МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»

Кафедра

обчислювальної техніки

**ЛАБОРАТОРНА РОБОТА 3**

з дисципліни

“Операційні Системи”

Файловая система (часть 1)

Студента 3 курсу

групи ІП-53

Сулима О.С.

Перевірив: Сімоненко А.В.

Київ 2017

**Завдання:**

Розробити драйвер файлової системи для пристрою зберігання інформації

блочного типу, використовуючи ідею структур даних, описану вище. В якості

файлової системи взяти звичайний файл, розмір файлу визначає обсяг

пристрої зберігання інформації. Самостійно вибрати розмір блоку,

кількість посилань на блоки в дескріторе файлу, максимальну довжину імені

файлу.

Розробити консольну програму, яка підтримує наступні команди:

* mount - підключити файлову систему, збережену в файлі;
* umount - відключити файлову систему;
* filestat id - вивести інфо про дескрипторі файлу з номером;
* ls - вивести список файлів, із зазначенням номерів дескрипторів файлів;
* create ім'я - створити файл з 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 зміщення розмір - прочитати дані з файлу за заданим

зміщення зазначеного розміру;

* write fd зміщення розмір - записати дані в файл за заданим

зміщення зазначеного 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.

**Лістинг:**

// Lab3.cpp : Defines the entry point for the console application.

//

#include <vector>

#include <map>

#include <iostream>

#include <conio.h>

using namespace std;

#define DESC\_COUNT 100

struct File

{

char\* Name;

};

struct Directory

{

char\* Name;

vector<File\*> files;

} dir;

struct Block

{

char content;

};

struct MapBlocks

{

vector<Block\*> links;

};

struct Descriptor

{

File\* file = nullptr;

Directory\* directory = nullptr;

MapBlocks\* mapBlock = nullptr;

int countLinks = 0;

int sizeOfFile = 0;

};

void create(char\* name, vector<Descriptor>\* descriptors, map<char\*, int>\* descFiles);

void Is(map<char\*, int> descFiles);

int open(char\* name, map<char\*, int> descFiles);

void close(int\* fd);

void read(int fd, int arrow, int size, vector<Descriptor> descriptors);

void write(int fd, int arrow, int size, char\* content, map<char\*, int>\* descFiles, vector<Descriptor>\* descriptors);

Descriptor\* link(char\* name, Descriptor\* lnk, map<char\*, int> descFiles, vector<Descriptor>\* descriptors);

void unlink(Descriptor\* lnk);

void truncate(char\* name, int size, map<char\*, int>\* descFiles, vector<Descriptor>\* descriptors);

void fileStat(int fd, vector<Descriptor> descriptors);

void mount(vector<Descriptor>\* descriptors, map<char\*, int>\* descFiles);

void umount(vector<Descriptor>\* descriptors, map<char\*, int>\* descFiles);

int main()

{

vector<Descriptor> descriptors;

map<char\*, int> descFiles;

create("first", &descriptors, &descFiles);

umount(&descriptors, &descFiles);

mount(&descriptors, &descFiles);

int fd = open("first", descFiles);

write(fd, 0, 3, "teg", &descFiles, &descriptors);

read(fd, 0, 3, descriptors);

Descriptor\* descLink = new Descriptor();

descLink = link("first", descLink, descFiles, &descriptors);

unlink(descLink);

fileStat(fd, descriptors);

Is(descFiles);

close(&fd);

\_getch();

return 0;

}

void create(char\* name, vector<Descriptor>\* descriptors, map<char\*, int>\* descFiles)

{

Descriptor d;

File\* f = new File();

MapBlocks\* mapBlock = new MapBlocks();

f->Name = name;

d.countLinks = 0;

d.sizeOfFile = 0;

d.file = f;

d.mapBlock = mapBlock;

descriptors->push\_back(d);

descFiles->insert(pair<char\*, int>(name, descriptors->size() - 1));

}

void Is(map<char\*, int> descFiles)

{

for (auto i = descFiles.begin(); i != descFiles.end(); i++)

{

cout << i->first << " " << i->second << endl;

}

}

int open(char\* name, map<char\*, int> descFiles)

{

return descFiles.find(name)->second;

}

void close(int\* fd)

{

\*fd = -1;

}

void read(int fd, int arrow, int size, vector<Descriptor> descriptors)

{

if (arrow >= 0 && arrow < size && size <= descriptors[fd].mapBlock->links.size() - arrow)

{

for (int i = arrow; i < size; i++)

{

cout << descriptors[fd].mapBlock->links.at(i)->content;

}

cout << endl;

}

}

void write(int fd, int arrow, int size, char\* content, map<char\*, int>\* descFiles, vector<Descriptor>\* descriptors)

{

if (size + arrow > descriptors->at(fd).sizeOfFile)

{

truncate(descriptors->at(fd).file->Name, size + arrow - descriptors->at(fd).sizeOfFile, descFiles, descriptors);

}

for (int i = arrow; i < size; i++)

{

descriptors->at(fd).mapBlock->links[i]->content = content[i - arrow];

}

}

Descriptor\* link(char\* name, Descriptor\* lnk, map<char\*, int> descFiles, vector<Descriptor>\* descriptors)

{

lnk = &descriptors->at(descFiles.find(name)->second);

lnk->countLinks++;

return lnk;

}

void unlink(Descriptor\* lnk)

{

lnk->countLinks--;

lnk = nullptr;

}

void truncate(char\* name, int size, map<char\*, int>\* descFiles, vector<Descriptor>\* descriptors)

{

if (descriptors->at(descFiles->find(name)->second).sizeOfFile < size)

{

for (int i = descriptors->at(descFiles->find(name)->second).sizeOfFile; i < size; i++)

{

Block\* blck = new Block();

descriptors->at(descFiles->find(name)->second).mapBlock->links.push\_back(blck);

}

descriptors->at(descFiles->find(name)->second).sizeOfFile = size;

return;

}

if (descriptors->at(descFiles->find(name)->second).sizeOfFile > size)

{

for (int i = size; i < descriptors->at(descFiles->find(name)->second).sizeOfFile; i++)

{

descriptors->at(descFiles->find(name)->second).mapBlock->links.pop\_back();

}

descriptors->at(descFiles->find(name)->second).sizeOfFile = size;

return;

}

}

void mount(vector<Descriptor>\* descriptors, map<char\*, int>\* descFiles)

{

FILE \*f;

f = fopen("input.dot", "rb");

Descriptor temp;

while (fread(&temp, sizeof(Descriptor), 1, f)) {

descriptors->push\_back(temp);

descFiles->insert(pair<char\*, int>(temp.file->Name, descriptors->size() - 1));

};

fclose(f);

}

void umount(vector<Descriptor>\* descriptors, map<char\*, int>\* descFiles)

{

FILE \*f;

f = fopen("input.dot", "wb");

for (int i = 0; i < descriptors->size(); i++)

{

fwrite(&descriptors->at(i), sizeof(Descriptor), 1, f);

}

descriptors->clear();

descFiles->clear();

fclose(f);

}

void fileStat(int fd, vector<Descriptor> descriptors)

{

Descriptor d = descriptors[fd];

char\* type;

if (d.file != nullptr)

{

type = "File";

}

else

{

type = "Directory";

}

cout << "==================================" << endl;

cout << "Number: " << fd << " Type file: " << type << " Count links: " << d.countLinks << " Size of file: " << d.sizeOfFile << endl;

cout << "==================================" << endl;

}