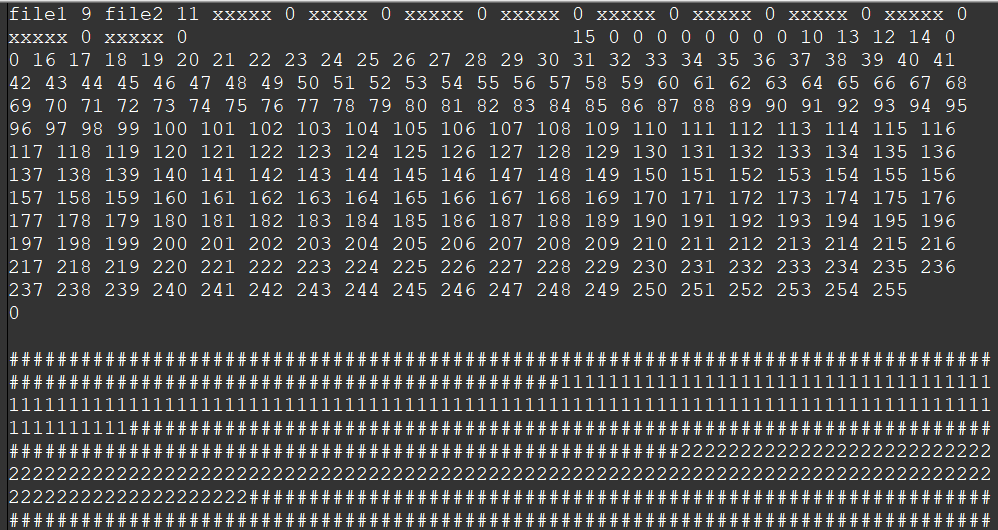
Zoe Veale

Lab 3 and 4



#include "fileSystem.h"

int main(int argc, char\* argv) {

Sdisk diskA("diskA", 256, 128);

FileSystem fsys("diskA", 256, 128);

fsys.NewFile("file1");

fsys.NewFile("file2");

std::string bfile1;

for (int i = 1; i <= 1024; i++) {

bfile1 += "1";

}

fsys.AddBlock("file1", bfile1);

//fsys.RemoveFile("file1");

//fsys.RemoveFile("file2");

std::string bfile2;

std::string block;

int blocknumber = 0;

for (int i = 1; i <= 2048; i++) {

bfile2 += "2";

}

blocknumber = fsys.AddBlock("file2", bfile2);

//fsys.DeleteBlock("file2", blocknumber);

return 0;

}

#ifndef FILESYSTEM\_H

#define FILESYSTEM\_H

#include <vector>

#include <sstream>

#include <iomanip>

#include <utility>

#include "sDisk.h"

class FileSystem : public Sdisk {

public:

FileSystem(std::string diskname, int numberofblocks, int blocksize);

int FileSystemClose();

int FileSystemSynch();

int NewFile(std::string file);

int RemoveFile(std::string file);

std::pair<int, int> GetFirstBlock(std::string file);

int AddBlock(std::string file, std::string blockBuffer);

int DeleteBlock(std::string file, int blocknumber);

int ReadBlock(std::string file, int blocknumber, std::string& buffer);

int WriteBlock(std::string file, int blocknumber, std::string buffer);

int NextBlock(std::string file, int blocknumber);

//std::vector<std::string> Block(std::string buffer, int blockSize);

private:

const int FILE\_ALLOCATION\_SIZE = 4;

const int FILE\_NAME\_SIZE = 10;

const int FAT\_MEMORY\_POSITION = 1;

const int ROOT\_MEMORY\_POSITION = 0;

const int FREE\_BLOCK = 0;

int FreeBlock();

int rootSize; // maximum number of entries in ROOT

int fatSize; // number of blocks occupied by FAT

std::vector<std::string> filename; // filenames in ROOT

std::vector<int> firstBlock; // firstblocks in ROOT

std::vector<int> fat; // FAT

};

#endif // !FILESYSTEM\_H

#include "fileSystem.h"

FileSystem::FileSystem(

std::string diskname, int numberofblocks, int blocksize) :

Sdisk(diskname, numberofblocks, blocksize),

rootSize(GetBlockSize() / FILE\_NAME\_SIZE),

fatSize(GetNumberOfBlocks()\* FILE\_ALLOCATION\_SIZE / GetBlockSize()) {

std::string buffer;

GetBlock(1, buffer);

if (buffer[1] == '#') {

//new disk

for (int i = 0; i < rootSize; i++) {

//TODO:: preprocessor defines for no file and first block zero, NULL maybe

filename.push\_back("xxxxx");

firstBlock.push\_back(0);

///

}

for (int i = 0; i < GetNumberOfBlocks(); i++) {

if (i < fatSize + FAT\_MEMORY\_POSITION) {

fat.push\_back(0);

}

else {

fat.push\_back(i + 1);

}

}

//fat.resize(GetNumberOfBlocks(), -1);

fat[0] = fatSize + FAT\_MEMORY\_POSITION;

fat[GetNumberOfBlocks() - FAT\_MEMORY\_POSITION] = 0;

FileSystemSynch();

}

else {

//load disk

buffer.clear();

GetBlock(0, buffer);

std::stringstream diskData(buffer);

for (unsigned int i = 0; i < rootSize; i++) {

std::string name;

int blockNumber;

diskData >> name >> blockNumber;

filename.push\_back(name);

firstBlock.push\_back(blockNumber);

}

diskData.str("");

diskData.clear();

for (int i = FAT\_MEMORY\_POSITION; i < fatSize + FAT\_MEMORY\_POSITION; i++) {

GetBlock(i, buffer);

diskData.str(diskData.str() + buffer);

}

for (unsigned int i = 0; i < GetNumberOfBlocks(); i++) {

int blockNumber;

diskData >> blockNumber;

fat.push\_back(blockNumber);

}

}

}

int FileSystem::FileSystemClose() {

FileSystemSynch();

return 1;

}

int FileSystem::FileSystemSynch() {

std::string syncBuffer;

for (int i = 0; i < filename.size(); i++) {

syncBuffer += filename[i] + " " + std::to\_string(firstBlock[i]) + " ";

}

for (int i = syncBuffer.size(); i < GetBlockSize(); i++) {

syncBuffer += " ";

}

PutBlock(ROOT\_MEMORY\_POSITION, syncBuffer);

syncBuffer.clear();

//TODO:: convert int to hex

for (unsigned i = 0; i < fat.size(); ++i) {

syncBuffer += std::to\_string(fat[i]) + " ";

}

for (int i = syncBuffer.size(); i < GetBlockSize() \* fatSize; i++) {

syncBuffer += " ";

}

for (int i = 0; i < fatSize; i++) {

PutBlock(i + 1, syncBuffer.substr(i \* GetBlockSize(), GetBlockSize()));

}

return 0;

}

int FileSystem::NewFile(std::string file) {

for (std::string i : filename) {

if (i == file)

return 0; // file exists already

}

int freeFileIndex = 0;

for (int i : firstBlock) {

if (i == 0) {

break;

}

++freeFileIndex;

}

if (freeFileIndex >= rootSize) return 0;

//TODO:: fat index following new file needs to = 0

filename[freeFileIndex] = file;

firstBlock[freeFileIndex] = fat[FREE\_BLOCK];

fat[FREE\_BLOCK] = fat[fat[fat[FREE\_BLOCK]]];

fat[fat[firstBlock[freeFileIndex]]] = 0;

FileSystemSynch();

return 1;

}

int FileSystem::RemoveFile(std::string file) {

std::pair<int, int> firstBlockPair = GetFirstBlock(file);

if (firstBlockPair.first < fatSize + FAT\_MEMORY\_POSITION) return 0;

DeleteBlock(file, firstBlockPair.first);

filename[firstBlockPair.second] = "xxxxx";

fat[FREE\_BLOCK] = firstBlock[firstBlockPair.second];

firstBlock[firstBlockPair.second] = 0;

FileSystemSynch();

return 1;

}

//first element = firstBlock[index], second element = index

std::pair<int, int> FileSystem::GetFirstBlock(std::string file) {

std::pair<int, int> firstBlockPair;

int freeFileIndex = 0;

for (std::string i : filename) {

if (i == file)

break;

else

++freeFileIndex;

}

//TODO >= or just >??

if (freeFileIndex > rootSize) return std::pair<int, int>(0, 0);

firstBlockPair.first = firstBlock[freeFileIndex];

firstBlockPair.second = freeFileIndex;

FileSystemSynch();

return std::pair<int, int>(firstBlock[freeFileIndex], freeFileIndex);

}

int FileSystem::AddBlock(std::string file, std::string blockBuffer) {

int endBlock = 0;

int firstBlock = GetFirstBlock(file).first;

if (firstBlock < fatSize + FAT\_MEMORY\_POSITION) return 0;

for (int i = firstBlock; i < fat.size(); i = fat[i]) {

if (fat[i] == 0) {

endBlock = i;

break;

}

}

PutBlock(endBlock, blockBuffer);

fat[endBlock] = fat[FREE\_BLOCK];

fat[FREE\_BLOCK] = fat[fat[FREE\_BLOCK]];

fat[fat[endBlock]] = 0;

FileSystemSynch();

return 0;

}

int FileSystem::DeleteBlock(std::string file, int blocknumber) {

int startBlock = GetFirstBlock(file).first;

for (int i = startBlock; i < fat.size(); i = fat[i]) {

if (i == blocknumber) {

startBlock = blocknumber;

break;

}

if (fat[i] == 0) {

return 0;

}

}

for (int i = startBlock; i < fat.size(); i = fat[i]) {

if (fat[i] == 0) {

fat[i] = fat[FREE\_BLOCK];

fat[FREE\_BLOCK] = fat[startBlock];

return 1;

}

}

return 0;

}

int FileSystem::ReadBlock(std::string file, int blocknumber, std::string& blockBuffer) {

int startBlock = GetFirstBlock(file).first;

for (int i = startBlock; i < fat.size(); i = fat[i]) {

if (i == blocknumber) {

GetBlock(blocknumber, blockBuffer);

return 1;

}

}

return 0;

}

int FileSystem::WriteBlock(std::string file, int blocknumber, std::string blockBuffer) {

int startBlock = GetFirstBlock(file).first;

for (int i = startBlock; i < fat.size(); i = fat[i]) {

if (i == blocknumber) {

PutBlock(blocknumber, blockBuffer);

return 1;

}

}

return 0;

}

int FileSystem::NextBlock(std::string file, int blocknumber) {

return 0;

}

int FileSystem::FreeBlock() {

int index = 0;

for (int i : fat) {

if (i == -1) {

return index;

}

++index;

}

return 0;

}