Національний технічний університет України «Київський політехнічний інститут»

Факультет інформатики і обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №3**

з курсу «СПЗ»

по темі:

«Файлова система(частина 1)»

**Виконав:** студент 4 курсу

групи ІО-11 ФІОТ

Ротенберг О.В

Київ 2014

**Завдання**

  Розробити драйвер файлової системи для пристрою зберігання інформації блокового типу, використовуючи ідею структур даних, описану вище. В якості файлової системи взяти звичайний файл, розмір файлу визначає обсяг пристрою зберігання інформації. Самостійно вибрати розмір блоку, кількість посилань на блоки в дескрипторі файлу, максимальну довжину імені файлу.

Розробити консольну програму, що підтримує наступні команди:

mount - підключити файлову систему, збережену в файлі;

umount - відключити файлову систему, в драйвері не повинно залишитися будь-яких даних про файлову систему;

filestat id - вивести інформацію про дескриптори файлу з вказаним номером;

ls - вивести список файлів, із зазначенням номерів дескрипторів файлів;

create ім'я - створити файл із заданим ім'ям;

open ім'я - відкрити файл з вказаним ім'ям, команда повинна призначити унікальний номер fd, числовий дескриптор для роботи з відкритим файлом;

close fd - закрити раніше відкритий файл, унікальний номер fd більше не повинен бути пов'язаний з файлом;

read fd зсув розмір - прочитати дані з файлу за заданим зміщення вказаного розміру;

write fd зсув розмір - записати дані у файл по заданому зміщення вказаного розміру;

link імя1 імя2 - створити посилання з ім'ям імя2 на існуючий файл з ім'ям імя1;

unlink ім'я - знищити посилання з заданим ім'ям;

truncate ім'я розмір - змінити розмір файлу, якщо розмір файлу збільшується, то неініціалізовані дані дорівнюють нулю

**Виконання роботи**

**Опис ідеї реалізації файлової системи.**

Розроблена файлова систему для пристроїв зберігання інформації блочного типу. Розмір блоку та їх кількість статична. У файловій системі є два типи файлів: звичайні і директорії. Реалізовано однорівневу файлову систему з одного директорією, що містить звичайні файли.

Вміст кожного файлу зберігається в блоках. Для обліку зайнятості блоків використовується біт зайнятості, один біт на один блок. Блоки в структурах файлової системи адресуються за їх порядковими номерами.

Кожен файл, як об'єкт файлової системи, представлений дескриптором. Кількість дескрипторів заздалегідь задається, тому в файловій системі не може бути створено більше певного кількість файлів, навіть якщо є вільне місце. Дескриптор файлу містить таку інформацію: тип файлу (звичайний файл або директорія), кількість посилань на файл, розмір файлу, карта розташування блоків файлу.

Карта розташування блоків має наступну структуру: є кілька прямих посилань на блоки та одне посилання на один блок, що містить прямі посилання на блоки. Номер (позиція) посилання визначає зміщення даних у файлі, посилання нумеруються підряд.

Директорія - це файл, дані якого це масив посилань на файли. Посилання на файл це ім'я файлу і відповідний цьому імені номер дескриптора файлу. Так як ім'я файлу не є частиною дескриптора файлу, то на один файл може бути декілька посилань з різними іменами.

**Описа форматів структур файлової системи**

DataBlock – клас блоку даних.

FDescript – клас дескриптора файла\папки.

FileSystem – клас, реалізуючий файлову систему.

Main – головний клас програми.

**Лістинг розробленої програми**

import java.io.\*;

import java.util.HashMap;

import java.util.Scanner;

public class Main {

public static Scanner scan = new Scanner(System.in);

public static String c;

public static FileSystem fileSystem;

public static FDescript fDescript;

public static void main(String[] args) {

while (true) {

c = scan.nextLine() + " ";

try {

switch (c.substring(0, c.indexOf(" "))) {

case "mount":

if (getNumberOfSpace(c) == 2 && c.length() > 7) {

String firstName = c.substring(c.indexOf(" ") + 1, c.length() - 1);

File a = new File(firstName + ".fs");

if (a.exists()) {

open(firstName);

} else

fileSystem = new FileSystem();

System.out.println("File system mounted.(" + fileSystem.getDataBlocks().length + " blocks)");

}

break;

case "unmount":

if (getNumberOfSpace(c) == 2 && c.length() > 9) {

String firstName = c.substring(c.indexOf(" ") + 1, c.length() - 1);

save(firstName);

}

break;

case "filestat":

if (getNumberOfSpace(c) == 2 && c.length() > 10) {

int id = Integer.parseInt(c.substring(c.indexOf(" ") + 1, c.length() - 1));

FDescript tempDescr = fileSystem.getFileDescriptors().get(id);

System.out.println("(" + tempDescr.getNumberOfLinks() + ") Links: " + tempDescr.getLinks());

}

break;

case "create":

String name = c.substring(c.indexOf(" ") + 1, c.length() - 1);

int fd = getDescriptorByName(name);

if (fileSystem.getFileCount() < fileSystem.getMaxFileCount()) {

fDescript = new FDescript(false, 1, 0);

int n = fileSystem.getFileDescriptors().size();

int numberOfFolderLinks = fileSystem.getFolder().getNumberOfLinks();

if (numberOfFolderLinks < fileSystem.getFolder().getMaxNumberOfLinks() - 1) {

int block = findFirstFreeBlock();

// Add link into folder

fileSystem.getFolder().addLink(block);

// Add descriptor info into block

fileSystem.getDataBlocks()[block].setUsed(true);

fileSystem.getDataBlocks()[block].setData(name, n);

// Add descriptor to List of descriptors

fileSystem.getFileDescriptors().put(n, fDescript);

fileSystem.setFileCount(fileSystem.getFileCount() + 1);

System.out.println("New file created: " + name);

} else {

int blockWithLinks = findFirstFreeBlock();

// Add link into folder

fileSystem.getFolder().addLink(blockWithLinks);

// Create block with links and write link in it

int block = findFirstFreeBlock();

fileSystem.getDataBlocks()[blockWithLinks].setUsed(true);

fileSystem.getDataBlocks()[blockWithLinks].setData(block);

// Create block (witch linked from block with links)

fileSystem.getDataBlocks()[block].setUsed(true);

fileSystem.getDataBlocks()[block].setData(name, n);

// Add descriptor to List of descriptors

fileSystem.getFileDescriptors().put(n, fDescript);

fileSystem.setFileCount(fileSystem.getFileCount() + 1);

System.out.println("New file created: " + c.substring(c.indexOf(" "), c.length() - 1));

}

} else

System.out.println("You can't create more files.");

break;

case "ls":

System.out.println("Files:");

for (int i = 0; i < fileSystem.getFolder().getLinks().size(); i++) {

int adrOfBlock = fileSystem.getFolder().getLinks().get(i);

HashMap<String, Integer> h = fileSystem.getDataBlocks()[adrOfBlock].getDataList();

System.out.println(h);

}

break;

case "open":

name = c.substring(c.indexOf(" ") + 1, c.length() - 1);

fd = getDescriptorByName(name);

if (fd != 999) {

fileSystem.getOpenedFiles().add(fd);

System.out.println("[" + name + "] was opened. FD = " + fd);

} else {

System.out.println("This file doesn't exist in file system.");

}

break;

case "close":

name = c.substring(c.indexOf(" ") + 1, c.length() - 1);

fd = getDescriptorByName(name);

if (fileSystem.getOpenedFiles().contains(fd)) {

fileSystem.getOpenedFiles().remove(fd);

System.out.println("[" + name + "] was closed. FD = " + fd);

} else {

System.out.println("This file is close.");

}

break;

case "read":

String buf1 = c.substring(c.indexOf(" ") + 1, c.length() - 1);

String name2 = buf1.substring(0, buf1.indexOf(" "));

if (getNumberOfSpace(c) == 4 && c.length() > 6) {

int fd1 = getDescriptorByName(name2);

buf1 = buf1.substring(buf1.indexOf(" ") + 1, buf1.length());

int disp1 = Integer.parseInt(buf1.substring(0, buf1.indexOf(" ")));

buf1 = buf1.substring(buf1.indexOf(" ") + 1, buf1.length());

int size1 = Integer.parseInt(buf1);

if (fileSystem.getOpenedFiles().contains(fd1)) {

FDescript d = fileSystem.getFileDescriptors().get(fd1);

int blockSize = fileSystem.getDataBlocks()[0].getMaxBlockSize();

int block = d.getLinks().get(d.getLinks().size() - 1);

int blocksToRead = ((disp1 + size1) + (1 + blockSize)) / blockSize;

String data = "";

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

data += fileSystem.getDataBlocks()[block].getData();

block++;

}

data = data.substring(disp1, size1 + disp1);

System.out.println("Result:\n " + data);

}

}

break;

case "write":

String buf = c.substring(c.indexOf(" ") + 1, c.length() - 1);

String name1 = buf.substring(0, buf.indexOf(" "));

if (getNumberOfSpace(c) == 5 && c.length() > 8) {

int fd1 = getDescriptorByName(name1);

buf = buf.substring(buf.indexOf(" ") + 1, buf.length());

int disp1 = Integer.parseInt(buf.substring(0, buf.indexOf(" ")));

buf = buf.substring(buf.indexOf(" ") + 1, buf.length());

int size1 = Integer.parseInt(buf.substring(0, buf.indexOf(" ")));

buf = buf.substring(buf.indexOf(" ") + 1, buf.length());

String data1 = buf;

if (fileSystem.getOpenedFiles().contains(fd1)) {

if (data1.length() > size1)

data1 = data1.substring(0, size1);

FDescript d = fileSystem.getFileDescriptors().get(fd1);

int block = findFirstFreeBlock();

if (!fileSystem.getFileDescriptors().get(fd1).getLinks().isEmpty())

block = d.getLinks().get(0);

// Count displacement

int blockSize = fileSystem.getDataBlocks()[0].getMaxBlockSize();

int localDisp;

int blocksDisp;

if (disp1 == 0) {

localDisp = 0;

blocksDisp = 0;

} else {

localDisp = disp1 % blockSize;

blocksDisp = disp1 / blockSize;

}

int spaceWeHave = fileSystem.getDataBlocks().length - block;

if (spaceWeHave < blocksDisp) {

d.addLink(block);

System.out.println("Need more data blocks. Displacement will be 0.");

} else {

block += blocksDisp;

d.addLink(block);

}

// Count local displacement && size

int dataSizeInBlocks = (((localDisp + size1) + (blockSize - 1)) / blockSize);

int spaceWeNeed = block + dataSizeInBlocks;

if (spaceWeNeed > spaceWeHave) {

localDisp = 0;

System.out.println("Need more data blocks. Local displacement will be 0.");

}

d.setSize(dataSizeInBlocks);

d.setFolder(false);

// Create mass with data to write in blocks

String spaces = "";

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

spaces += " ";

}

data1 = spaces + data1;

for (int i = 0; i < blockSize - localDisp + 1; i++) {

data1 += " ";

}

// Write in blocks

int curBlock = block;

for (int i = 0; data1.length() >= blockSize; i++) {

String dataToWrite = data1.substring(0, blockSize);

data1 = data1.substring(blockSize, data1.length());

char[] dToWr = dataToWrite.toCharArray();

char[] dWeHaveInCurBlock = fileSystem.getDataBlocks()[curBlock].getData().toCharArray();

fileSystem.getDataBlocks()[curBlock].setUsed(true);

char[] dToWrInCurBlock = new char[blockSize];

for (int j = 0; j < blockSize; j++) {

if (dToWr[j] == ' ') {

dToWrInCurBlock[j] = dWeHaveInCurBlock[j];

} else {

dToWrInCurBlock[j] = dToWr[j];

}

}

fileSystem.getDataBlocks()[curBlock].setData(new String(dToWrInCurBlock));

curBlock++;

d.addLink(curBlock);

}

d.getLinks().remove(d.getLinks().size() - 1);

fileSystem.getFileDescriptors().put(fd1, d);

} else

System.out.println("This file is close.");

}

break;

case "link":

String buf3 = c.substring(c.indexOf(" ") + 1, c.length() - 1);

if (getNumberOfSpace(c) == 3 && c.length() > 6) {

String firstName = buf3.substring(0, buf3.indexOf(" "));

String secodName = buf3.substring(buf3.indexOf(" ") + 1, buf3.length());

int block = findFirstFreeBlock();

fileSystem.getFolder().getLinks().add(block);

// Add descriptor info into block

int descr = getDescriptorByName(firstName);

fileSystem.getDataBlocks()[block].setUsed(true);

fileSystem.getDataBlocks()[block].setData(secodName, descr);

fileSystem.getFileDescriptors().get(descr).

setSize(fileSystem.getFileDescriptors().get(descr).getSize() + 1);

System.out.println(firstName + " linked with " + secodName);

}

break;

case "unlink":

if (getNumberOfSpace(c) == 2 && c.length() > 8) {

String firstName = c.substring(c.indexOf(" ") + 1, c.length() - 1);

int descr = getDescriptorByName(firstName);

fileSystem.getFileDescriptors().get(descr).setSize(fileSystem.getFileDescriptors().

get(descr).getSize() - 1);

for (int i = 0; i < fileSystem.getFolder().getLinks().size(); i++) {

if (fileSystem.getDataBlocks()[fileSystem.getFolder().getLinks().get(i)].

getDataList().containsKey(firstName)) {

fileSystem.getDataBlocks()[fileSystem.getFolder().getLinks().get(i)].

getDataList().remove(firstName);

fileSystem.getFolder().getLinks().remove(i);

break;

}

}

System.out.println(firstName + " is unlinked.");

}

break;

case "truncate":

String buf2 = c.substring(c.indexOf(" ") + 1, c.length() - 1);

String name3 = buf2.substring(0, buf2.indexOf(" "));

if (getNumberOfSpace(c) == 3 && c.length() > 9) {

int fd1 = getDescriptorByName(name3);

buf2 = buf2.substring(buf2.indexOf(" ") + 1, buf2.length());

int size1 = Integer.parseInt(buf2);

int blockSize = fileSystem.getDataBlocks()[0].getMaxBlockSize();

FDescript d = fileSystem.getFileDescriptors().get(fd1);

int block = d.getLinks().get(d.getLinks().size() - 1);

if (d.getLinks().size() > 0) {

block = d.getLinks().get(0);

}

String fileData = "";

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

fileData += fileSystem.getDataBlocks()[block + i].getData();

}

if (fileData.length() < size1) {

for (int i = 0; i < size1 - fileData.length(); i++) {

fileData += "0";

}

} else {

int fd\_l = fileData.length();

fileData = fileData.substring(0, size1);

for (int i = 0; i < fd\_l - size1; i++) {

fileData += " ";

}

}

// Write in blocks

int dataSizeInBlocks = (fileData.length() + (blockSize + 1)) / blockSize;

if (dataSizeInBlocks > 1) {

int curBlock = block;

for (int i = 0; fileData.length() >= blockSize; i++) {

String dataToWrite = fileData.substring(0, blockSize);

fileData = fileData.substring(blockSize, fileData.length());

char[] dToWr = dataToWrite.toCharArray();

fileSystem.getDataBlocks()[curBlock].setUsed(true);

char[] dToWrInCurBlock = new char[blockSize];

for (int j = 0; j < blockSize; j++) {

dToWrInCurBlock[j] = dToWr[j];

}

fileSystem.getDataBlocks()[curBlock].setData(new String(dToWrInCurBlock));

curBlock++;

}

} else {

fileSystem.getDataBlocks()[block].setData(fileData);

}

}

break;

case "print":

for (int i = 0; i < fileSystem.getDataBlocks().length; i++) {

System.out.println("[" + i + "] " + fileSystem.getDataBlocks()[i].getData());

}

break;

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

private static int getNumberOfSpace(String name) {

int n = 0;

for (char a : name.toCharArray()) {

if (a == ' ')

n++;

}

return n;

}

private static int getDescriptorByName(String name) {

int descr = 999;

for (int i = 0; i < fileSystem.getFolder().getLinks().size(); i++) {

int adrOfBlock = fileSystem.getFolder().getLinks().get(i);

HashMap<String, Integer> h = fileSystem.getDataBlocks()[adrOfBlock].getDataList();

if (h.containsKey(name)) {

descr = h.get(name);

break;

}

}

return descr;

}

private static int findFirstFreeBlock() {

int c = 0;

for (int i = 0; i < fileSystem.getDataBlocks().length; i++) {

if (fileSystem.getDataBlocks()[i].isUsed()) {

c++;

} else {

break;

}

}

return c;

}

public static void open(String s) {

File myFile = new File(s);

FileInputStream fileIn;

try {

fileIn = new FileInputStream(myFile.getAbsolutePath() + ".fs");

ObjectInputStream in1 = new ObjectInputStream(fileIn);

fileSystem = (FileSystem) in1.readObject();

in1.close();

fileIn.close();

} catch (Exception e) {

e.printStackTrace();

}

}

public static void save(String s) {

File myFile = new File(s);

FileOutputStream fileOut;

try {

fileOut = new FileOutputStream(myFile.getPath() + ".fs");

ObjectOutputStream out = new ObjectOutputStream(fileOut);

out.writeObject(fileSystem);

out.close();

fileOut.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

public class FileSystem implements Serializable {

private final int numberOfBlocks = 8;

private final int maxFileCount = 4;

private DataBlock[] dataBlocks;

private FDescript folder;

private int fileCount;

private HashMap<Integer, FDescript> fileDescriptors;

private ArrayList<Integer> openedFiles;

public FileSystem() {

folder = new FDescript(true, 1, 0);

dataBlocks = new DataBlock[numberOfBlocks];

fileCount = 0;

for (int i = 0; i < dataBlocks.length; i++) {

dataBlocks[i] = new DataBlock();

}

fileDescriptors = new HashMap<>();

openedFiles = new ArrayList<>();

}

public ArrayList<Integer> getOpenedFiles() {

return openedFiles; }

public HashMap<Integer, FDescript> getFileDescriptors() {

return fileDescriptors; }

public DataBlock[] getDataBlocks() {

return dataBlocks; }

public int getMaxFileCount() {

return maxFileCount; }

public FDescript getFolder() {

return folder; }

public int getFileCount() {

return fileCount; }

public void setFileCount(int fileCount) {

this.fileCount = fileCount; }

}

public class FDescript implements Serializable {

private final int maxNumberOfLinks = 5;

private boolean isFolder;

private int size;

private ArrayList<Integer> links;

public FDescript(boolean isFolder, int numberOfLinks, int size) {

this.isFolder = isFolder;

this.size = size;

links = new ArrayList<>();

}

public ArrayList<Integer> getLinks() {

return links; }

public void addLink(int link) {

links.add(link); }

public int getMaxNumberOfLinks() {

return maxNumberOfLinks; }

public boolean isFolder() {

return isFolder; }

public void setFolder(boolean folder) {

isFolder = folder; }

public int getNumberOfLinks() {

return links.size(); }

public int getSize() {

return size; }

public void setSize(int size) {

this.size = size; }

}

public class DataBlock implements Serializable{

private final int maxBlockSize = 8;

private boolean isUsed;

private String data;

private ArrayList<Integer> links;

private HashMap<String, Integer> listOfLinks;

public DataBlock() {

listOfLinks = new HashMap<>();

links = new ArrayList<>();

data = "";

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

data += " ";

}

}

public String getData() {return data; }

public void setData(int id) {

if (links.size() < maxBlockSize) {

links.add(id);

} else {

System.out.println("You can't add more links in this block");

}

}

public void setData(String data) {

this.data = data; }

public ArrayList<Integer> getDataLinks() {

return links; }

public HashMap<String, Integer> getDataList() {

return listOfLinks; }

public void setData(String name, int id) {

if (listOfLinks.size() < maxBlockSize) {

listOfLinks.put(name, id);

} else {System.out.println("You can't add more links in this block"); }

}

public int getMaxBlockSize() {return maxBlockSize; }

public boolean isUsed() {return isUsed; }

public void setUsed(boolean used) {isUsed = used;}

}