|  |
| --- |
| */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* Programmer: Eugene Kim \* Quarter: Fall 2019 \* Course: CSE 461 Advanced Operating Systems \* Professor: Owen Murphy \* Assignment: Labs 3 & 4 \* Assigned: October 8, 2019 \* Due: October 29, 2019 \* File Name: FileSys.cpp \* Description: This file defines the class member functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  **#include <iostream>** **#include <string>** **#include <vector>** **#include <sstream>** **#include <fstream>** **#include <sstream>** **#include <iomanip>** **#include <stdlib.h>**  **#include "Sdisk.h"** **#include "FileSys.h"**  **using** **namespace** std;  FileSys::FileSys(string diskName, **int** numberOfBlocks, **int** blockSize): Sdisk(diskName, numberOfBlocks, blockSize) {  rootSize = blockSize / 13;  fatSize = 4 \* numberOfBlocks / blockSize + 1;  string buffer;  getBlock(1, buffer);  **if** (buffer[0] == '#') *//No file system*  {  cout << "No file system found" << endl;  *//Creating ROOT directory*  **for** (**int** i = 0; i < **this**->rootSize; i++)  {  fileName.push\_back("xxxxxxxx");  firstBlock.push\_back(0);  }   fat.push\_back(2 + fatSize);  fat.push\_back(-1);  **for** (**int** i = 1; i <= fatSize; i++)  {  fat.push\_back(-1);  }  **for** (**int** i = fatSize + 2; i < getNumberOfBlocks(); i++)  {  fat.push\_back(i + 1);  }  fat.push\_back(0);  fsSynch();  }  **else** *//File system exists*  {  cout << "Reading existing file system..." << endl;  *//Read-in ROOT directory*  string readBuffer, file;  **int** block;  istringstream inStream;  getBlock(1, readBuffer);  inStream.str(readBuffer);   **for** (**int** i = 0; i < rootSize; i++)  {  inStream >> file >> block;  fileName.push\_back(file);  firstBlock.push\_back(block);  }  *//Read-in FAT directory*  istringstream inStream2;  buffer.clear();  **for** (**int** i = 0; i < fatSize; i++)  {  string b;  getBlock(2 + i, b);  buffer = buffer + b;  }  inStream2.str(buffer);  **int** toInt;  **for** (**int** i = 0; i < numberOfBlocks; i++)  {  inStream2 >> toInt;  fat.push\_back(toInt);  }  fsSynch();  }  fsSynch(); }  vector<string> FileSys::block(string s, **int** b) {  *// s IS buffer*  *// b IS blockSize, A PRIVATE DATA MEMBER*  *// RETURNS A VECTOR OF STRING(S)*  *// EACH STRING HAS b BYTES*   vector<string> blocks;  **int** numberOfBlocks = 0; *// numberOfBlocks IS A PRIVATE DATA MEMBER*    *// SETS numberOfBlocks ACCORDING TO THE BUFFER SIZE OF, s,*  *// THAT HAS BEEN PASSED-IN AS A PARAMETER*  **if**(s.size() % b == 0)  {  numberOfBlocks = s.size() / b;  }  **else**  {  numberOfBlocks = s.size() / b + 1;  }   *// AFTER SETTING numberOfBlocks, THIS LOOP SETS EACH BLOCK WITH*  *// A STRING FROM THE BUFFER*  **for** (**int** i = 0; i < numberOfBlocks; i++)  {  *// b \* i EQUALS A POSITION THAT BEGINS AT ONE BLOCK.*  *// b IS THE BLOCKSIZE OR NUMBER OF BYTES OR NUMBER OF CHARACTERS*  *// AS i ITERATES UP, EACH BLOCK WILL HAVE A SUB-STRING.*  blocks.push\_back(s.substr(b \* i, b));  }   **int** lastBlock = blocks.size() - 1;  **for** (**int** i = blocks[lastBlock].length(); i < b; i++)  {  *// APPENDS/CONCATENATES "#" TO THE END OF THE STRING*  *// CONTAINED IN THE LAST INDEX OF THE VECTOR OF STRINGS, blocks.*  blocks[lastBlock] += "#";  }   **return** blocks; }  **int** FileSys::fsSynch() {  *// Synching the ROOT directory to the software disk*  string buffer;  ostringstream outStream;  **for** (**int** i = 0; i < fileName.size(); ++i)  {  outStream << std::left << setfill(' ') << setw(8) << fileName[i] << " " << std::left << setfill(' ') << setw(3) << firstBlock[i] << " ";  }  buffer = outStream.str();  vector<string> blocks = block(buffer, 13);  buffer.clear();  **for** (**int** i = 0; i < blocks.size(); ++i)  {  buffer = buffer + blocks[i];  }  blocks.clear();  putBlock(1, buffer);  buffer.clear();   *// Synching the FAT to the software disk*  ostringstream outStream2;  **for** (**int** i = 0; i < fat.size(); i++)  {  outStream2 << std::left << setfill(' ') << setw(3) << fat[i] << " ";  }  buffer = outStream2.str();  blocks = block(buffer, getBlockSize());  buffer.clear();  outStream2.str("");   **for** (**int** i = 0; i < blocks.size(); ++i)  {   putBlock(2 + i, blocks[i]);  }   **return** 1; }  **int** FileSys::fsClose() {  fsSynch();  **return** 1; }  **int** FileSys::newFile(string file) {  **for** (**int** i = 0; i < rootSize; i++)  {  **if** (fileName[i] == file)  {  cout << "File already exists. " << endl;  **return** 0;  }  }   **for** (**int** i = 0; i < rootSize; i++)  {  **if** (fileName[i] == "xxxxxxxx")  {  fileName[i] = file;  fsSynch();  **return** 1;  }  }  **return** 0; }  **int** FileSys::rmFile(string file) {  **for** (**int** i = 0; i < rootSize; i++)  {  **if** (fileName[i] == file)  {  **if** (firstBlock[i] != 0)  {  cout << "File is not empty." << endl;  **return** 0;  }  **else**  {  fileName[i] = "xxxxxxxx";  fsSynch();  **return** 1;  }  }  }  cout << "File does not exist." << endl;  **return** 0; }  **int** FileSys::getFirstBlock(string file) {  **for** (**int** i = 0; i < fileName.size(); ++i)  {  **if** (fileName[i] == file)  {  **return** firstBlock[i];  }  }  cout << "No such file" << endl;  **return** 0; }  **int** FileSys::addBlock(string file, string block) {  **int** first = getFirstBlock(file);  **if** (first == -1)  {  **return** 0;  }   **int** allocate = fat[0];  **if** (allocate == 0)  {  **return** 0;  }   fat[0] = fat[fat[0]];  fat[allocate] = 0;   **if** (first == 0)  {  **for** (**int** i = 0; i < rootSize; i++)  {  **if** (fileName[i] == file)  {  firstBlock[i] = allocate;  fsSynch();  putBlock(allocate, block);  **return** allocate;  }  }  }  **else**  {  **int** iBlock = first;  **while** (fat[iBlock] != 0)  {  iBlock = fat[iBlock];  }  fat[iBlock] = allocate;  fsSynch();  putBlock(allocate, block);  **return** allocate;  }  fsSynch(); }   **int** FileSys::checkBlock(string file, **int** blockNumber) {  **int** iBlock = getFirstBlock(file);  **while**(iBlock != 0)  {  **if** (iBlock == blockNumber)  {  **return** true;  }  iBlock = fat[iBlock];  }  **return** false; }  **int** FileSys::delBlock(string file, **int** blockNumber) {  **if** (!checkBlock(file, blockNumber))  {  **return** 0;  }   **int** deAllocate = blockNumber;   **if** (blockNumber == getFirstBlock(file))  {  **for** (**int** i = 0; i < fileName.size(); i++)  {  **if** (file == fileName[i])  {  firstBlock[i] = fat[blockNumber];  **break**;  }  }  fat[deAllocate] = fat[0];  fat[0] = deAllocate;  string hashTags;  **for** (**int** i = 0; i < getBlockSize(); i++)  {  hashTags = hashTags + '#';   }  putBlock(deAllocate, hashTags);  fsSynch();  }  **else**  {  **int** iBlock = getFirstBlock(file);  **while** (fat[iBlock] != blockNumber)  {  iBlock = fat[iBlock];  }  *// fat[iBlock] == blockNumber*  fat[iBlock] = fat[blockNumber];  fat[deAllocate] = fat[0];  fat[0] = deAllocate;  string hashTags;  **for** (**int** i = 0; i < getBlockSize(); i++)  {  hashTags = hashTags + '#';   }  putBlock(deAllocate, hashTags);  fsSynch();  } }  **int** FileSys::readBlock(string file, **int** blockNumber, string& buffer) {  **if** (checkBlock(file, blockNumber))  {  getBlock(blockNumber, buffer);  cout << "Block number " << blockNumber << " contains " << '"'<< buffer << '"' << endl;  **return** 1;  }  **else**  {  cout << "No such block number in " << file << endl;  **return** 0;  } }  **int** FileSys::writeBlock(string file, **int** blockNumber, string buffer) {  **if** (checkBlock(file, blockNumber))  {  string hashTags;  **for** (**int** i = 0; i < getBlockSize(); i++)  {  hashTags = hashTags + '#';   }  putBlock(blockNumber, hashTags);  putBlock(blockNumber, buffer);  fsSynch();  **return** 1;  }  **else**  {  **return** 0;  } }  **int** FileSys::nextBlock(string file, **int** blockNumber) {  **if** (checkBlock(file, blockNumber))  {  **return** fat[blockNumber];  }  **else**  {  **return** -1;  } } |

|  |
| --- |
| */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* Programmer: Eugene Kim \* Quarter: Fall 2019 \* Course: CSE 461 Advanced Operating Systems \* Professor: Owen Murphy \* Assignment: Labs 3 & 4 \* Assigned: October 8, 2019 \* Due: October 29, 2019 \* File Name: FileSys.h \* Description: This file defines the class interface \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  **#ifndef FILESYS\_H** **#define FILESYS\_H**  **#include <iostream>** **#include <string>** **#include <vector>** **#include <sstream>** **#include <iomanip>** **#include <stdlib.h>**  **#include "Sdisk.h"**  **class** **FileSys**: **public** Sdisk { **private**:  **int** rootSize;  **int** fatSize;  vector<string> fileName;  vector<**int**> firstBlock;  vector<**int**> fat; **public**:  FileSys(string diskName, **int** numberOfBlocks, **int** blockSize);  **int** **fsSynch**();  **int** **fsClose**();  **int** **newFile**(string file);  **int** **rmFile**(string file);  **int** **getFirstBlock**(string file);  **int** **addBlock**(string file, string block);  **int** **delBlock**(string file, **int** blockNumber);  **int** **checkBlock**(string file, **int** blockNumber);  **int** **readBlock**(string file, **int** blockNumber, string& buffer);  **int** **writeBlock**(string file, **int** blockNumber, string buffer);  **int** **nextBlock**(string file, **int** blockNumber);  vector<string> block(string s, **int** b); };  **#endif** |

|  |
| --- |
| */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* Programmer: Eugene Kim \* Quarter: Fall 2019 \* Course: CSE 461 Advanced Operating Systems \* Professor: Owen Murphy \* Assignment: Labs 3 & 4 \* Assigned: October 8, 2019 \* Due: October 29, 2019 \* File Name: Sdisk.cpp \* Description: This file defines the class member functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  **#include <cstdlib>** **#include <iostream>** **#include <string>**  **#include <fstream>**  **#include "Sdisk.h"**  **using** **namespace** std;  Sdisk::Sdisk(string diskName, **int** numberOfBlocks, **int** blockSize) {  **this**->diskName = diskName;  **this**->numberOfBlocks = numberOfBlocks;  **this**-> blockSize = blockSize;   fstream inFile;  inFile.open(diskName.c\_str(), ios::in | ios::out); *// Allows input (read operations) from a stream and allows output (write operations) to a stream.*  **if** (inFile.good())  {  cout << diskName << " has been opened." << endl;  }  **else**  {  cout << diskName << " has been created."<< endl;  inFile.open(diskName.c\_str(), ios::out);  **for** (**int** i = 0; i < (blockSize \* numberOfBlocks); i++)  {  inFile.put('#');  }  }   inFile.close(); }  **int** Sdisk::getBlock(**int** blockNumber, string& buffer) *// retrieves block blockNumber from the disk and stores the data in the string buffer. It returns an error code of 1 if successful amd 0 otherwise.* {   fstream iFile;  iFile.open(diskName.c\_str(), ios::in | ios::out);  iFile.seekg(blockNumber \* blockSize);  **if** (iFile.good())  {  **for** (**int** i = 0; i < blockSize; i++)  {  buffer += iFile.get();  }   iFile.close();  **return** 1;  }  **else**  {  **return** 0;  } }  **int** Sdisk::putBlock(**int** blockNumber, string buffer) {  fstream outFile;  outFile.open(diskName.c\_str(), ios::in | ios::out);  outFile.seekg(blockNumber \* blockSize);  **if** (outFile.good())  {  **for** (**int** i = 0; i < buffer.length() && i < blockSize; i++)  {  outFile.put(buffer[i]);  }  outFile.close();  buffer.clear();  **return** 1;  }  **else**  {  cout << "Error" << endl;  **return** 0;  } }  **int** Sdisk::getNumberOfBlocks() {  **return** numberOfBlocks; }  **int** Sdisk::getBlockSize() {  **return** blockSize; }  vector<string> Sdisk::block(string s, **int** b) {  *// s IS buffer*  *// b IS blockSize, A PRIVATE DATA MEMBER*  *// RETURNS A VECTOR OF STRING(S)*  *// EACH STRING HAS b BYTES*   vector<string> blocks;  **int** numberOfBlocks = 0; *// numberOfBlocks IS A PRIVATE DATA MEMBER*    *// SETS numberOfBlocks ACCORDING TO THE BUFFER SIZE OF, s,*  *// THAT HAS BEEN PASSED-IN AS A PARAMETER*  **if**(s.size() % b == 0)  {  numberOfBlocks = s.size() / b;  }  **else**  {  numberOfBlocks = s.size() / b + 1;  }   *// AFTER SETTING numberOfBlocks, THIS LOOP SETS EACH BLOCK WITH*  *// A STRING FROM THE BUFFER*  **for** (**int** i = 0; i < numberOfBlocks; i++)  {  *// b \* i EQUALS A POSITION THAT BEGINS AT ONE BLOCK.*  *// b IS THE BLOCKSIZE OR NUMBER OF BYTES OR NUMBER OF CHARACTERS*  *// AS i ITERATES UP, EACH BLOCK WILL HAVE A SUB-STRING.*  blocks.push\_back(s.substr(b \* i, b));  }   **int** lastBlock = blocks.size() - 1;  **for** (**int** i = blocks[lastBlock].length(); i < b; i++)  {  *// APPENDS/CONCATENATES "#" TO THE END OF THE STRING*  *// CONTAINED IN THE LAST INDEX OF THE VECTOR OF STRINGS, blocks.*  blocks[lastBlock] += "#";  }   **return** blocks; } |

|  |
| --- |
| */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* Programmer: Eugene Kim \* Quarter: Fall 2019 \* Course: CSE 461 Advanced Operating Systems \* Professor: Owen Murphy \* Assignment: Labs 3 & 4 \* Assigned: October 8, 2019 \* Due: October 29, 2019 \* File Name: Sdisk.h \* Description: This file defines the class interface \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  **#include <cstdlib>** **#include <iostream>** **#include <cstring>** **#include <vector>**  **using** **namespace** std;  **#ifndef SDISK\_H** **#define SDISK\_H**  **class** **Sdisk** { **private** :  string diskName; *// file name of software-disk*  **int** numberOfBlocks; *// number of blocks on disk*  **int** blockSize; *// block size in bytes*  **public** :  Sdisk(string diskName, **int** numberOfBlocks, **int** blockSize);  **int** **getBlock**(**int** blockNumber, string& buffer); *// retrieves block blockNumber from the disk and stores the data in the string buffer. It returns an error code of 1 if successful amd 0 otherwise.*  **int** **putBlock**(**int** blockNumber, string buffer); *// writes the string buffer to block blockNumber. It returns an error code of 1 if successful and 0 otherwise.*  **int** **getNumberOfBlocks**(); *// accessor function*  **int** **getBlockSize**(); *// accessor function*  vector<string> block(string s, **int** b); };  **#endif** |

|  |
| --- |
| */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* Programmer: Eugene Kim \* Quarter: Fall 2019 \* Course: CSE 461 Advanced Operating Systems \* Professor: Owen Murphy \* Assignment: Labs 3 & 4 \* Assigned: October 8, 2019 \* Due: October 29, 2019 \* File Name: main.cpp \* Description: This program creates a file system \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/* **#include <iostream>**  **#include "Sdisk.h"** **#include "FileSys.h"**  **using** **namespace** std;  *// You can use this to test your Filesys class*   **int** **main**() {  Sdisk **disk1**("disk1", 256, 128);  FileSys **fsys**("disk1", 256, 128);  fsys.newFile("file1");  fsys.newFile("file2");   string bfile1;  string bfile2;   **for** (**int** i = 1; i <= 1024; i++)  {  bfile1 += "1";  }   vector<string> blocks = fsys.block(bfile1, 128);   **int** blockNumber = 0;   **for** (**int** i = 0; i < blocks.size(); i++)  {  blockNumber = fsys.addBlock("file1", blocks[i]);  }   fsys.delBlock("file1", fsys.getFirstBlock("file1"));    **for** (**int** i = 1; i <= 2048; i++)  {  bfile2 += "2";  }   blocks = fsys.block(bfile2, 128);   **for** (**int** i = 0; i < blocks.size(); i++)  {  blockNumber = fsys.addBlock("file2", blocks[i]);  }    fsys.delBlock("file2", blockNumber); } |

|  |
| --- |
| FileSys: main.o Sdisk.o FileSys.o  g++ main.o Sdisk.o FileSys.o -o FileSys  main.o: main.cpp  g++ -c main.cpp  FileSys.o:: FileSys.cpp  g++ -c FileSys.cpp  Sdisk.o: Sdisk.cpp  g++ -c Sdisk.cpp  make clean:  rm \*.o FileSys |

