**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

**отчет**

**по лабораторной работе №3**

**по дисциплине «Программирование»**

**Тема: Обход файловой системы**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3343 |  | Пименов П.В. |
| Преподаватель |  | Государкин Я.С. |

Санкт-Петербург

2024

## Цель работы

Изучить общий принцип работы рекурсивных алгоритмов, написать программу на языке C, выполняющую обход файловой системы.

## Задание

Вариант 1. Дана некоторая корневая директория, в которой может находиться некоторое количество папок, в том числе вложенных. В этих папках хранятся некоторые текстовые файлы, имеющие имя вида .txt. Требуется найти файл, который содержит строку "Minotaur" (файл-минотавр). Файл, с которого следует начинать поиск, всегда называется file.txt (но полный путь к нему неизвестен). Каждый текстовый файл, кроме искомого, может содержать в себе ссылку на название другого файла (эта ссылка не содержит пути к файлу). Таких ссылок может быть несколько. Программа должна вывести правильную цепочку файлов (с путями), которая привела к поимке файла-минотавра. Цепочка, приводящая к файлу-минотавру может быть только одна. Общее количество файлов в каталоге не может быть больше 3000. Циклических зависимостей быть не может. Файлы не могут иметь одинаковые имена. Ваше решение должно находиться в директории /home/box, файл с решением должен называться solution.c. Результат работы программы должен быть записан в файл result.txt. Ваша программа должна обрабатывать директорию, которая называется labyrinth.

## Выполнение работы

Описание функций

1. *char\* read\_file(char\* path)* – считывает текст из файла, возвращает указатель на считанную строку
2. *char\* strsepcat(char\* left, char\* sep, char\* right)* – соединяет три строки в одну (в порядке расположения аргументов), возвращает указатель на новую строку
3. *char\* create\_path(char\* path, char\* step)* – создает новый путь к файлу (строку), возвращает на нее указатель
4. *char\* create\_track(char\* old\_track, char\* step)* – создает новую цепочку файлов с путями (строку), возвращает на нее указатель
5. *char\* find\_file(char\* root\_path, char\* search\_path, char\* target\_file)* – осуществляет рекурсивный поиск файла по имени

Изначально программа ищет файл с именем «file.txt»‎. Считывается его текст, если это файл-минотавр, то возвращается путь до него, если же он содежит «@include …», то запускается поиск файлов по имени, содержащихся в этих «включениях». В итоге возвращается функция возвращает полную цепочку файлов с путями до файла-минотавра, которая записывает эту цепочку в файл «result.txt»‎.

Разработанный программный код см. в приложении А.

## 

## Выводы

Был изучен общий принцип работы рекурсивных алгоритмов, написана программа на языке C, выполняющая обход файловой системы.

# Приложение А Исходный код программы

Название файла: main.c

#include <dirent.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MEMORY\_BLOCK\_SIZE 10

#define ROOT\_DIR "./labyrinth"

#define ROOT\_FILE "file.txt"

#define OUTPUT\_FILE "result.txt"

char\* read\_file(char\* path);

char\* strsepcat(char\* left, char\* sep, char\* right);

char\* create\_path(char\* path, char\* step);

char\* create\_track(char\* old\_track, char\* step);

char\* find\_file(char\* root\_path, char\* search\_path, char\* target\_file);

char\*

read\_file(char\* path) {

FILE\* file = fopen(path, "r");

if (!file) {

return NULL;

}

size\_t size = 0, capacity = MEMORY\_BLOCK\_SIZE;

char\* content = (char\*)malloc(sizeof(char) \* capacity);

int last\_char = getc(file);

if (last\_char == EOF || last\_char == '\0') {

fclose(file);

free(content);

content = NULL;

return NULL;

}

content[size++] = last\_char;

content[size] = '\0';

last\_char = getc(file);

while (last\_char != EOF && last\_char != '\0') {

content[size++] = last\_char;

content[size] = '\0';

if (size == capacity - 1) {

capacity += MEMORY\_BLOCK\_SIZE;

content = (char\*)realloc(content, capacity);

}

last\_char = getc(file);

}

fclose(file);

return content;

}

char\*

strsepcat(char\* left, char\* sep, char\* right) {

char\* new = (char\*)malloc(sizeof(char) \* (strlen(left) + strlen(sep) + strlen(right) + 1));

sprintf(new, "%s%s%s", left, sep, right);

return new;

}

char\*

create\_path(char\* path, char\* step) {

return strsepcat(path, "/", step);

}

char\*

create\_track(char\* old\_track, char\* step) {

return strsepcat(step, "\n", old\_track);

}

char\*

find\_file(char\* root\_path, char\* search\_path, char\* target\_file) {

char\* result = NULL;

DIR\* current\_dir = opendir(search\_path);

if (!current\_dir) {

return result;

}

struct dirent\* entry;

while ((entry = readdir(current\_dir))) {

if (entry->d\_type == DT\_REG) {

if (!strcmp(entry->d\_name, target\_file)) {

char\* file\_path = create\_path(search\_path, target\_file);

char\* text = read\_file(file\_path);

if (!strcmp(text, "Minotaur")) {

result = strdup(file\_path);

free(file\_path);

file\_path = NULL;

free(text);

text = NULL;

break;

} else if (strstr(text, "@include ")) {

char\* text\_duplicate = strdup(text);

char\* msc;

char\* line = strtok\_r(text\_duplicate, "\n", &msc);

char\* node\_file\_name = NULL;

char\* node\_result = NULL;

while (line && !node\_result) {

node\_file\_name = strstr(line, " ") + 1 \* sizeof(char);

node\_result = find\_file(root\_path, root\_path, node\_file\_name);

line = strtok\_r(NULL, "\n", &msc);

}

if (node\_result) {

result = create\_track(node\_result, file\_path);

free(text\_duplicate);

text\_duplicate = NULL;

free(node\_result);

node\_result = NULL;

free(file\_path);

file\_path = NULL;

free(text);

text = NULL;

break;

}

free(text\_duplicate);

text\_duplicate = NULL;

}

//случай с Deadlock или мусором автоматически отбрасывается

free(file\_path);

file\_path = NULL;

free(text);

text = NULL;

break;

}

} else if (entry->d\_type == DT\_DIR) {

if (!strcmp(entry->d\_name, ".") || !strcmp(entry->d\_name, "..")) {

continue;

}

char\* new\_path = create\_path(search\_path, entry->d\_name);

char\* next\_dir = find\_file(root\_path, new\_path, target\_file);

free(new\_path);

new\_path = NULL;

if (next\_dir) {

result = next\_dir;

break;

}

}

}

closedir(current\_dir);

return result;

}

int

main() {

char\* result\_track = find\_file(ROOT\_DIR, ROOT\_DIR, ROOT\_FILE);

FILE\* output\_file = fopen(OUTPUT\_FILE, "w");

if (output\_file) {

fprintf(output\_file, "%s", result\_track);

}

fclose(output\_file);

free(result\_track);

result\_track = NULL;

return 0;

}

# Приложение Б ТЕСТИРОВАНИЕ

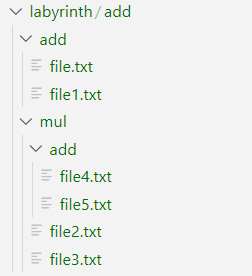


Рисунок 1 – Расположение файлов для теста

Содержимое файлов:

* file.txt:

@include file1.txt

@include file4.txt

@include file5.txt

* file1.txt:

Deadlock

* file2.txt:

@include file3.txt

* file3.txt:

Minotaur

* file4.txt:

@include file2.txt

@include file1.txt

* file5.txt:

Deadlock

Результат работы программы: содержимое файла «result.txt»

./labyrinth/add/add/file.txt

./labyrinth/add/mul/add/file4.txt

./labyrinth/add/mul/file2.txt

./labyrinth/add/mul/file3.txt

Примечание: программа работает корректно.