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. 3341 _____ Лодыгин И.А.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Целью работы является освоение работы с файлами и рекурсией в языке Си на примере использующей их программы. Для этого требуется изучить библиотеку dirent.h и ознакомиться с принципом работы рекурсии.

Задание

1 вариант.

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.

Пример

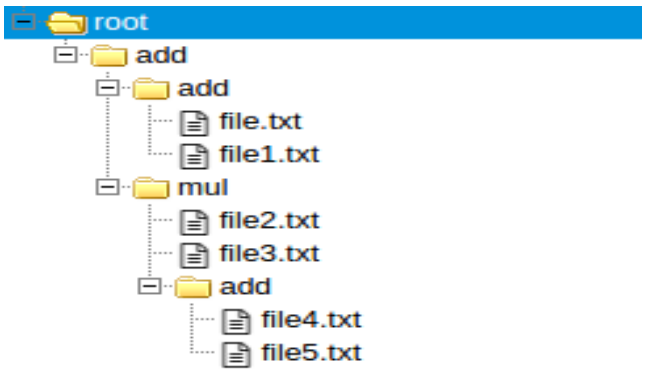

Рисунок 1. Пример вложенных директорий

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

Правильный ответ:

./root/add/add/file.txt

./root/add/mul/add/file4.txt

./root/add/mul/file2.txt

./root/add/mul/file3.txt

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.

Основные теоретические положения

Библиотека dirent.h в языке C предоставляет функции для работы с каталогами и файлами в UNIX-подобных системах. Основные функции, предоставляемые этой библиотекой, включают opendir(), readdir(), closedir() и другие. С их помощью можно открывать каталог, считывать его содержимое и закрывать его.

Рекурсия в языке C - это процесс, при котором функция вызывает саму себя. Рекурсивные функции могут быть использованы для решения задач, которые могут быть разбиты на более простые подзадачи того же типа. Рекурсивные функции обычно имеют базовый случай (base case), который определяет условие завершения рекурсии, и рекурсивный случай (recursive case), который вызывает функцию снова с измененными параметрами.

Выполнение работы

Программа начинает сво выполнение с функции main(). В ней создается указатель targetFilePath и вызывается функция getTargetFilePath("./labyrinth", &targetFilePath, "file.txt"), которая инициализирует путь к целевому файлу.

Функция getTargetFilePath() открывает каталог по указанному пути dirPath и начинает обход всех элементов в нем. Если элемент является файлом и его имя совпадает с targetFile, то функция копирует путь к этому файлу в переменную targetFilePath. Если элемент является каталогом, то функция рекурсивно вызывает саму себя для этого каталога.

После получения пути к целевому файлу, создается структура Data под названием labyrinthPath с помощью функции createData(). В эту структуру записывается путь к целевому файлу с помощью функции pushBack(labyrinthPath, targetFilePath).

Затем вызывается функция crawling(targetFilePath, labyrinthPath), которая открывает файл по указанному пути, считывает его содержимое и проверяет первую строку. Если первая строка равна "Minotaur", то программа выводит результат в файл "result.txt" с помощью функции output("result.txt", labyrinthPath). Если первая строка равна "Deadlock", то программа завершает выполнение. В противном случае программа продолжает обходить содержимое файла, вызывая рекурсивно функцию crawling() для каждого нового пути.

Функция crawling() также использует вспомогательные функции getDataFromFile(), getSubPath() и getNewCopy(), чтобы обрабатывать данные из файлов и создавать копии структуры Data.

Вся программа работает с директориями и файлами, обходя лабиринт (представленный в виде файловой системы) и находя путь к конечному файлу "file.txt" от стартовой директории "./labyrinth".

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	file.txt:@include file1.txt file1.txt:@include file3.txt file2.txt: Deadlock file3.txt: Minotaur file4.txt: fdfd file5.txt: dsgag	*Результат работы записывается в файл result.txt*	Тест moevm
2.	file.txt:@include file4.txt file1.txt: Deadlock file2.txt: hehe file3.txt: lmao file4.txt: Minotaur	*Результат работы записывается в файл result.txt*	В результате обхода директории был найден файл file4.txt, на котором обход закончился

Выводы

Была освоена работа с файлами на языке Си на примере использующей их программы. Произошло ознакомление с реализацией рекурсии для обхода файловой системы.

Результатом работы стала программа, которая при помощи рекурсии обходит директорию, содержащую неизвестное число папок, некоторые из которых вложены в другие.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: solution.c

```
#include <string.h>
#include <dirent.h>
#include <stdio.h>
#include <stdlib.h>

#define CAPACITY 128
#define TERMINAL_ZERO '\0'
#define END_LINE '\n'
#define SLASH '/'
#define CURRENT_DIR "."
#define PARENT_DIR ".."

typedef struct Data {
     int size;
     int capacity;
     char** array;
} Data;

Data* createData() {
     Data* element = (Data*) calloc(1, sizeof(Data));
     return element;
}

char* at(Data* element, int index) {
     return element->array[index];
}

void printData(Data *element) {
     for (int i=0; i<element->size; i++) {
          printf("%s\n", at(element, i));
     }
}

void pushBack(Data* element, char* new_element) {
    if (element->size >= element->capacity) {
        element->capacity = (element->size == 0) ? 2 :
element->capacity*element->capacity;
        element->array = realloc(element->array,
element->capacity*sizeof(char*));
     }
    element->array[element->size++] = new_element;
}

Data* getNewCopy(Data* element, char* new_element) {
     Data* copy = createData();
     for (int i=0; i<element->size; i++) {
          pushBack(copy, at(element, i));
     }
     pushBack(copy, new_element);
     return copy;
```

```
    }

    void getDataFromFile(char* filePath, Data* element) {
        FILE* fin = fopen(filePath, "r");

        char* string = (char*) malloc(CAPACITY*sizeof(char));
        while (fgets(string, CAPACITY, fin)) {
            if (strchr(string, END_LINE)) {
                *(strchr(string, END_LINE)) = TERMINAL_ZERO;
            }
            pushBack(element, string);
            string = (char*) malloc(CAPACITY*sizeof(char));
        }

        fclose(fin);
    }

    char* getSubPath(char* dirPath, char* subName) {
        char* subPath = (char*) calloc((strlen(dirPath) + strlen(subName)
+ 2), sizeof(char));
        strcpy(subPath, dirPath);
        subPath[strlen(dirPath)] = SLASH;
        strcat(subPath, subName);
        return subPath;
    }

    void getTargetFilePath(char* dirPath, char** targetFilePath, char*
targetFile) {
        DIR* dir = opendir(dirPath);
        struct dirent* content = readdir(dir);
        while (content) {
            char* subPath = getSubPath(dirPath, content->d_name);
            if (content->d_type == DT_REG && strcmp(content->d_name,
targetFile) == 0) {
                (*targetFilePath)            =            (char*)
malloc((strlen(subPath)+1)*sizeof(char));
                strcpy(*targetFilePath, subPath);
            }
            if (content->d_type == DT_DIR) {
                if (strcmp(content->d_name, CURRENT_DIR) != 0 &&
strcmp(content->d_name, PARENT_DIR) != 0) {
                    getTargetFilePath(subPath,    targetFilePath,
targetFile);
                }
            }
            content = readdir(dir);
        }
        closedir(dir);
    }

    void output(char* filePath, Data* labyrinthPath) {
        FILE* fout = fopen(filePath, "w");
        char buffer[124];
        for (int i=0; i<labyrinthPath->size; i++) {
            sprintf(buffer, "%s\n", at(labyrinthPath, i));
         fputs(buffer, fout);
         }
        fclose(fout);
```

```
}

void crawling(char* targetFilePath, Data* labyrinthPath, char* targetString, char* resultFile, char* deadlock, char* directoryName) {
        Data* data = createData();
        getDataFromFile(targetFilePath, data);
        if (strcmp(at(data, 0), targetString) == 0) {
            output(resultFile, labyrinthPath);
            return;
        }
        if (strcmp(at(data, 0), deadlock) == 0) {
            return;
        }
        for (int i=0; i<data->size; i++) {
            char* targetFilePath = NULL;
            getTargetFilePath(directoryName, &targetFilePath, strchr(at(data, i), ' ') + 1);

            crawling(targetFilePath, getNewCopy(labyrinthPath, targetFilePath), targetString, resultFile, deadlock, directoryName);
        }
}

int main() {
        char* targetFilePath = NULL;
        getTargetFilePath("./labyrinth", &targetFilePath, "file.txt");
        Data* labyrinthPath = createData();
        pushBack(labyrinthPath, targetFilePath);
        crawling(targetFilePath, labyrinthPath, "Minotaur", "result.txt", "Deadlock", "./labyrinth");
}
```