Національний технічний університет України «Київський політехнічний інститут»

Факультет інформатики і обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №4**

з курсу «СПЗ»

по темі:

«Файлова система(частина 2)»

**Виконав:** студент 4 курсу

групи ІО-92 ФІОТ

Петрук В.О.

Київ 2012

**Завдання**

  Додати в драйвер файлової системи, розроблений в лабораторній роботі

№ 3, підтримку дерева директорій і символічних посилань. Додати в консольну програму команди для роботи з директоріями і символічними

посиланнями.

  Консольна програма повинна підтримувати шляхові імена файлів в командах, файл шукається по колійному імені починаючи з поточної робочої директорії.

Команди create, open, truncate і cd повинні працювати з файлами на яке вказує ім'я (якщо остання компонента імені файлу це символічне посилання, то необхідно перейти за цим посиланням). Всі інші команди, які працюють з іменами файлів, працюють із заданими файлами по імені (якщо остання компонента імені файлу це символічне посилання, то команда застосовується до самої символічної посилання, а не до файлу на який вказує символьне посилання). Команда link повинна створювати жорсткі посилання тільки для звичайних файлів.

  Додати в консольну програму підтримку наступних команд:

  mkdir ім'я - створити директорію з вказаним ім'ям;

 rmdir ім'я - знищити порожню директорію з вказаним ім'ям (директорія не повинна містити будь-яких посилань, за винятком зумовлених "." І "..");

 cd ім'я - змінити поточну робочу директорію;

 pwd - вивести поточну робочу директорію;

 symlink імя1 імя2 - створити символічне посилання з ім'ям імя2 на шляхове ім'я імя1.

**Опис ідеї реалізації файлової системи.**

В програму були додані команди для роботи з директоріями і символічними посиланнями.

Консольна програма підтримує шляхові імена та символічні посилання в командах (rmdir, cd, symlink).Коренева директорія має фіксований номер дескриптора, а решта директорії мають довільні номери дескрипторів.Будь-яка директорія має два посилання: "." і "..", які вказують на поточну і батьківську директорію. Посилання ".." в кореневій директорії вказує на себе.Ім'я файлу може бути абсолютним або відносним. В кожен момент часу існує поточнадиректорія, яку можна змінювати.

Символічне посилання вказує на файл, але не є ще одним ім'ям файлу. Створення і знищення символічного посилання не призводить до зміни лічильника посилань файлу. Створення і знищення посилань на директорії і звичайні файли не впливають на символічні посилання, можна створювати символічне посилання на неіснуюче ім'я файлу.

**Опис форматів структур файлової системи.**

DataBlock – клас блоку даних.

FDescript – клас дескриптора файла\папки.

FileSystem – клас, реалізуючий файлову систему.

Main – головний клас програми, де все відбувається.

**Лістинг основної частини розробленої програми.**

case "mkdir":

mkdir();

break;

case "rmdir":

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

rmdir(firstName);

break;

case "cd":

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

cd(firstName);

break;

case "pwd":

pwd();

break;

case "symlink":

symlink();

break;

}

}

} catch (Exception e) {

System.out.println("General error.");

e.printStackTrace();

}

}

private static void symlink() {

if (getNumberOfSpace(c) == 3 &&c.length() > 11) {

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

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

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

String name2 = buf.substring(buf.indexOf(" ") + 1, buf.length());

int b = findFirstFreeBlock();

// Add link in current dir descriptor

fileSystem.getFileDescriptors().get(getDescriptorByName(fileSystem.getCurentPoss().

get(fileSystem.getCurentPoss().size() - 1))).addLink(b);

fileSystem.getDataBlocks()[b].getSymbolicLinks().put(name1, name2);

System.out.println("Linked: " + name1 + " -> " + name2);

}

private static void pwd() {

if (c.length() > 3) {

System.out.println("Your possition is:\n" + getCurPath());

System.out.println("Curentdir is: " + fileSystem.getCurentPoss().get(fileSystem.getCurentPoss().size() - 1));

}

private static void cd(String firstName) {

if (getNumberOfSpace(c) == 2 &&c.length() > 4) {

if (firstName.contains("/")) {

// Rewrite in ArrList

ArrayList<String> path = new ArrayList<>();

StringTokenizerst = new StringTokenizer(firstName, "/");

intcurToken = 0;

intct = st.countTokens();

while (st.hasMoreTokens()) {

String key = st.nextToken();

if (!path.contains(key)) {

if (isFolderExist(key) || key.equals("home")) {

path.add(key);

} else {

System.out.println("Dir [" + key + "] does not exist.");

break;

}

}

if (curToken != ct - 1) {

String val = st.nextToken();

if (!path.contains(val)) {

if (isFolderExist(val) || val.equals("home")) {

path.add(val);

} else {

System.out.println("Dir [" + val + "] does not exist.");

break;

}

}

}

curToken += 2;

}

if (path.get(0).equals("home")) {

fileSystem.setCurentPoss(path);

} else {

for (inti = 0; i<path.size(); i++) {

fileSystem.getCurentPoss().add(path.get(i));

}

}

pwd();

} else if (firstName.contains("..")) {

fileSystem.getCurentPoss().remove(fileSystem.getCurentPoss().size() - 1);

System.out.println("Your possition is:\n" + getCurPath());

} else {

String curD = fileSystem.getCurentPoss().get(fileSystem.getCurentPoss().size() - 1);

// Check symlink

if (allSymlinksOffolder(curD).containsKey(firstName)) {

cd(allSymlinksOffolder(curD).get(firstName));

} else {

booleanisFHSF = isFolderHasSubfolder(curD);

if (isFHSF&& fileSystem.getFileDescriptors().containsKey(getDescriptorByName(firstName))) {

fileSystem.getCurentPoss().add(firstName);

System.out.println("Your possition is:\n" + getCurPath());

} else System.out.println("Could not find this dir.");

}

}

}

}

private static HashMap<String, String> allSymlinksOffolder(String curD) {

HashMap<String, String>allSymlinks = new HashMap<>();

FDescript f = fileSystem.getFileDescriptors().get(getDescriptorByName(curD));

for (inti = 0; i<f.getLinks().size(); i++) {

HashMap<String, String>localSymlinks = fileSystem.getDataBlocks()[f.getLinks().get(i)].getSymbolicLinks();

for (String key : localSymlinks.keySet()) {

allSymlinks.put(key, localSymlinks.get(key));

}

}

returnallSymlinks;

}

private static void rmdir(String firstName) {

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

String curD = fileSystem.getCurentPoss().get(fileSystem.getCurentPoss().size() - 1);

if (allSymlinksOffolder(curD).containsKey(firstName)) {

String symbolicLink = allSymlinksOffolder(curD).get(firstName);

int a = symbolicLink.lastIndexOf('/')+1;

int b = symbolicLink.length();

String reaDir = symbolicLink.substring(a, b);

rmdir(reaDir);

} else {

intdescrNumber = getDescriptorByName(firstName);

if (fileSystem.getFileDescriptors().containsKey(descrNumber)) {

FDescriptdF = fileSystem.getFileDescriptors().get(descrNumber);

if (dF.getLinks().size() == 1) {

intdataBlockNumber = -1;

HashMap<String, Integer> h = null;

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

h = fileSystem.getDataBlocks()[i].getDataList();

if (!h.isEmpty())

if (h.containsKey(firstName)) {

dataBlockNumber = i;

break;

}

}

// Get parent dir name

String parentDir = fileSystem.getDataBlocks()[dataBlockNumber].getStaticLinks().get("..");

// Remove link from parent dir descriptor

intdescN = getDescriptorByName(parentDir);

fileSystem.getFileDescriptors().get(descN).getLinks().remove(fileSystem.getFileDescriptors().

get(descN).getLinks().indexOf(dataBlockNumber));

// Clear data block

fileSystem.getDataBlocks()[dataBlockNumber] = new DataBlock();

// Remove descriptor

fileSystem.getFileDescriptors().remove(getDescriptorByName(firstName));

// Remove from curPoss

if (fileSystem.getCurentPoss().contains(firstName))

fileSystem.getCurentPoss().remove(fileSystem.getCurentPoss().indexOf(firstName));

System.out.println("Dir [" + firstName + "] was removed.");

} else System.out.println("This dir does not empty.");

} else System.out.println("Could not find this dir.");

}

} else

System.out.println("You enter wrong command.");

}

private static void mkdir() {

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

FDescript f0 = new FDescript(true, 1, 0);

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

int b = findFirstFreeBlock();

if (fileSystem.getCurentPoss().size() == 0) {

// Go to 'home' dir

fileSystem.getCurentPoss().add("home");

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

fileSystem.getDataBlocks()[b].setData("home", 0);

// Add '..' and '.'

fileSystem.getDataBlocks()[b].getStaticLinks().put("..", "home");

fileSystem.getDataBlocks()[b].getStaticLinks().put(".", "home");

f0.addLink(b);

// Add to descriptors list

fileSystem.setRootFolder(f0);

fileSystem.getFileDescriptors().put(0, f0);

System.out.println("Dir [home] was created.");

} else {

String curDir = fileSystem.getCurentPoss().get(fileSystem.getCurentPoss().size() - 1);

// Add link on a block (new dir from current dir)

fileSystem.getFileDescriptors().get(getDescriptorByName(curDir)).addLink(b);

// Set current poss to this dir

fileSystem.getCurentPoss().add(newDir);

// Write in block new dir name and descriptor number

intdesctNumber = fileSystem.getFileDescriptors().size();

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

fileSystem.getDataBlocks()[b].setData(newDir, desctNumber);

// Add '..' and '.'

fileSystem.getDataBlocks()[b].getStaticLinks().put("..", curDir);

fileSystem.getDataBlocks()[b].getStaticLinks().put(".", newDir);

f0.addLink(b);

// Add to descriptor list

fileSystem.getFileDescriptors().put(desctNumber, f0);

System.out.println("Dir [" + newDir + "] was created.");

}

} else

System.out.println("You enter wrong command.");

}

private static booleanisFolderHasSubfolder(String firstName) {

booleanisHas = false;

String curDir = fileSystem.getCurentPoss().get(fileSystem.getCurentPoss().size() - 1);

ArrayList<Integer> l = fileSystem.getFileDescriptors().get(getDescriptorByName(curDir)).getLinks();

for (inti = 0; i<l.size(); i++) {

if (fileSystem.getDataBlocks()[l.get(i)].getDataList().containsKey(firstName)) {

isHas = true;

break;

}

}

returnisHas;

}

public static String getCurPath() {

String curPath = "/";

for (inti = 0; i<fileSystem.getCurentPoss().size(); i++) {

curPath += fileSystem.getCurentPoss().get(i) + "/";

}

returncurPath;

}

public static booleanisFolderExist(String key) {

booleanfolderExist = false;

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

if (fileSystem.getDataBlocks()[i].getStaticLinks().containsValue(key)) {

folderExist = true;

break;

}

}

returnfolderExist;

}