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: Рекурсия и обход файлового дерева

Студентка гр. 3342 _____ Антипина В.А.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Целью работы является освоение работы с рекурсивными функциями и файловой системой, а также ее рекурсивным обходом.

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.

Пример

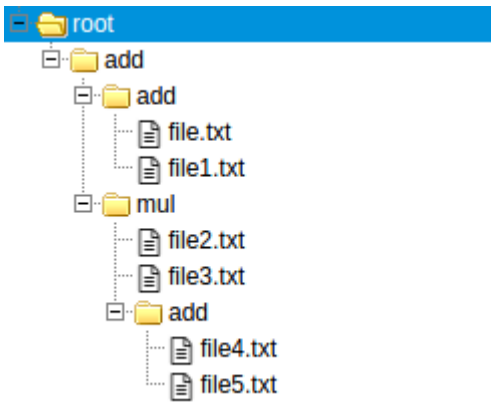

Рисунок 1 — иллюстрация к заданию

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

Рекурсия – это вызов функции в ней самой же. Условия, при которых не происходит рекурсивного вызова, называют условиями выхода. При каждом вызове переменные не перезаписываются, у каждой функции свой набор локальных переменных, который не зависит от других функций. Поэтому когда происходит рекурсивный вызов, то переменные в вызывающей функции останутся неизменными.

Характеристикой рекурсии является ее глубина – количество одновременно запущенных экземпляров рекурсивной функции.

Каждый новый вызов функции требует дополнительного места в «стековой памяти», которая выделяется при запуске программы, для хранения локальных переменных. При достаточно большой глубине рекурсии «стековая память» может закончиться, что вызовет ошибку переполнения стека (Stack Overflow) и аварийное завершение программы. Также каждый вызов функции требует копирования аргументов функции и передачи управления в другую функцию. Данные события требуют дополнительного времени для выполнения, что увеличивает время работы программы, а также не позволяет компилятору применить часть оптимизаций.

Функции для работы с файлами определены в заголовочном файле stdio.h. Для работы с файлами используется файловый поток, реализованный структурой FILE. Напрямую работа с данной структурой не производится, поэтому содержимое структуры рассмотрено не будет. Основными функциями для работы с файлами являются:

- FILE * fopen(const char * filename, const char * mode) – открывает файл с названием filename в режиме mode и возвращает указатель на файловый поток FILE;
- int fclose (FILE * stream) – закрывает файловый поток stream, полученный из fopen(). Основными режимами открытия файла являются “r” и “w”: первый режим открывает файл на чтение, второй – на запись.

Работа с файлами может осуществляться также как и со стандартными потоками ввода и вывода. Для этого существуют функции fprintf, fscanf, fgetc, fgets, fputc, fputs – они работают аналогично своим двойникам для стандартных потоков ввода и вывода, однако принимают дополнительный аргумент в виде указателя на файловый поток FILE. Также есть функции для чтения и записи бинарных данных: fread и fwrite.

Определение структур и функций для работы с директориями находятся в заголовочном файле dirent.h. Для работы с директориями используется поток директории, реализованный структурой DIR (по аналогии с файловым потоком FILE), которая используется в качестве аргумента для функций. Основными функциями для работы с директориями являются:

- DIR *opendir(const char *dirname) – открывает директорию dirname и возвращает указатель на поток директории DIR. Если не удалось открыть директорию, то возвращается NULL;
- int closedir (DIR *dir) – закрывает поток директории dir, который был получен из opendir();
- struct dirent *readdir (DIR *dirstream) – считывает следующий элемент из потока директории dirstream и возвращает указатель на прочитанный элемент. Если в потоке больше не осталось элементов, то возвращается NULL.

Структура dirent содержит информацию о файле. Основную информацию содержат следующие поля:

- поле d_name (тип char[]) – имя файла, которое является строкой, заканчивающаяся символом конца строки;
- поле d_type (тип unsigned char) – тип файла, основными значениями являются: DT_UNKNOWN – неизвестный тип файла; DT_REG – обычный (регулярный) файл, который можно открыть на чтение/запись; DT_DIR – директория. Указатель, который возвращает функция readdir(),

указывает на область памяти, связанную со структурой DIR. Данные в этой области памяти изменяются при каждом вызове функции readdir(). Поэтому если есть необходимость использовать данные из структуры dirent, то их необходимо предварительно скопировать.

Как было упомянуто ранее, директории могут содержать в себе не только файлы, но и другие директории, которые в свою очередь также могут содержать директории и т. д. Таким образом, получается иерархия директорий. В Linux и UNIX-подобных системах корневой директорией, которая не имеет родительской директории, является “/”. Иерархия директорий является воплощением структуры данных дерева.

Выполнение работы

Были подключены стандартные библиотеки stdio.h, string.h, stdlib.h, dirent.h, regex.h, sys/types.h, определена константа STEP.

В функции pathcat, которая получает на вход указатели на массивы символов path1, path2 и возвращает переменную типа char* составляется путь к файлу. Вычисляется суммарная длина итоговой строки с помощью функций strlen, прибавляется 2 (для «/» и символа конца строки). Динамически с помощью функции malloc выделяется память, проверяется, была ли найдена память нужного размера, в строку записываются с помощью функции sprintf поданные на вход pathcat строки через «/».

Функция find_file возвращает переменную того же типа. На вход получает название директории и файла, который нужно в ней найти. full_path_file, переменной типа char*, в которой будет храниться результат, присваивается NULL. Открывается директория с помощью функции readdir. Если открыть папку не удалось, выводится ошибка, возвращается указатель на нуль. Если директорию удалось открыть, то вызывается функция readdir. Если найден файл с нужным названием (это проверяется функцией strcmp), full_path_name присваивается результат выполнения функции pathcat от указателя на директорию и названия файла. Если была найдена директория (и её название не «..» или «.», чтобы избежать зацикливания), создаётся переменная new_dir, в которую записывается результат вызова pathcat — путь к найденной директории. Вызывается функция find_file, на вход ей подаётся новая переменная и название файла, который мы ищем. После этого очищается память из-под new_dir. Если был найден файл, чтение содержимого папки прекращается, директория закрывается. Возвращается full_path_file.

Если файл содержит ссылку на другой файл, она представлена в виде «@include file<>.txt». Чтобы выделить название файла была реализована функция find_filename. Были созданы переменные типа char* prefix, postfix, в которых были записаны постоянные части названий файлов (а именно «file» и

«.txt»), start, end, содержащие указатели на вхождения упомянутых ранее переменных в полученной на вход строке. Если эти указатели ненулевые (то есть имя файла действительно есть в строке; в этой же функции проверяется, что файл нужного типа), выделяется память под массив символов, содержащий название файла (для этого вычисляется длина строки без части @include), проверяется успешное выделение памяти. В выделенную память посимвольно копируются элементы массива символов с названием файла. В конец дописывается символ конца строки. Возвращается созданная строка или ноль, если имя файла не найдено.

В функции read_files осуществляется поиск «Минотавра» путём обхода файлового дерева. Возвращается целое значение. Открывается полученный на вход файл (функции подаётся путь к этому файлу), если открыть файл в переменную start_file не удалось, выводится ошибка, функция возвращает 0. Выделяется память под содержимое файла (переменная filename). Если выделить память не удалось, выводится ошибка. С помощью функции fgetc из файла считываются все символы до символа переноса строки или конца файла, записываются в filename. Если было записано слово «Minotaur», в файл, указатель на который подаётся на вход функции read_files, записываются пути файлов, по которым был найден нужный файл. Возвращается 1. Если было записано слово «Deadlock», освобождается память из-под последнего записанного элемента двумерного массива paths. Этот массив также подаётся на вход функции read_files и содержит информацию о файлах, которые открываются в процессе выполнения программы. Освобождение памяти нужно для того, чтобы массив содержал информацию только о тех файлах, которые содержат слово «Минотавр» или приводят к нему. Вызывается функция find_filename, результат выполнения записывается в переменную filename. Создаётся переменная для хранения пути к файлу, туда записывается результат вызова функции find_file. В массиве paths создаётся новый элемент, туда с помощью функции strcpy сохраняется найденный путь к файлу. Затем

снова вызывается функция read_files, освобождается память из-под переменной, куда был записан путь найденного файла.

В переменную filename данные записываются посимвольно, память выделяется динамически блоками, поэтому в программе предусмотрена ситуация, когда выделенной памяти оказалось недостаточно для считывания строки из файла. В этом случае увеличивается размер массива, вызывается функция realloc и проверяется успешность выполнения вызова последней функции.

В конце чтения файла обнуляется элемент массива paths (чтобы в нём не хранились файлы, приводящие к тупику). Закрывается файл.

В функции main открывается файл result.txt для записи (если его нет, он создаётся). Если открыть файл не удалось, выводится ошибка, возвращается 1. Вызывается функция find_file для поиска пути к файлу file.txt. Выделяется память под двумерный массив символов paths. Объявляется целое число i для его индексации. В файл с ответом записывается адрес file.txt. Вызывается функция read_files, на вход которой подаётся путь к исходному файлу, двумерный массив paths, указатель на целое число i и указатель на открытый файл. Результат сохраняется в переменную p. Закрывается файл. Освобождается память из-под массива paths.

Разработанный программный код см. в приложении А.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	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	./root/add/add/file.txt ./root/add/mul/add/file4.txt ./root/add/mul/file2.txt ./root/add/mul/file3.txt	Корректно

Выводы

Была освоена работа с рекурсивными функциями и файловой системой, а также её рекурсивным обходом. Была реализована программа, которая находит в директории файл, содержащий только слово «Minotaur», и выводит пути файлов, через которые был найден нужный. В работе использовалась стандартная библиотека direct.h и её функции для работы с директориями, а также функции для работы с файлами стандартной библиотеки stdio.h.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Antipina_Veronika_lb3.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <dirent.h>
#include <regex.h>
#include <sys/types.h>
#define STEP 10

int isValid(char* filename){
      char* regexp = "+.txt";
      regex_t regexComp;

      if(regcomp(&regexComp,regexp, REG_EXTENDED))
            return 0;

      return regexec(&regexComp, filename, 0, NULL, 0)==0;
}

char* pathcat(const char* path1, const char* path2){
      int res_path_len = strlen(path1) + strlen(path2) + 2;

      char* res_path = (char*)malloc(res_path_len*sizeof(char));

      sprintf(res_path,"%s/%s",path1,path2);

      return res_path;
}

char* find_file(const char* dir_name, const char* filename){
      char* full_path_file = NULL;
      DIR* dir = opendir(dir_name);
      if(dir){
            struct dirent* de = readdir(dir);
            while(de){
                  if(de->d_type              ==              DT_REG
&& !strcmp(de->d_name,filename) ){
                        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);
                        full_path_file                            =
find_file(new_dir,filename);
                        free(new_dir);
                  }
                  if(full_path_file)
                        break;
                  de = readdir(dir);
            }
            closedir(dir);
```

```
        }else
                printf("Failed to open %s directory\n", dir_name);
        return full_path_file;
    }

    char* find_filename(const char* filename){
          const char* prefix = "file";
          const char* postfix = ".txt";
          const char* start = strstr(filename,prefix);
        const char* end = strstr(filename,postfix);

          if(start!=NULL&&end!=NULL){
                int end = strlen(filename);
                int length = end-9+1;
                char* result = (char*)malloc(length*sizeof(char));
                int k = 0;
                for(int i = 9;i<end;i++)
                        result[k++] = filename[i];
                result[k] = '\0';

                return result;
          }else
                return NULL;
    }

    int read_files(const char* path_st, char** paths, int* i, FILE*
fp){
        FILE* start_file = fopen(path_st,"r");
        if(!start_file){
             puts("Failed to open your file>0<\n");
             return 0;
        }
        char* filename = (char*)malloc(STEP * sizeof(char));

        int index = 0;
        int max_size = STEP;
        char c = EOF;
        do{
             c = fgetc(start_file);
             if((c=='\n'||c==EOF)&&index>0){
                  filename[index] = '\0';

                  if(strcmp(filename,"Minotaur")==0){
                       for(int k = 0;k<(*i)-1;k++)
                            fprintf(fp,"./%s\n",paths[k]);
                       fprintf(fp,"./%s\n",path_st);
                       return 1;
                  }else if(strcmp(filename,"Deadlock")==0){
                       free(paths[--(*i)]);
                       return 0;
                  }
                  filename = find_filename(filename);

                  char* path = find_file("labyrinth",filename);
                  int len = strlen(path);
```

```
                    paths[(*i)]                                                =
(char*)malloc((len+1)*sizeof(char));
                    strcpy(paths[(*i)],path);
                    (*i)++;
                    if(path){
                         int r = read_files(path,paths,i,fp);
                    }else
                         printf("File %s not found\n",filename);

                    free(path);
                    index = 0;
               }else{
                    filename[index++] = c;
               }
               if(index >= max_size){
                    max_size += STEP;
                    filename = realloc(filename,max_size);
               }

          }while(c!=EOF);
          paths[--(*i)] = '\0';
          fclose(start_file);
     }

     int main(){
          FILE *fp = fopen("result.txt","w");
          if(!fp){
               perror("Error");
               return 1;
          }

          char* start_path = find_file("labyrinth","file.txt");
          char** paths = (char**)malloc(3500*sizeof(char*));
          if(!paths){
               perror("Could not find memory");
               return 0;
          }
          int i = 0;

             fprintf(fp,"./%s\n",start_path);

          int p = read_files(start_path,paths,&i,fp);

          fclose(fp);
          for(int j = 0;j<i;j++){
               free(paths[j]);}
          free(paths);

          return 0;
     }
```