := superblock ! sb.topPosWindow;

numberOfFreeBlocks := superblock ! sb.numberOfFreeBlocks;

newTop := top + 1;

newTopPosWindow := topPosWindow + 1;

newNumberOfFreeBlocks := numberOfFreeBlocks - 1;

if newTop > ((superblock ! sb.end) + 1) then {

out("stack.pop() error: newTop = %d > end = %d\n", newTop, superblock ! sb.end);

resultis -1;

}

r := superblock.update(newTop, newTopPosWindow, newNumberOfFreeBlocks);

if r < 0 then {

resultis r;

}

resultis stackWindow ! ((superblock ! sb.topPosWindow) - 1);

}

let find.free.block() be {

resultis stack.top();

}

let use.free.block() be {

resultis stack.pop();

}

let createHeaderBlock(type, blocknumber, permissions) be { //creates a header block for a file or directory in the given block

let r;

let dir = vec(128);

r := devctl(DC\_DISC\_READ, 1, blocknumber, 1, dir);

if r < 0 then {

out("createHeaderBlock error %d, unable to read block %d to dir\n", r, blocknumber);

resultis -1;

}

dir ! 0 := type; // file type

datetime2(dir ! 1); // date created //datetime2 needs at least two words

datetime2(dir ! 3); // date modified

datetime2(dir ! 5); // date accessed

dir ! 7 := permissions;

r := devctl(DC\_DISC\_WRITE, 1, blocknumber, 1, dir);

if r < 0 then {

out("createHeaderBlock error %d, unable to write dir to block %d\n", r, blocknumber);

resultis -1;

}

}

let enter(name, dirBlockNumber, type, permissions) be { // add an entry to a given directory (only establishes header blocks for each type)

// permissions can be: (001xb):execute (010xb):write (100xb):read and any combination of the three

let r;

let ptr; // will store the block number of entry's header block

let dir = vec(128);

let posFreeSpace, freeBlockNum;

r := devctl(DC\_DISC\_READ, 1, dirBlockNumber, 1, dir);

if r < 0 then {

out("enter error %d, unable to read block %d to dir\n", r, dirBlockNumber);

resultis -1;

}

if dir ! 0 = 0 /\ dirBlockNumber <> superblock ! sb.rootdir then {

out("enter error, given block is not a directory\n");

resultis -1;

}

if type <> 0 /\ type <> 1 then {

out("enter error, given type can only be 0 or 1 => file or directory\n");

resultis -1;

}

// is there space in the parent directory? (3 words)

posFreeSpace := find.free.entry(dirBlockNumber);

if posFreeSpace = -1 then {

out("directory is full\n");

resultis -1;

}

// is there space to create header block?

// Note: might make header block that is unable to point to any data, since no space left in disc

r := find.free.block();

if r = -1 then {

out("disc is full\n");

resultis -1;

}

//create entry

strncpy(dir ! posFreeSpace, name, 15);

ptr := use.free.block();

if ptr < 0 then {

out("enter error(), use.free.block() error\n");

resultis -1;

}

dir ! (posFreeSpace + 2) := ptr;

//create headerblock

r := createHeaderBlock(type, ptr, permissions);

if r < 0 then {

out("enter error(), createHeaderBlock() error\n");

resultis -1;

}

resultis 0;

}

let releaseBlock(blocknumber) be {

let r;

let buffer = vec(128);

//clear block

for i = 0 to 127 do {

buffer ! i := 0;

}

r := devctl(DC\_DISC\_WRITE, 1, blocknumber, 1, buffer);

if r < 0 then {

out("releaseBlock error %d, unable to write buffer to block %d\n", r, blocknumber);

resultis -1;

}

r := stack.push(blocknumber);

if r < 0 then {

out("releaseBlock error %d, unable to push block %d to stack\n", r, blocknumber);

resultis -1;

}

resultis 0;

}

let remove(name, dirBlockNumber) be { // remove a directory entry

let r;

let dir = vec(128);

r := devctl(DC\_DISC\_READ, 1, dirBlockNumber, 1, dir);

if r < 0 then {

out("remove error %d, unable to read block %d to dir\n", r, dirBlockNumber);

resultis -1;

}

out("remove \"%s\" from directory %d\n", name, dirBlockNumber);

test superblock ! 0 = dirBlockNumber then { // if rootdir

for ptr = 0 to 127 by 3 do {

if strncmp(name, ptr, 15) then {

r := releaseBlock(dir ! (ptr + 2));

if r < 0 then {

out("remove error %d, releaseBlock error rootdir\n");

resultis -1;

}

dir ! ptr := 0;

dir ! (ptr + 1) := 0;

dir ! (ptr + 2) := 0;

r := devctl(DC\_DISC\_WRITE, 1, dirBlockNumber, 1, dir);

if r < 0 then {

out("remove error %d, unable to write dir to block %d\n", r, dirBlockNumber);

resultis -1;

}

resultis 0;

}

}

//otherwise not found

resultis -1;

}

else {

for ptr = 8 to 127 by 3 do { // check after metadata for every other directory

if strncmp(name, ptr, 15) then {

r := releaseBlock(dir ! (ptr + 2));

if r < 0 then {

out("remove error %d, releaseBlock error dir\n");

resultis -1;

}

dir ! ptr := 0;

dir ! (ptr + 1) := 0;

dir ! (ptr + 2) := 0;

r := devctl(DC\_DISC\_WRITE, 1, dirBlockNumber, 1, dir);

if r < 0 then {

out("remove error %d, unable to write dir to block %d\n", r, dirBlockNumber);

resultis -1;

}

resultis 0;

}

}

//otherwise not found

resultis -1;

}

}

let print.dir() be {

let v = vec(7);

out("list the directory\n");

for ptr = 0 to 127 by 4 do {

if rootdir ! ptr <> 0 then {

outsn(rootdir + ptr, 7);

datetime(rootdir ! (ptr + 2), v);

out(", %04d-%02d-%02d %02d:%02d:%02d, block %d\n",

v ! 0, v ! 1, v ! 2, v ! 4, v ! 5, v ! 6, rootdir ! (ptr + 3))

}

}

}

let print.superblock() be {

out("sb.rootdir = %d\n", superblock ! sb.rootdir);

out("sb.topBlock = %d\n", superblock ! sb.topBlock);

out("sb.endBlock = %d\n", superblock ! sb.endBlock);

out("sb.numberOfFreeBlocks = %d\n", superblock ! sb.numberOfFreeBlocks);

out("sb.top = %d\n", superblock ! sb.top);

out("sb.end = %d\n", superblock ! sb.end);

out("sb.topPosWindow = %d\n", superblock ! sb.topPosWindow);

}

let print.freeList() be {

let t;

let p;

for i = 0 to superblock ! sb.end do {

p := stack.pop();

t := stack.top();

out("top = %d pop = %d\n", t, p);

out("\n");

print.superblock();

}

}

let print.freeList2() be {

let r;

let buffer = vec(128);

for i = superblock ! sb.topBlock to superblock ! sb.endBlock do {

r := devctl(DC\_DISC\_READ, 1, i, 1, buffer);

if r < 0 then {

out("error %d\n", r);

return;

}

for j = 0 to 127 do {

out("%d\n", buffer ! j);

}

}

}

let print.stackWindow() be {

for i = 0 to 255 do {

out("%d\n", stackWindow ! i);

}

}

let write(name, contents) be { // contents is everything all in one string

let bn = find.free.block(), r;

out("create file \"%s\" with given content\n", name);

enter(name, bn);

r := devctl(DC\_DISC\_WRITE, 1, bn, 1, contents);

if r < 0 then

out("write failed at write/1\n");

resultis r

}

let read(name, buffer) be { // read entire contents into buffer

let blocknumber = lookup(name), r;

out("read file \"%s\" into buffer\n", name);

if blocknumber < 0 then

{ out("read: lookup failed\n");

resultis -1; }

r := devctl(DC\_DISC\_READ, 1, blocknumber, 1, buffer);

if r < 0 then

out("read failed at read/1\n");

resultis r

}

let read.line(string, size) be {

let length = 0;

size -:= 1;

while true do

{ let c;

if length = size then

{ byte length of string := 0;

resultis length; }

c := inch();

if c = '\n' then

{ byte length of string := 0;

resultis length; }

byte length of string := c;

length +:= 1; }

}

let read.lines(string, size) be {

let length = 0, last.was.newline = false, maybe.ending = false;

size -:= 1;

while true do

{ let c;

if length = size then

{ byte length of string := 0;

resultis length; }

c := inch();

if maybe.ending /\ c = '\n' then

{ byte length - 1 of string := 0;

resultis length }

maybe.ending := last.was.newline /\ c = '\*';

last.was.newline := c = '\n';

byte length of string := c;

length +:= 1; }

}

let start() be {

let file, dir;

file := 0;

dir := 1;

format();

mount();

print.superblock();

enter("abcdefghijklmno", superblock ! sb.rootdir, file, 7);

lookup("abcdefghijklmno", superblock ! sb.rootdir);

out("\n");

print.superblock();

}