Міністерство освіти і науки України

Національний технічний університет України

«Київський Політехнічний Інститут»

Факультет прикладної математики

Кафедра Спеціалізованих комп’ютерних систем

Лабораторна робота №4

З дисципліни «Операційні системи» :

«Файлова система»

Варіант №16

Виконали: Перевірив:

студенти групи КВ-91 Зайцев В. Г.

Вакулюк В. О.

Федай Г. Г.

Київ 2012

**Завдання:**

Відповідно до моделі файлової системи FAT-16 створити таблицю розміщення файлів. Сумарна кількість кластерів має бути визначена попередньо. Необхідно імітувати процеси створення файлів заданого у кластерах розміру, збільшення файлів на задану кількість кластерів та знищення файлів. Передбачити пошук файлів за їх ідентифікатором та роздруківку номерів кластерів, які займає файл.

#include <iostream>

using namespace std;

namespace local

{

const int ClasterNumber = 1000;

const int SizeOfClaster = 64;

const int MaxFileSize/\*in clasters\*/ = 500;

typedef struct Node

{

int \*address;

size\_t size;

char \*name;

Node \*\*Ways;

int CurrentWay;

int WaysNumber;

Node \*prev;

Node \*CurrentFolder;

};

Node \*Top;

typedef struct IDENT

{

Node \*CurrentFolder;

} \*IDENT\_PTR;

bool Exists(Node ident, char \*name)

{

for (int i = 0; i != ident.CurrentWay; i++)

if (strcmp(ident.Ways[i]->name, name) == NULL)

return true;

return false;

}

#define MaxPathLength 100 // Depth of includes

#define MaxNameLength 64 // Limit on name size

char \*\*strings; // Path string

char \*\*strs; // Command strings

typedef char \*char\_ptr;

typedef Node \*Node\_ptr;

struct TREE

{

int CurrentPathStringPosition;

char\*\* CurrentPath;

Node \*top;

Node \*CurrentPosition;

};

TREE Tree;

// Commands' mnemonics

const char \*cd = "cd";

const char \*pp = "..";

const char \*mkdir = "mkdir";

const char \*mkfile = "mkfile";

const char \*del = "del";

const char \*resize = "resize";

const char \*copy = "copy";

const char \*move = "move";

const char \*toscreen = "toscreen";

const char \*tree = "tree";

const char \*dir = "dir";

const char \*exit = "exit";

const char \*dump = "dump";

// Errors' array

char \*Errors[] = { "", "Cannot include more objects", "Limit of depth", "", "", "", ""};

// Defines of command errors

#define ERROR\_NONE 0x0001

#define ERROR\_THE\_LIMIT\_OF\_INCLUDE\_OBJECTS 0x0002

bool AddObject(Node \*ident, char \*name, size\_t size);

void Diagnostic()

{

}

void CreateFileSystem(char \*name, int capacity)

{

int i = 0;

while (name[i] != '\0') i++;

name[i] = ':'; name[i + 1] = '\\'; name[i + 2] = '\0';

Tree.top = new Node;

Tree.top->CurrentWay = 0;

Tree.top->name = new char[capacity];

strcpy(Tree.top->name, name);

Tree.top->WaysNumber = capacity;

Tree.top->Ways = new Node\_ptr[Tree.top->WaysNumber];

Tree.CurrentPosition = Tree.top;

Tree.CurrentPath = new char\_ptr[MaxPathLength];

//Tree.CurrentPathStringPosition = 0;

for (int i = 0; i != MaxPathLength; i++)

Tree.CurrentPath[i] = new char[MaxNameLength];

strcpy(Tree.CurrentPath[1], name);

Tree.CurrentPathStringPosition = 1;

cout << "File system was created";

}

void ShowCurrentPosition()

{

for (int i = 0; i != Tree.CurrentPathStringPosition; i++)

cout << Tree.CurrentPath[i] << "\\";

}

void InitRequestArray()

{

strings = new char\_ptr[MaxPathLength];

for (int i = 0; i != MaxPathLength; i++)

strings[i] = new char[MaxNameLength];

}

int Split(const char \*way, const char \*delimiter, char \*\*strings)

{

int i = 0, j = 0, index = 0;

char \*word = new char[MaxNameLength];

for (i = 0; i != strlen(way); i++)

if (way[i] == \*delimiter)

{

word[j++] = '\0';

strcpy(strings[index++], word);

j = 0;

}

else

{

word[j] = way[i];

j++;

}

word[j] = '\0';

strcpy(strings[index], word);

return index + 1;

}

Node \*ChooseWay(Node \*node, char\* name)

{

for (int i = 0; i != node->CurrentWay; i++)

if (strcmp(node->Ways[i]->name, name) == NULL)

return node->Ways[i];

return NULL;

}

Node \*FindWay(char \*\*way, int number)

{

Node \*tmp = new Node;

tmp->CurrentWay = 0;

tmp->name = new char[MaxNameLength];

for (int i = 0; i != number; i++)

if (!(tmp = ChooseWay(local::Tree.CurrentPosition, way[i])))

//tmp = Tree.CurrentPosition;

//else

return NULL;

return tmp;

}

bool RetBack()

{

//if ()

local::Tree.CurrentPathStringPosition--;

local::Tree.CurrentPosition = local::Tree.CurrentPosition->prev;

return true;

}

struct ERROR

{

int error\_code;

};

int Exit = false;

int SwitchOff = false;

void MemoryDump();

void \*malloc\_(int Size, char \*name);

void \*free\_(int);

void DelProc(Node \*node)

{

int i = 0, j = 0;

int \*ptr;

ptr = node->address;

while (\*ptr)

{

free\_(\*ptr);

ptr = ++node->address;

}

for (i = 0; i != node->CurrentWay; i++)

{

delete[] node->Ways[i];

node->Ways[i] = NULL;

}

}

Node \* Remove(Node \*node)

{

int i;

for (i = 0; i != node->prev->CurrentWay; i++)

{

if (strcmp(node->prev->Ways[i]->name, node->name) == NULL)

break;

}

if (node->prev->CurrentWay)

{

node->prev->Ways[i] = node->prev->Ways[node->prev->CurrentWay -1];

node->prev->CurrentWay--;

}

return node;

}

void RecDelProc(Node \*node)

{

for (int i = 0; i != node->CurrentWay; i++)

RecDelProc(node->Ways[i]);

DelProc(node);

}

void RecDel(Node \*node)

{

Remove(node);

RecDelProc(node);

}

void CheckCommand(char \*command)

{

ERROR error;

int number = Split(command, "|", strs);

if (strcmp(\*local::strs, local::cd) == NULL)

{

if (strcmp(local::strs[1], pp) == NULL)

RetBack();

//CurrentNode = CurrentNode.prev;

else

{

int j = 0;

int number = Split(local::strs[1], "\\", local::strings);

Node \*tmp = FindWay(local::strings, number);

if (!tmp)

{

cout << "Defined way wasn't found";

return;

}

Tree.CurrentPosition = tmp;

// error.error\_code = (FindWay(strs, number))?ERROR\_NONE:ERROR\_THE\_LIMIT\_OF\_INCLUDE\_OBJECTS;

for (int i = Tree.CurrentPathStringPosition + 1; i != Tree.CurrentPathStringPosition + number + 1; i++)

{

strcpy(Tree.CurrentPath[i], strings[j]);

j++;

}

Tree.CurrentPathStringPosition += number;

}

return;

}

if (strcmp(\*strs, exit) == NULL)

strcpy(Tree.CurrentPosition->Ways[Tree.CurrentPosition->CurrentWay]->name, strs[1]);

if (strcmp(\*strs, exit) == NULL)

{

Exit = true;

SwitchOff = true;

return;

}

if (strcmp(\*local::strs, mkdir) == NULL)

{

AddObject(Tree.CurrentPosition, strs[1], 0);

return;

}

if (strcmp(\*local::strs, mkfile) == NULL)

{

AddObject(Tree.CurrentPosition, strs[1], atoi(strs[2]));

return;

}

if (strcmp(\*local::strs, dump) == NULL)

{

MemoryDump();

return;

}

if (strcmp(\*local::strs, dir) == NULL)

{

void \*address;

for (int i = 0; i != local::Tree.CurrentPosition->CurrentWay; i++)

if (address = local::Tree.CurrentPosition->Ways[i]->address)

{

cout << local::Tree.CurrentPosition->Ways[i]->name;

cout << " Start address: " << Tree.CurrentPosition->Ways[i]->address;

cout << " Size: " << Tree.CurrentPosition->Ways[i]->size;

cout << " Ext: ";

cout << "\n";

}

else

{

cout << Tree.CurrentPosition->Ways[i]->name;

cout << "\n";

}

return;

}

if (strcmp(\*local::strs, del) == NULL)

{

Node \*ptr;

int number = Split(local::strs[1], "\\", local::strings);

if (ptr = FindWay(strings, number))

RecDel(ptr);

return;

}

else

{

cout << "Unknown command '" << command << "'\n";

//ShowCurrentPosition();

}

}

void Request(void)

{

cout << "\n";

for (int i = 1; i != Tree.CurrentPathStringPosition + 1; i++)

cout << Tree.CurrentPath[i] << "\\";

}

bool LoadFileSystem()

{

//char \*str = new char[10];

cout << "\n" << Tree.CurrentPosition->name << ":\\";

strs = new char\_ptr[MaxPathLength];

for (int i = 0; i != MaxPathLength; i++)

strs[i] = new char[50];

strings = new char\_ptr[MaxPathLength];

for (int i = 0; i != MaxPathLength; i++)

strings[i] = new char[MaxNameLength];

//scanf("%s", str);

//CheckCommand(str);

return true;

}

bool ExitFileSystem()

{

char \*str = new char[10];

cout << "root:\\";

scanf("%s", str);

CheckCommand(str);

return true;

}

int \*malloc\_\_(size\_t size, char\* name);

bool AddObject(Node \*ident, char \*name, size\_t size)

{

if (Exists(\*ident, name))

{

cout << "File already exists";

return false;

}

if (ident->CurrentWay + 1 != ident->WaysNumber)

{

int \*ptr;

//ident.CurrentFolder->Ways[ident.CurrentFolder->CurrentWay] = new Node;

ident->Ways[ident->CurrentWay] = new Node;

ident->Ways[ident->CurrentWay]->address = NULL;

ident->Ways[ident->CurrentWay]->address = new int[1024];

\*(ident->Ways[ident->CurrentWay]->address) = 0;

if (size)

if (!(ident->Ways[ident->CurrentWay]->address[0] = (int)malloc\_(size, name)))

{

if (ptr = malloc\_\_(size, name))

ident->Ways[ident->CurrentWay]->address = ptr;

else

return false;

}

else

{

ident->Ways[ident->CurrentWay]->address[1] = 0;

ident->Ways[ident->CurrentWay]->size = size;

}

ident->Ways[ident->CurrentWay]->prev = ident;

ident->Ways[ident->CurrentWay]->CurrentWay = 0;

ident->Ways[ident->CurrentWay]->WaysNumber = 10;

ident->Ways[ident->CurrentWay]->name = new char[MaxNameLength];

ident->Ways[ident->CurrentWay]->Ways = new Node\_ptr[ident->Ways[ident->CurrentWay]->WaysNumber];

strcpy(ident->Ways[ident->CurrentWay]->name, name);

ident->CurrentWay++;

}

else

{

cout << "Not enough place";

return false;

}

return true;

}

struct str\_ptr

{

bool flag;

char file[300];

int size;

int addr;

int empty;

str\_ptr \*prev;

str\_ptr \*next;

};

str\_ptr \*first;

str\_ptr \*last;

str\_ptr \*memory;

void Message(int sign, int size)

{

if (sign == 1)

cout << "Cannot allocate the memory block with specified length - " << size <<"\n";

}

void Paste(int addr\_to, int buffer\_from)

{

}

//void \*malloc\_(int Size);

void \*free\_(int addr)

{

str\_ptr \*tmp = new str\_ptr;

str\_ptr \*ptr = new str\_ptr;

ptr = first->next;

while (ptr != NULL)

{

if ((ptr->next->addr) == addr)

{

tmp = ptr->next;

/\*

if(ptr->empty != 0)

{

Message(1, addr);

return NULL;

}

\*/

ptr->empty += tmp->size + tmp->empty;

local::first->empty += tmp->size;//tmp->size + tmp->empty;

ptr->next = tmp->next;

tmp->next->prev = ptr;

delete[] tmp;

break;

}

ptr = ptr->next;

}

return (void\*)addr;

}

FILE \*fp;

char filename[200];

char \*str = new char[300];

bool OpenFile(char \*filename)

{

if (fp = fopen(filename, "r"))

return true;

return false;

}

char\* CommandRead()

{

//while (str = fgets(str, 300, fp))

return fgets(local::str, 300, local::fp);

}

void \*realloc\_(int addr, int Size, char \*name)

{

int copy\_desc;

str\_ptr \*ptr = new str\_ptr;

int tmp\_addr;

int Resize;

ptr = first->next;

while(ptr != NULL)

{

if (ptr->addr == addr)

{

Resize = ptr->next->size + Size;

if(Resize > ptr->next->size + ptr->next->empty)

{

//copy\_desc = Copy(ptr->next);

if(tmp\_addr = (int)malloc\_(Resize, name))

Paste(tmp\_addr, copy\_desc);

else

{

Message(1, Size);

return NULL;

}

}

else

{

ptr->next->size += Size;

ptr->next->empty -= Size;

}

}

ptr = ptr->next;

}

if (ptr == NULL)

{

Message(1, Size);

return NULL;

}

else

{

free\_(ptr->addr);

return (void\*)tmp\_addr;

}

// return (void\*)tmp\_addr;

}

void \*malloc\_(int Size, char \*name)

{

str\_ptr \*tmp = new str\_ptr;

str\_ptr \*ptr = new str\_ptr;

str\_ptr \*newblock = new str\_ptr;

ptr = first->next;

//Size += (Size % 4);

while(ptr != NULL)

{

if (ptr->empty >= Size)

{

tmp = ptr->next;

ptr->next = newblock;

tmp->prev = newblock;

newblock->prev = ptr;

newblock->next = tmp;

newblock->flag = true;

newblock->size = Size;

strcpy(newblock->file, name);

newblock->addr = ptr->addr + Size;

newblock->empty = ptr->empty - Size;

first->empty -= Size;

ptr->empty = 0;

break;

}

ptr = ptr->next;

}

if (ptr == NULL)

{

Message(1, Size);

return NULL;

}

else

return (void\*)newblock->addr;//(newblock->addr - newblock->size);

}

int \*malloc\_\_(size\_t size, char \*name)

{

int index = 0;

int \*address\_array = new int[1024];

str\_ptr \*ptr = local::first->next;

if (local::first->empty >= size)

{

while (ptr)

{

if (ptr->empty < size)

{

if (ptr->empty != 0)

{

size -= ptr->empty;

address\_array[index] = (int)malloc\_(ptr->empty, name);

index++;

}

}

else

{

address\_array[index] = (int)malloc\_(size, name);

address\_array[index + 1] = 0;

size = 0;

break;

}

ptr = ptr->next;

}

if (!ptr)

{

cout << "Unusual error";

return NULL;

}

}

else

{

cout << "Not enough place to allocate size " << size;

return NULL;

}

return address\_array;

}

void MemoryDump()

{

str\_ptr \*ptr;// = new str\_ptr;

ptr = first;

while(ptr->next != NULL)

{

printf("size: %d, ", ptr->size);

printf("flag: %d, ", ptr->flag);

printf("addr: %d, ", ptr->addr);

printf("empty: %d ", ptr->empty);

printf("file: %s ", ptr->file);

printf("\n");

ptr = ptr->next;

}

}

#define MaxCommandLength 50

void CommandManager()

{

char \*command = new char[MaxCommandLength];

//OpenFile("C:\\Documents and Settings\\Seva\\Мои документы\\Visual Studio 2008\\Projects\\OS\_L4a\\Debug\\commands.txt");

//OpenFile("D:\\commands.txt");

while (!SwitchOff)

{

while (!Exit)

{

/\*

while (command = CommandRead())

{

Request(); // form new request

command[strlen(command) - 1] = '\0';

printf("%s", command);

CheckCommand(command); // perform the reaction

}

\*/

//fclose(fp);

Request(); // form new request

scanf("%s", command); // waiting reaction on the request

CheckCommand(command); // perform the reaction

}

}

}

int ConvertToInt(char\* str)

{

int sum;

int main = 0;

int weight = 1;

int length = strlen(str) - 1;

for(int i = length; i != -1; i--)

{

if( str[i] == '0')

sum = 0;

if( str[i] == '1')

sum = 1;

if( str[i] == '2')

sum = 2;

if( str[i] == '3')

sum = 3;

if( str[i] == '4')

sum = 4;

if( str[i] == '5')

sum = 5;

if( str[i] == '6')

sum = 6;

if( str[i] == '7')

sum = 7;

if( str[i] == '8')

sum = 8;

if( str[i] == '9')

sum = 9;

sum \*= weight;

main += sum;

weight \*= 10;

}

return main;

}

#define MeMSiZe 1024

void Init()

{

first = new str\_ptr;

last = new str\_ptr;

first->size = 0;

first->addr = 0;

first->flag = true;

first->empty = MeMSiZe;

first->addr = 0;

memory = new str\_ptr;

memory->empty = MeMSiZe;

memory->flag = true;

memory->size = 0;

memory->prev = first;

memory->next = last;

memory->addr = 0;

first->next = memory;

last->size = 0;

last->flag = true;

last->next = NULL;

last->prev = memory;

}

}

using namespace local;

int main()

{

char\* name,\* number\_str;

cout << "Insert name of main folder\n";

scanf("%s", name = new char[10]);

cout << "Insert capacity value\n";

scanf("%s", number\_str = new char[10]);

int number = atoi(number\_str);

Init();

malloc\_(120, "0");

CreateFileSystem(name, number);

LoadFileSystem();

CommandManager();