**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.**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 3343 |  | Синицкая Д.В. |
| Преподаватель |  | Государкин Я. С. |

Санкт-Петербург

2024

## Цель работы

Освоение работы с рекурсивными функциями и файловой системой, в языке программирования Cи. Создание программы, выполняющей рекурсивный обход файловой системы для поиска файла, содержащего заданное слово.

## 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.

Пример:

Содержимое файла a1.txt

@include a2.txt

@include b5.txt

@include a7.txt

А также файл может содержать тупик:

Содержимое файла a2.txt

Deadlock

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

Функция char \*pathcat(const char \*path1, const char \*path2) нужна для создания путей. Она принимает две строки (пути к файлам), объединяет их и возвращает новую строку-путь.

Функция char \*find\_file(const char \*dir\_name, const char \*filename) нужна для поиска файла в директории. Она рекурсивно ищет файл с заданным именем в указанной директории. Если файл найден, возвращается его полный путь.

Функция int working\_with\_a\_file(const char \*file\_path, char \*\*result) нужна для работы с файлом. Она открывает файл, читает содержимое строки за строкой. Если встречает определенные ключевые слова ("Minotaur", "@include"), то выполняет определенные действия.

В функции main() происходит основная логика программы. Она ищет файл "file.txt" в директории "labyrinth", обрабатывает его содержимое с помощью `working\_with\_a\_file`, и записывает результат в файл "result.txt".

Таким образом, код реализует задачу обработки файлов и директорий, включая рекурсивный поиск и обработку содержимого файлов внутри других файлов (через директиву `@include`).

Разработанный программный код см. в приложении А.

## Выводы

В процессе выполнения лабораторной работы мной были освоены навыки работы с рекурсивными функциями, директориями и файловой системой. Были изучены необходимые языковые конструкции и особенности написания рекурсивных функций, функции для работы с файлами и директориями языка Cи.

# Приложение А Исходный код программы

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <sys/types.h>

//функция для создания путей

char \*pathcat(const char \*path1, const char \*path2)

{

int res\_path\_len = strlen(path1) + strlen(path2) + 2; //определение длины новой строки с учетом символов / и символа конца строки

char \*res\_path = malloc(res\_path\_len \* sizeof(char)); //выделение памяти под новую строку

sprintf(res\_path, "%s/%s", path1, path2); //форматный вывод данных в строку return res\_path;

return res\_path;

}

//функция поиска файла в директории

char \*find\_file(const char \*dir\_name, const char \*filename)

{

char \*full\_path\_file = NULL; //изначально файл не найден

DIR \*dir = opendir(dir\_name); //открытие директории

if (dir)

{

struct dirent \*de = readdir(dir);

while (de)

{

if (de->d\_type == DT\_REG && !strcmp(de->d\_name, filename))

{

//файл найден

full\_path\_file = pathcat(dir\_name, filename);

}

else if (de->d\_type == DT\_DIR && strcmp(de->d\_name, ".") != 0 && strcmp(de->d\_name, "..") != 0)

{

char \*new\_dir = pathcat(dir\_name, de->d\_name);

//запись результата поиска во вложенной директории

full\_path\_file = find\_file(new\_dir, filename);

free(new\_dir);

}

if (full\_path\_file) //файл найден, завершение поиска

break;

de = readdir(dir);

}

closedir(dir); //закрытие директории

}

else

printf("Failed to open %s directory\n", dir\_name);

return full\_path\_file;

}

//функция работы с файлом

int working\_with\_a\_file(const char \*file\_path, char \*\*result)

{

FILE \*file = fopen(file\_path, "r"); //открытие файла на чтение

if (!file)

{

printf("Failed to open %s file\n", file\_path);

exit(0);

}

char information[500];

char \*read\_result = fgets(information, 500, file);

while (read\_result != NULL)

{

if (strcmp(information, "Minotaur") == 0) //поиск слова Minotaur

{

fclose(file); //закрытие файла

return 1; //выход из функции

}

else if (strcmp(information, "Deadlock") == 0) //поиск тупика

{

break; //выход из цикла

}

else

{

//считывание строки, которая начинается с @include

sscanf(information, "@include %s", read\_result); //считывание названия другого файла из ссылки

char \*file\_from\_the\_link = find\_file("labyrinth", information); //поиск файла

if (working\_with\_a\_file(file\_from\_the\_link, result))

{

strcat(\*result, "./"); //запись корня текущего каталога

strcat(\*result, file\_from\_the\_link); //запись полного пути к файлу

strcat(\*result, "\n"); //запись символа переноса строки

free(file\_from\_the\_link); //очистка памяти

fclose(file); //закрытие файла

return 1; //выход из функции

}

free(file\_from\_the\_link); //очистка памяти

}

read\_result = fgets(information, 500, file);

}

fclose(file);

return 0;

}

int main()

{

char \*result = malloc(sizeof(char) \* 3000); //выделение памяти под результат

char \*file = find\_file("labyrinth", "file.txt"); //поиск файла в директории

working\_with\_a\_file(file, &result); //проверка содержимого файла

strcat(result, "./"); //запись корня текущего каталога

strcat(result, file); //запись файла

strcat(result, "\n"); //запись символа переноса строки

int counter = 0; //счётчик

//объединение данных для последующей записи в файл

char \*\*result\_for\_writing\_to\_a\_file = malloc(sizeof(char \*));

char \*token = strtok(result, "\n");

while (token != NULL)

{

result\_for\_writing\_to\_a\_file[counter++] = token;

result\_for\_writing\_to\_a\_file = realloc(result\_for\_writing\_to\_a\_file, sizeof(char \*) \* (counter + 1));

token = strtok(NULL, "\n");

}

//запись результата в файл

FILE \*res\_file = fopen("result.txt", "w");

for (int i = counter - 1; i > -1; i--)

{

if (i == 0)

{

fprintf(res\_file, "%s", result\_for\_writing\_to\_a\_file[i]);

}

else

{

fprintf(res\_file, "%s\n", result\_for\_writing\_to\_a\_file[i]);

}

}

fclose(res\_file); //закрытие файла с результатом

free(result); //очистка памяти

free(result\_for\_writing\_to\_a\_file); //очистка памяти

free(file); //очистка памяти

return 0;

}