***Коваленко Сергей***

***Группа ИВ-63***

***Операционные системы***

***Лабораторная работа № 3***

***Драйвер файловой системы JFFS***

***Описание.***

*Драйвер файловой системы спроектирован с использованием обьектно-ориентированного подхода и структур памяти. Структура драйвера монолитна, с выделением функциональности по сериализации в методы моделей страниц физической памяти.*

***Структура драйвера.***

***1.*** *Дескриптор файлового пространства. Содержит данные о размере и количестве страниц.*

***2.*** *Таблица дескрипторов страниц физической памяти. Объем составляет 20 байт. На диске занимает 16 байт (стандартный размер служебных данных для этой файловой системы). В состав входят: обьем и кол-во страниц в пространстве, версия (кол-во перезаписей страницы), количество команд, помещенных на страницу.*

***3.*** *Таблица файлов и файловых дескрипторов. Используется для манипуляций с файлами и проверки актуальности команд. Содержит виртуальные отображения файлов (в оперативной памяти) позволяющие быстрое чтение файлов.*

***4.*** *Списки команд. Представляют собой протоколы модификации дискового пространства. Используются для восстановления состояния файловой системы и ее виртуального отображения. При монтировании формируют собой список команд, отсортированный по идентификатору, то есть по времени применения к файловой системе. При отключении список новых команд переписывается на страницы с наименьшей версией.*

***5.*** *Список областей изменений в файлах, присутствующих в файловой системе.*

***Структура ядра драйвера.***

*Отдельный интерес представляет собой структура ядра драйвера.*

*В ее состав включены два набора функций автоматов управления. Один из них работает непосредственно со структурами памяти в драйвере. Он является интерпретатором набора команд для файловой системы. Управляется вектором функций.*

*Управляющим для него является второй автомат, работающий непосредственно с функциями ввода. Его задача – сформировать пакет команды, состоящий из идентификатора команды, ее кода и дополнительной информации, представленной байтовым массивом. В таком же виде команды хранятся на страницах файловой системы.*

***Программный код:***

#ifndef \_SPACE

#define \_SPACE

class SpaceDescr

{

public:

//char\* path;

unsigned int pageSize;

unsigned int pageAmount;

};

#endif //\_SPACE

#ifndef \_PDT

#define \_PDT

#include <stdio.h>

#include "Command.h"

class PageDescr

{

public:

unsigned int version;

unsigned int records;

unsigned int activeR;

unsigned int pageSize;

unsigned int pageAmount;

Command\* rchain;

unsigned int\* passive;

//PageDescr\* next;

void serialize(FILE\* dst);

void deserialize(FILE\* dst);

};

#endif //\_PDT

#ifndef \_FDT

#define \_FDT

class FileDescr

{

public:

char\* linkName;

unsigned int id;

unsigned int len;

char\* field;

FileDescr\* next;

};

#endif //\_FDT

#ifndef \_COMMAND

#define \_COMMAND

class Command

{

public:

unsigned int id;

unsigned int code;

char\* data;

Command\* next;

};

#endif //\_COMMAND

class Change

{

public:

//unsigned int fId;

unsigned int offset;

unsigned int amount;

Change\* next;

};

class ChangeChain

{

public:

Change\* ch;

unsigned int fId;

unsigned int cId;

ChangeChain\* next;

};

#ifndef \_JFFS\_DRIVER

#define \_JFFS\_DRIVER

#include <stdlib.h>

#include "Space.h"

#include "PDT.h"

#include "FDT.h"

#include "Command.h"

#include "Changes.h"

class JFFSdriver

{

private:

static bool go;

void (JFFSdriver::\*actions[13])(char\*);

void (JFFSdriver::\*inputActions[13])();

//datastructs

static unsigned int cId;

static FILE\* fs;

static SpaceDescr sd;

static Command\* cm;

static Command\* ncm;

static PageDescr\* pdt;

static FileDescr\* fdt;

// methods of input commands

void inputFormat();

void inputMount();

void inputUnmount();

void inputBlockStat();

void inputCreate();//

void inputUnlink();//

void inputList();

void inputOpen();

void inputClose();

void inputRead();

void inputWrite();//

void inputAddName();//

void inputTruncate();//

// operations for executing with file system and files (interpreter)

void format(char\* data);

void mount(char\* data);

void unmount(char\* data);

void blockStat(char\* data);

void create(char\* data);

void unlink(char\* data);

void list(char\* data);

void open(char\* data);

void close(char\* data);

void read(char\* data);

void write(char\* data);

void addName(char\* data);

void truncate(char\* data);

public:

static ChangeChain\* chgs;

JFFSdriver();

static void setReady();

void commandCycle();

};

#endif //\_JFFS\_DRIVER

#include "PDT.h"

void PageDescr::serialize(FILE \*dst)

{

//запись в файл размера страницы, кол-ва страниц в фс, версии даной страницы, количества записей на ней

fprintf(dst, "%u", &(this->pageSize));

fprintf(dst, "%u", &(this->pageAmount));

fprintf(dst, "%u", &(this->version));

fprintf(dst, "%u", &(this->records));

int wr = this->pageSize;//счетчик оставшегося места на странице

char t = 0;

while (rchain != NULL)//идем по списку команд, которые есть на странице

{

Command\* curr = rchain;//записываем для каждой команды ид, коп, данные команды

fprintf(dst, "%u", &(curr->id));

fprintf(dst, "%u", &(curr->code));

fprintf(dst, "%c", &(curr->data[0]));

for(int i = 1; i < (curr->data[0]); i++)

fprintf(dst, "%c", &(curr->data[i]));

wr -= (8 + curr->data[0]);//отнимаем от счетчика оставшегося места обьем записи о команде

rchain = curr->next;

delete curr;//убиваем записанную команду

}

for (int i = 0; i < wr; i++)

fprintf(dst, "%c", &t);//глушим остаток места на странице нулями

}

void PageDescr::deserialize(FILE \*dst)

{//читаем размер страницы, их кол-во, версию, кол-во записей

fscanf(dst, "%u", &(this->pageSize));

fscanf(dst, "%u", &(this->pageAmount));

fscanf(dst, "%u", &(this->version));

fscanf(dst, "%u", &(this->records));

this->rchain = NULL;

unsigned int wr = this->pageSize;

for (unsigned int i = 0; i < this->records; i++)

{//newer commands written first

Command\* nc = new Command();

char t;//воссоздаем список записей команд

fscanf(dst, "%u", &(nc->id));

fscanf(dst, "%u", &(nc->code));

fscanf(dst, "%c", &t);

nc->data = new char[t];

for (char a = 1; a < t; a++)

fscanf(dst, "%c", &(nc->data[a]));

nc->data[0] = t;//читая ИД, коп, данные команды

nc->next = this->rchain;//добавляем в цепочку команд страницы

this->rchain = nc;

wr -= (t + 8);

}

fseek(dst, wr, SEEK\_CUR);

}

#include <stdio.h>

#include <string.h>

#include <conio.h>

#include "JFFSdriver.h"

bool JFFSdriver::go;

unsigned int JFFSdriver::cId;

//void (JFFSdriver::\*actions[])();

FILE\* JFFSdriver::fs;

SpaceDescr JFFSdriver::sd;

PageDescr\* JFFSdriver::pdt;

FileDescr\* JFFSdriver::fdt;

Command\* JFFSdriver::cm;

Command\* JFFSdriver::ncm;

ChangeChain\* JFFSdriver::chgs;

//emulator actions

void JFFSdriver::format(char \*data)

{

sd.pageSize = \*((unsigned int\*)(((unsigned long)data) + 1));//читаем обьем страницы из данных команды

sd.pageAmount = \*((unsigned int\*)(((unsigned long)data) + 5));//читаем кол-во страниц из данных команды

cm = NULL;

fdt = NULL;

pdt = new PageDescr[sd.pageAmount];//инит массива дескрипторов страниц

fs = fopen(((char\*)(((unsigned long) data) + 9)), "w");//открытие файла, в котором будет записана фс

if (fs != NULL)

for(unsigned int i = 0; i < sd.pageAmount; i++)

{

pdt[i].activeR = 0;//инит каждого дескриптора страницы - кол-во активных команд

pdt[i].pageAmount = sd.pageAmount; //кол-во страниц

pdt[i].pageSize = sd.pageSize; //обьем страниц

pdt[i].records = 0; //кол-во записей, версия, пустой список команд, запуск сериализации

pdt[i].version = 0;

pdt[i].rchain = NULL;

pdt[i].serialize(fs);

}

fclose(fs);//закрытие файла и очищение статических полей драйвера

sd.pageAmount = 0;

sd.pageSize = 0;

delete[] pdt;

fs = NULL;

}

void JFFSdriver::mount(char \*data)

{

fs = fopen(((char\*)(((unsigned long) data) + 1)), "r");

if (fs != NULL)

{

fscanf(fs, "%u", &(sd.pageSize));//считаем из файла кол-во и размер страниц

fscanf(fs, "%u", &(sd.pageAmount));

fseek(fs, -8, SEEK\_CUR);

this->pdt = new PageDescr[sd.pageAmount];//развертывание и инит таблицы дескрипторов страниц

for(unsigned int i = 0; i < sd.pageAmount; i++)

{

pdt[i].deserialize(fs);

}

// form common command list

Command\*\* pcl = new Command\*[sd.pageAmount];

unsigned int cSum = 0;

for (unsigned int i = 0; i < sd.pageAmount; i++)

{

cSum += pdt[i].records;

pcl[i] = pdt[i].rchain;

}//подготовка к слитию списков команд каждой стрницы в общий команд-лист

Command\* mIdCom = NULL;

Command\* nc = NULL;

for (unsigned int j = 0; j < cSum; j++)

{

unsigned int pdtCounter = 0;

while(mIdCom == NULL)

{

if (pcl[pdtCounter] != NULL)

mIdCom = pcl[pdtCounter];

pdtCounter++;

}

for (unsigned int i = pdtCounter; i < sd.pageAmount; i++)

if (pcl[pdtCounter]->id < mIdCom->id)

mIdCom = pcl[pdtCounter];

nc = new Command();

nc->code = mIdCom->code;

nc->id = mIdCom->id;

nc->next = cm;

nc->data = new char[mIdCom->data[0]];

for (char i = 0; i < mIdCom->data[0]; i++)

nc->data[i] = mIdCom->data[i];

}//слитие команд в список в порядке возрастания ид

// interpret common command list

Command\* curr = cm;

while (curr != NULL)//в списке произведем интерпретацию собранного команд-листа

{

(this->\*actions[curr->code])(curr->data);

curr = curr->next;

}

}

fclose(fs);

}

void JFFSdriver::unmount(char\* data)

{

bool writeNeed = false;

if (ncm != NULL)

writeNeed = true;

while (ncm != NULL)

{

// select page

unsigned int rwrPage = 0;

for (unsigned int i = 1; i < sd.pageAmount; i++)

{

if (pdt[rwrPage].version > pdt[i].version)

rwrPage = i;

}//выберем страницу с наименьшим номером версии

// select passive commands

Command\* cmnds = pdt[rwrPage].rchain;

//if (cmnds == NULL)

//{

bool\* actual = new bool[pdt[rwrPage].records];//актуальны ли команды (масив)

Command\* curr = cmnds;

unsigned int currActual = 0;

char\* isActual = NULL;

FileDescr\* cf;

unsigned int dFId;

bool deleted;

Command\* c1;

while (curr != NULL)//проверим актуальность всех приведенных (модифицирующих) команд

{

switch (curr->code)

{

case 4://check create node

deleted = true;//не актуален, если файла нет

dFId = \*((unsigned int\*)(&(curr->data[1])));

cf = fdt;

while ((cf != NULL)&&(deleted))

{

if(cf->id == dFId)

deleted = false;

cf = cf->next;

}

if (deleted)

actual[currActual] = false;

else

actual[currActual] = true;

break;

case 5: // check unlink node

c1 = cm;//неактуален, если есть пара (комана создания) и нет ссылок.

actual[currActual] = false;

while (c1 != NULL)

{

if (((c1->code == 4)&&(strcmp(&(curr->data[1]),&(c1->data[5]))))||

((c1->code == 11)&&(strcmp(&(c1->data[1 + strlen(&(c1->data[1]))]),&(curr->data[1])))))

actual[currActual] = true;

c1 = c1->next;

}

break;

case 10: // check write node

// упрощенный вар-т - не актулен если файла нет

dFId = \*((unsigned int\*)(&(curr->data[1])));

cf = fdt;

actual[currActual] = false;

while ((cf != NULL)&&(!actual[currActual]))

{

if (cf->id == dFId)

actual[currActual] = true;

cf = cf->next;

}

break;

case 11: // check addName

//неактуален, если ссылка удалена

cf = fdt;

actual[currActual] = false;

while ((cf != NULL)&&(!actual[currActual]))

{

if (strcmp(cf->linkName, &(curr->data[(1 + strlen(&(curr->data[1])))])) == 0)

actual[currActual] = true;

cf = cf->next;

}

break;

case 12: // check truncate node

// неактуален если файл удален

dFId = \*((unsigned int\*)(&(curr->data[1])));

cf = fdt;

actual[currActual] = false;

while ((cf != NULL)&&(!actual[currActual]))

{

if (cf->id == dFId)

actual[currActual] = true;

cf = cf->next;

}

break;

}

currActual++;

}

//}

//формируем список исключительно актуальных зкоманд

Command\* cc = pdt[rwrPage].rchain;

Command\* cc1 = NULL;

unsigned int ps = pdt[rwrPage].pageSize;

unsigned int cmi = 0;

pdt[rwrPage].rchain = NULL;

while ((cc != NULL)&&(cc1 == NULL))

{

if (actual[cmi])

cc1 = cc;

cc = cc->next;

cmi++;

}

if (cc1 != NULL)//записываем в файл актуальные команды, дополненные частью новых, внесенных за последний сеанс

pdt[rwrPage].rchain = cc1;

ps = ps - (cc1->data[0] + 8);

while (cc != NULL)

{

if (actual[cmi])

{

cc1->next = cc;

cc1 = cc;

ps = ps - (cc1->data[0] + 8);

}

cc = cc->next;

cmi++;

}

while ((ncm != NULL)&&((ncm->data[0] + 8) < ps))

{

cc1->next = ncm;

cc1 = ncm;

ps = ps - (cc1->data[0] + 8);

ncm = ncm->next;

}

pdt[rwrPage].version++;

}//откываем файл и сериализуем все страницы

fs = fopen(&(data[1]),"w");

for (unsigned int i = 0; i < sd.pageAmount; i++)

{

pdt[i].serialize(fs);

}

fclose(fs);

fs = NULL;

}

void JFFSdriver::blockStat(char \*data)

{

printf("Id of block is %u\n", \*((unsigned int\*)(&(data[1]))));

unsigned int bId = \*((unsigned int\*)(((unsigned long)data) + 1));//вывод статистики блока

if (bId < sd.pageAmount)//вывод остальных служебных полей и определения состояния

{

printf("Block was recorded %u times\n", &(pdt[bId].version));

printf("It contain %u records\n", &(pdt[bId].records));

printf("Size of block is %u\n", &(pdt[bId].pageSize));

}

else

{

printf("There is no such page\n");

go = false;

}

\_getch();

}

void JFFSdriver::create(char \*data)

{//создание ссылки на новый файл

FileDescr\* t = new FileDescr();

t->field = NULL;

t->next = fdt;

t->id = \*((unsigned int\*)(&(data[1])));//присвоение ид, в случае, если команда прочитана из готовой страницы

FileDescr\* curr = fdt;

if (t->id == 0)

while (curr != NULL)

{

if (curr->id >= t->id)

t->id = curr->id + 1;

curr = curr->next;

}

else

t->id = 1; //определение ид создаваемого файла

t->linkName = new char[(data[0] - 5)];

strcpy(t->linkName, &(data[5]));

t->len = 0;

fdt = t;

ChangeChain\* t1 = new ChangeChain();

t1->ch = NULL;

t1->fId = \*((unsigned int\*)(&(data[1])));

t1->next = chgs;

chgs = t1;//продолжение реинита дескриптора файла

}

void JFFSdriver::unlink(char \*data)

{

// execute//удаление ссылки на файл

FileDescr\* unlDescr = NULL;

unsigned int unlinkFId = 0;

FileDescr\* curr = fdt;

if (curr == NULL)

{

printf("Error in link name: %s", &(data[1]));

go = false;//нет такого имени - остановка

}

else

{

if (strcmp(curr->linkName, &(data[1]))==0)

{

unlinkFId = curr->id;

FileDescr\* d = NULL;

while (curr->next != NULL)

{

while((curr->next != NULL)&&(curr->next->id == unlinkFId))

{

d = curr->next;

curr->next = d->next;

delete[] d->linkName;

delete[] d->field;

delete d;

}

curr = curr->next;//найден файл

}

}

else

{

FileDescr\* d = NULL;

while ((curr->next != NULL)&&(unlinkFId == 0))

{

if((strcmp(&(data[1]), curr->next->linkName))==0)

{

FileDescr\* d = curr->next;

curr->next = d->next;

unlinkFId = d->id;

delete[] d->linkName;

delete[] d->field;

delete d;

}

curr = curr->next;

}

d = NULL;

curr = fdt;

while (curr->next != NULL)

{

while((curr->next != NULL)&&(curr->next->id == unlinkFId))

{

d = curr->next;

curr->next = d->next;

delete[] d->linkName;

delete[] d->field;

delete d;

}

curr = curr->next;

}//удаление ссылки из списка страниц

}

}

}

void JFFSdriver::list(char\* data)

{

FileDescr\* curr = fdt;

while (curr != NULL)

{

printf("%s\n", curr->linkName);

curr = curr->next;

}

}//список имен фвйлов

void JFFSdriver::open(char \*data)

{

FileDescr\* curr = fdt;

unsigned int FId = 0;

while (curr != NULL)

{//найти нужный файл, вернуть его ид

if (strcmp(&(data[1]), curr->linkName)==0)

{

FId = curr->id;

printf("File Id is %u\n", &(curr->id));

}

curr = curr->next;

}

if (FId == 0)

{

printf("There is no such link");

go = false;

}

}

void JFFSdriver::close(char\* data)

{

FileDescr\* curr = fdt; // проглотить)

unsigned int FId = 0;

while (curr != NULL)

{

if (strcmp(&(data[1]), curr->linkName)==0)

FId = curr->id;

curr = curr->next;

}

if (FId == 0)

{

printf("There is no such link");

go = false;

}

}

void JFFSdriver::read(char \*data)

{

FileDescr\* curr = fdt; //вернуть масив байтов требуемой длины с требуемого смещения

while (curr != NULL)

{

if (\*((unsigned int\*)(&(data[9]))) = curr->id)

{

for (unsigned int i = \*((unsigned int\*)(&(data[1])));

i < (\*((unsigned int\*)(&(data[1]))) + \*((unsigned int\*)(&(data[5])))); i++)

printf("%c", &(curr->field[i]));

curr = NULL;

}

else

curr = curr->next;

}

}

void JFFSdriver::write(char\* data)

{

FileDescr\* curr = fdt;//аналогично - запись + добвление команды в список

unsigned int u = 9;

while (curr != NULL)//добвление команды в список

{

if (\*((unsigned int\*)(&(data[1]))) = curr->id)

{

for (unsigned int i = \*((unsigned int\*)(&(data[5])));

i < ((\*((unsigned int\*)((unsigned long)data + 5))) + strlen(&(data[9]))); i++, u++)

curr->field[i] = data[u];

}

curr = curr->next;

}

ChangeChain\* t1 = chgs;

ChangeChain\* t2 = NULL;

while (go && (t1 != NULL))

if (t1->fId == \*((unsigned int\*)(&(data[5]))))

t2 = t1;

if (t2 != NULL)

{

Change\* t3 = new Change();

t3->amount = strlen(&(data[9]));

t3->offset = \*((unsigned int\*)(&(data[5])));

t3->next = t2->ch;

t2->ch = t3;

}

}

void JFFSdriver::addName(char \*data)

{//добавить ссылку на существ. файл

int fLen = strlen(&(data[1]));

char\* fName = new char[fLen];

char\* sName = new char[strlen(&(data[1 + fLen]))];

FileDescr\* curr = fdt;

unsigned int FId = 0;

unsigned int FLen = 0;

while (curr != NULL)

{

if (strcmp(fName, curr->linkName)==0)

{

FId = curr->id;

FLen = curr->len;

}

curr = curr->next;

}

if (FId == 0)

{

printf("There is no such link");

go = false;

}

else

{

FileDescr\* t = new FileDescr();

t->next = fdt;

t->id = FId;

t->linkName = new char[(data[0] - fLen)];

strcpy(t->linkName, &(data[fLen + 1]));

t->len = FLen;

fdt = t;

} //копируем его, если он существует

}

void JFFSdriver::truncate(char\* data)

{

// realize later //изменить размер

FileDescr\* curr = fdt;

unsigned int FId = 0;

while ((curr != NULL)&&(strcmp(&(data[5]), curr->linkName)!=0))

{

curr = curr->next;

}

if (FId == 0)

{

printf("No file with such link");

go = false;

}

else

{

char\* old = curr->field;

unsigned int oldLen = curr->len;

curr = fdt;

while (curr != NULL)

{

if (curr->id = FId)

{

curr->len = \*((unsigned int\*)(&(data[1])));

curr->field = new char[curr->len];

if (curr->len < oldLen)

for (unsigned int i = 0; i < curr->len; i++)

curr->field[i] = old[i];

}

}

delete[] old;

}

ChangeChain\* t1 = chgs;

ChangeChain\* t2 = NULL;

while (go && (t1 != NULL))

if (t1->fId == FId)

t2 = t1;

if (t2 != NULL)

{

Change\* t3 = new Change();

t3->amount = \*((unsigned int\*)(&(data[1])));

t3->offset = 0;

t3->next = t2->ch;

t2->ch = t3;

} //перезапись файл, добавить запись цепочку изменений

}

//input methods

void JFFSdriver::inputFormat()

{

char data[100];//упаковка кода обьем, кол-во блоков

printf("\nInput size of block: ");

scanf("%u", &(data[1]));

printf("\nInput amount of blocks: ");

scanf("%u", &(data[5]));

printf("\nInput target path: ");

scanf("%s", &(data[9]));

data[0] = 9 + (char)strlen(&(data[9]));

(this->\*actions[0])(data);

}

void JFFSdriver::inputMount()

{

char data[100];//упаковка пути к системе

printf("\nInput path to file system:");

scanf("%s", &(data[1]));

data[0] = 1 + (char)strlen(&(data[1]));

(this->\*actions[1])(data);

}

void JFFSdriver::inputUnmount()

{

char data[100];//ввод целевого файла, куда сохранися фс

printf("\nInput path to file system:");

scanf("%s", &(data[1]));

data[0] = 1 + (char)strlen(&(data[1]));

(this->\*actions[2])(data);

}

void JFFSdriver::inputBlockStat()

{//ввести номер блока, статистику которого мы получим

char data[5];

printf("\nInput id of block you need:");

scanf("%u", &(data[1]));

data[0] = 5;

(this->\*actions[3])(data);

}

void JFFSdriver::inputCreate()

{

char data[100];//создание команды, вызов процедуры обработки, добавление команды в список

printf("\nInput name of file you want create:");

scanf("%s", &(data[5]));

data[1] = 0; data[2] = 0; data[3] = 0; data[4] = 0;

data[0] = 5 + (char)strlen(&(data[1]));

(this->\*actions[4])(data);

Command\* nc = new Command();

nc->next = cm;

nc->code = 4;

nc->id = 0;

Command\* curr1;

if (ncm != NULL)

{

curr1 = ncm;

while (curr1->next != NULL)

curr1 = curr1->next;

nc->id = curr1->id + 1;

}

else

{

if (cm != NULL)

{

curr1 = cm;

while (curr1->next != NULL)

curr1 = curr1->next;

nc->id = curr1->id + 1;

}

}

nc->data = new char[data[0]];

for (int i = 0; i < data[0]; i++)

nc->data[i] = data[i];

cm = nc;

}

void JFFSdriver::inputUnlink()

{

char data[100]; //аналогично - удаление

printf("\nInput name of file you want unlink:");

scanf("%s", &(data[1]));

data[0] = 1 + (char)strlen(&(data[1]));

(this->\*actions[5])(data);

Command\* nc = new Command();

nc->next = cm;

nc->code = 5;

nc->id = 0;

Command\* curr1;

if (ncm != NULL)

{

curr1 = ncm;

while (curr1->next != NULL)

curr1 = curr1->next;

nc->id = curr1->id + 1;

}

else

{

if (cm != NULL)

{

curr1 = cm;

while (curr1->next != NULL)

curr1 = curr1->next;

nc->id = curr1->id + 1;

}

}

nc->data = new char[data[0]];

for (int i = 0; i < data[0]; i++)

nc->data[i] = data[i];

cm = nc;

}

void JFFSdriver::inputList()

{

char t = 0;

(this->\*actions[6])(&t);//команд показать список файлов

}

void JFFSdriver::inputOpen()

{

char data[100];

printf("\nInput name of file you want open:");

scanf("%s", &(data[1]));

data[0] = 1 + (char)strlen(&(data[1]));

(this->\*actions[7])(data);//упаковать ид требуемого файла и вернуть его ид

}

void JFFSdriver::inputClose()

{

char data[100];

printf("\nInput name of file you want close:");

scanf("%s", &(data[1]));

data[0] = 1 + (char)strlen(&(data[1]));

(this->\*actions[8])(data); //останавливается при ошбке, не делает ничего

}

void JFFSdriver::inputRead()

{

char data[100];

printf("\nInput id of file you want read:");

scanf("%u", &(data[9]));

printf("\nInput offset to read:");

scanf("%u", &(data[1]));

printf("\nInput amount of bytes to read:");

scanf("%u", &(data[5]));

data[0] = 13;

(this->\*actions[9])(data);считывает файл,с определенным смещением и опредленным размером даных

}

void JFFSdriver::inputWrite()

{

char data[100];

printf("\nInput id of file you want write:");

scanf("%u", &(data[1]));

printf("\nInput offset to write:");

scanf("%u", &(data[5]));

printf("\nInput info to write:");

scanf("%s", &(data[9]));

data[0] = 9 + (char)strlen(&(data[9]));

(this->\*actions[10])(data);

//input Change

Command\* nc = new Command();

nc->next = cm;

nc->code = 10;

nc->id = 0;

Command\* curr1;

if (ncm != NULL)

{

curr1 = ncm;

while (curr1->next != NULL)

curr1 = curr1->next;

nc->id = curr1->id + 1;

}

else

{

if (cm != NULL)

{

curr1 = cm;

while (curr1->next != NULL)

curr1 = curr1->next;

nc->id = curr1->id + 1;

}

}

nc->data = new char[data[0]];

for (int i = 0; i < data[0]; i++)

nc->data[i] = data[i];

cm = nc; //добавление оманды, модификация файла...

}

void JFFSdriver::inputAddName()

{

char data[100];

printf("\nInput name of file you want to add name:");

scanf("%s", &(data[1]));

data[0] = 1 + (char)strlen(&(data[1]));

printf("\nInput new name:");

scanf("%s", &(data[(data[0])]));

data[0] += (char)strlen(&(data[(data[0])]));

(this->\*actions[11])(data);

Command\* nc = new Command();

nc->next = cm;

nc->code = 11;

nc->id = 0;

Command\* curr1;

if (ncm != NULL)

{

curr1 = ncm;

while (curr1->next != NULL)

curr1 = curr1->next;

nc->id = curr1->id + 1;

}

else

{

if (cm != NULL)

{

curr1 = cm;

while (curr1->next != NULL)

curr1 = curr1->next;

nc->id = curr1->id + 1;

}

}

nc->data = new char[data[0]];

for (int i = 0; i < data[0]; i++)

nc->data[i] = data[i];

cm = nc;//упаковываем старое и новое имя файла, а также вносим команду в глобальный список команд

}

void JFFSdriver::inputTruncate()

{

char data[100];

printf("\nInput name of changing file: ");

scanf("%u", &(data[5]));

printf("\nInput new size of file: ");

scanf("%u", &(data[1]));

data[0] = 5 + (char)strlen(&(data[5]));

(this->\*actions[12])(data);

Command\* nc = new Command();

nc->next = cm;

nc->code = 12;

nc->id = 0;

Command\* curr1;

if (ncm != NULL)

{

curr1 = ncm;

while (curr1->next != NULL)

curr1 = curr1->next;

nc->id = curr1->id + 1;

}

else

{

if (cm != NULL)

{

curr1 = cm;

while (curr1->next != NULL)

curr1 = curr1->next;

nc->id = curr1->id + 1;

}

}

nc->data = new char[data[0]];

for (int i = 0; i < data[0]; i++)

nc->data[i] = data[i];

cm = nc; вкладываем имя файла и его новый размер

}

//init arrays of methods

JFFSdriver::JFFSdriver()

{

inputActions[0] = &JFFSdriver::inputFormat; actions[0] = &JFFSdriver::format;

inputActions[1] = &JFFSdriver::inputMount; actions[1] = &JFFSdriver::mount;

inputActions[2] = &JFFSdriver::inputUnmount; actions[2] = &JFFSdriver::unmount;

inputActions[3] = &JFFSdriver::inputBlockStat; actions[3] = &JFFSdriver::blockStat;

inputActions[4] = &JFFSdriver::inputCreate; actions[4] = &JFFSdriver::create;

inputActions[5] = &JFFSdriver::inputUnlink; actions[5] = &JFFSdriver::unlink;

inputActions[6] = &JFFSdriver::inputList; actions[6] = &JFFSdriver::list;

inputActions[7] = &JFFSdriver::inputOpen; actions[7] = &JFFSdriver::open;

inputActions[8] = &JFFSdriver::inputClose; actions[8] = &JFFSdriver::close;

inputActions[9] = &JFFSdriver::inputRead; actions[9] = &JFFSdriver::read;

inputActions[10] = &JFFSdriver::inputWrite; actions[10] = &JFFSdriver::write;

inputActions[11] = &JFFSdriver::inputAddName; actions[11] = &JFFSdriver::addName;

inputActions[12] = &JFFSdriver::inputTruncate; actions[12] = &JFFSdriver::truncate;

//инициализация автоматов

}

//service

void JFFSdriver::setReady()

{

go = true;

// TODO: add actions initialization here

}

void JFFSdriver::commandCycle()

{

fs = NULL;

int act;

while (go)

{

printf("\nInput code of command which you want to start:\n");

if (fs != NULL)

{

printf("2 - Unmount file system\n");

printf("3 - Block statistics\n");

printf("4 - Create new file\n");

printf("5 - Unlink link to the file\n");

printf("6 - Display list of files\n");

printf("7 - Open existing file\n");

printf("8 - Close opened file\n");

printf("9 - Read some information from file\n");

printf("10 - Write some information to file\n");

printf("11 - Set link to existing file\n");

printf("12 - Set size for file\n");

}

else

{

printf("0 - Format file system\n");

printf("1 - Mount file system\n");

}

scanf("%d", &act);//ввод кода команды и передача контроля в автомат

if (((act < 2)&&(act >= 0)&&(fs == NULL)) || ((act > 1)&&(act < 14)&&(fs != NULL)))

{

//JFFSdriver\* t = new JFFSdriver();

(this->\*inputActions[act])();

}

else

{

printf("\nIncorrect command code");

go = false;

}

\_getch();

}

}