МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет безопасности информационных технологий

Дисциплина:
«Основы системного программирования»

Лабораторная работа №1.2 «Работа с файлами, каталогами и плагинами»
Вариант 22

Выполнил:
студент гр. N32511
Копылов Н.М.

Проверил:
Горлина А.В
Подпись:_____

Санкт-Петербург
2023 г.

Цель работы:

Написать программу, позволяющую выполнять рекурсивный поиск совпадений по заданной строке в динамических библиотеках, начиная с указанного каталога, с помощью динамических (разделяемых) библиотек-плагинов.

Описание проекта:

Лабораторная работа предполагает написание программы на языке программирования C, которая будет осуществлять рекурсивный поиск в каталогах при помощи функции nftw() и находить заданные строки в динамических библиотеках. Для выполнения поиска в каталогах будет использоваться функция nftw(), которая позволяет обойти все файлы и подкаталоги в заданном каталоге и его подкаталогах. Для каждого найденного файла программа будет выполнять проверку на наличие заданных байт внутри файла.

Программа на вход принимает 3 аргумента <program_name> <path_to_lib> <option> <substring>. Добавлена возможность использовать переменную окружения LAB1DEBUG. С помощью ее можно вывести отладочную информацию на консоль. А именно вывести точное место, где был найдено совпадение байтов.

Отчет valgrind:

==22786== Memcheck, a memory error detector
==22786== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==22786== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==22786== Command: ./lab12knmN32511 ./libknmN32511.so --dl-sym Nikita .
==22786== HEAP SUMMARY:
==22786== in use at exit: 2,832 bytes in 6 blocks
==22786== total heap usage: 24 allocs, 18 frees, 49,465 bytes allocated
==22786==
==22786== LEAK SUMMARY:
==22786== definitely lost: 0 bytes in 0 blocks
==22786== indirectly lost: 0 bytes in 0 blocks
==22786== possibly lost: 0 bytes in 0 blocks
==22786== still reachable: 2,832 bytes in 6 blocks
==22786== suppressed: 0 bytes in 0 blocks
==22786== Reachable blocks (those to which a pointer was found) are not shown.
==22786== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==22786==
==22786== For lists of detected and suppressed errors, rerun with: -s
==22786== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

Коды файлов проекта:

Makefile:

```
CFLAGS=-Wall -Wextra -Werror -O3
TARGETS=lab12knmN32511 libknmN32511.so

.PHONY: all clean

all: $(TARGETS)

clean:
	rm -rf *.o $(TARGETS)

lab12knmN32511: lab12knmN32511.c plugin_api.h
	gcc $(CFLAGS) -o lab12knmN32511 lab12knmN32511.c -ldl

libknmN32511.so: lib12knmN32511.c plugin_api.h
	gcc $(CFLAGS) -shared -fPIC -o libknmN32511.so lib12knmN32511.c -ldl -lm
```

Код программы(lab12knmN32511.c):

```c
#include <errno.h> // Подключение стандартной библиотеки для работы с кодами ошибок
#include <stdlib.h>// Подключение стандартной библиотеки для работы с памятью и строками
#include <stdio.h>
#include <string.h>
#include <dlfcn.h>// Подключение библиотеки для динамической загрузки библиотек в процессе выполнения программы

#include "plugin_api.h"// Подключение пользовательской библиотеки, которая содержит API для работы с плагинами

int main(int argc, char *argv[]) {
    int opts_to_pass_len = 0; // Определяем переменную для хранения длины массива опций для передачи
    struct option *opts_to_pass = NULL;// Определяем указатель на массив опций для передачи и устанавливаем его в NULL
    struct option *longopts = NULL;// Определяем указатель на массив опций длинных параметров и устанавливаем его в NULL
    opterr = 0;// Устанавливаем флаг, чтобы отключить вывод ошибок getopt()

    // Minimum number of arguments is 4:
    // $ program_name lib_name --opt1 file_to_check
    if (argc < 4) {    // Проверяем, что число аргументов не меньше 4
        fprintf(stdout, "Usage: lab1call /path/to/libabcNXXXX.so [options_for_lib] /path/to/file\n");
        return 0;
    }
    char *lib_name = strdup(argv[1]);// Создаем копию имени библиотеки из первого аргумента
    bool version = 0, help = 0; // Устанавливаем переменные типа bool version и help в false(0)

    int option;// Определяем переменную для хранения текущей опции
    // Выделяем память для массива аргументов командной строки save_argv
    // и копируем туда аргументы из argv
    char** save_argv = calloc(argc, sizeof(char*));
    memcpy(save_argv, argv, argc * sizeof(char*));
    struct flag_option flags_opt = {0}; // Определяем структуру flag_option и инициализируем ее поля нулями
    while ((option = getopt(argc, save_argv, "P:AO:Nvh")) != -1) { // Анализируем аргументы командной строки
        switch (option) {
            case 'P':
                strcpy(lib_name, optarg);
                break;
```

```
35.             case 'A':
36.                 flags_opt.op_and = 1;
37.                 flags_opt.op_or = 0;
38.                 break;
39.             case 'O':
40.                 flags_opt.op_or = 1;
41.                 flags_opt.op_and = 0;
42.                 break;
43.             case 'N':
44.                 flags_opt.op_not = 1;
45.                 break;
46.             case 'v':
47.                 version = 1;
48.                 break;
49.             case 'h':
50.                 help = 1;
51.                 break;
52.             default:
53.                 break;
54.         }
55.     }
56.
57.     if (help) {
58.         printf("Usage: %s [OPTIONS]\n", argv[0]);
59.         printf("Options:\n");
60.         printf("  -P DIR   Specify the directory containing plugins\n");
61.         printf("  -A       Combine plugin options with AND operator (default)\n");
62.         printf("  -O       Combine plugin options with OR operator\n");
63.         printf("  -N       Negate plugin option search condition\n");
64.         printf("  -v       Print version information and exit\n");
65.         printf("  -h       Show this help message and exit\n");
66.         printf("  --dl-sym Checks whether the dynamic library contains the specified
    substring\n");
67.         printf("Example: ./lab12knmN32511 ./libknmN32511.so --dl-sym Nikita .\n");
68.         exit(EXIT_SUCCESS);
69.     }
70.     // Name of the file to analyze. Should be passed as the last argumtent.
71.     char *file_name = strdup(argv[argc-1]);// Создание копии строки с именем файла
72.
73.     struct plugin_info pi = {0};// Создание структуры информации о плагине, инициализация
    нулями
74.
75.     void *dl = dlopen(lib_name, RTLD_LAZY);// Получение дескриптора динамической
    библиотеки
76.     if (!dl) {
77.         fprintf(stderr, "ERROR: dlopen() failed: %s\n", dlerror());
78.         goto END;
79.     }
80.
81.     // Check for plugin_get_info() func
82.     void *func = dlsym(dl, "plugin_get_info"); // Получение указателя на функцию
    plugin_get_info()
83.     if (!func) {
84.         fprintf(stderr, "ERROR: dlsym() failed: %s\n", dlerror());
85.         goto END;
86.     }
87.     // Определение типа функции plugin_get_info() и приведение указателя на функцию к
    этому типу
88.     typedef int (*pgi_func_t)(struct plugin_info*);
89.     pgi_func_t pgi_func = (pgi_func_t)func;
90.     // Вызов функции plugin_get_info() с передачей структуры plugin_info в качестве
    аргумента
91.     int ret = pgi_func(&pi);
92.     if (ret < 0) {
93.         fprintf(stderr, "ERROR: plugin_get_info() failed\n");
94.         goto END;
95.     }
96.
97.     if (version) {
```

```
98.     // Plugin info
99.     fprintf(stdout, "Plugin purpose: %s\n", pi.plugin_purpose);
100.      fprintf(stdout, "Plugin author: %s\n", pi.plugin_author);
101.      fprintf(stdout, "Supported options: ");
102.      if (pi.sup_opts_len > 0) {
103.          fprintf(stdout, "\n");
104.          for (size_t i = 0; i < pi.sup_opts_len; i++) {
105.              fprintf(stdout, "\t--%s\t\t%s\n", pi.sup_opts[i].opt.name,
    pi.sup_opts[i].opt_descr);
106.          }
107.      }
108.      else {
109.          fprintf(stdout, "none (!?)\n");
110.      }
111.      fprintf(stdout, "\n");
112.          exit(EXIT_SUCCESS);
113.      }
114.      // If library supports no options then we have to stop
115.      if (pi.sup_opts_len == 0) { // Проверка на содержание дин. библиотекой опции, если
    нет то выход их программы
116.          fprintf(stderr, "ERROR: library supports no options! How so?\n");
117.          goto END;
118.      }
119.
120.      // Get pointer to plugin_process_file()
121.      func = dlsym(dl, "plugin_process_file");// Получение указателя на функцию
    plugin_process_file()
122.      if (!func) {
123.          fprintf(stderr, "ERROR: no plugin_process_file() function found\n");
124.          goto END;
125.      }
126.      //  создаём тип данных ppf_func_t, который является указателем на функцию,
    принимающую четыре аргумента: строку const char*, массив опций struct option*, размер
    этого массива size_t, и структуру struct
127.      // flag_option*, и возвращающую целочисленное значение.
128.      typedef int (*ppf_func_t)(const char*, struct option*, size_t, struct
    flag_option*);
129.      ppf_func_t ppf_func = (ppf_func_t)func;
130.
131.      // Prepare array of options for getopt_long
132.      longopts = calloc(pi.sup_opts_len + 1, sizeof(struct option));
133.      if (!longopts) {
134.          fprintf(stderr, "ERROR: calloc() failed: %s\n", strerror(errno));
135.          goto END;
136.      }
137.
138.      // Copy option information
139.      for (size_t i = 0; i < pi.sup_opts_len; i++) {
140.          // Mind this!
141.          // getopt_long() requires array of struct option in its longopts arg,
142.          // but pi.sup_opts is array of  structs, not option structs.
143.          memcpy(longopts + i, &pi.sup_opts[i].opt, sizeof(struct option));
144.      }
145.
146.      // Prepare array of actually used options that will be passed to
147.      // plugin_process_file() (Maximum pi.sup_opts_len options)
148.      opts_to_pass = calloc(pi.sup_opts_len, sizeof(struct option));
149.      if (!opts_to_pass) {
150.          fprintf(stderr, "ERROR: calloc() failed: %s\n", strerror(errno));
151.          goto END;
152.      }
153.
154.      // Now process options for the lib
155.      while (1) {
156.          int opt_ind = 0;
157.          ret = getopt_long(argc, argv, "", longopts, &opt_ind);
158.          if (ret == -1) break;
159.
160.          if (ret != 0) {
```

```
161.             fprintf(stderr, "ERROR: failed to parse options\n");
162.             goto END;
163.         }
164.
165. #ifndef ALLOW_OPT_ABBREV
166.         // glibc quirk: no proper way to disable option abbreviations
167.         // https://stackoverflow.com/questions/5182041/turn-off-abbreviation-in-getopt-
    long-optarg-h
168.         int idx = (longopts + opt_ind)->has_arg ? 2 : 1;
169.         const char *actual_opt_name = argv[optind - idx] + 2; // +2 for -- before
    option
170.         const char *found_opt_name = (longopts + opt_ind)->name;
171.         if (strcmp(actual_opt_name, found_opt_name)) {
172.             // It's probably abbreviated name, which we do not allow
173.             fprintf(stderr, "ERROR: unknown option: %s\n", argv[optind - idx]);
174.             goto END;
175.         }
176. #endif
177.
178.         // Check how many options we got up to this moment
179.         if ((size_t)opts_to_pass_len == pi.sup_opts_len) {
180.             fprintf(stderr, "ERROR: too many options!\n");
181.             goto END;
182.         }
183.
184.         // Add this option to array of options actually passed to plugin_process_file()
185.         memcpy(opts_to_pass + opts_to_pass_len, longopts + opt_ind, sizeof(struct
    option));
186.         // Argument (if any) is passed in flag
187.         if ((longopts + opt_ind)->has_arg) {
188.             // Mind this!
189.             // flag is of type int*, but we are passing char* here (it's ok to do so).
190.             (opts_to_pass + opts_to_pass_len)->flag = (int*)strdup(optarg);
191.         }
192.         opts_to_pass_len++;
193.     }
194.
195.     if (getenv("LAB1DEBUG")) {
196.         fprintf(stderr, "DEBUG: opts_to_pass_len = %d\n", opts_to_pass_len);
197.         for (int i = 0; i < opts_to_pass_len; i++) {
198.             fprintf(stderr, "DEBUG: passing option '%s' with arg '%s'\n",
199.                 (opts_to_pass + i)->name,
200.                 (char*)(opts_to_pass + i)->flag);
201.         }
202.     }
203.
204.     // Call plugin_process_file()
205.     errno = 0;
206.     ret = ppf_func(file_name, opts_to_pass, opts_to_pass_len, &flags_opt);
207.     fprintf(stdout, "\nplugin_process_file() returned %d\n", ret);
208.     if (ret < 0) {
209.         fprintf(stdout, "Error information: %s\n", strerror(errno));
210.     }
211.
212.     END:
213.     if (opts_to_pass) {
214.         for (int i = 0; i < opts_to_pass_len; i++)
215.             free( (opts_to_pass + i)->flag );
216.         free(opts_to_pass);
217.     }
218.     if (save_argv) free(save_argv);
219.     if (longopts) free(longopts);
220.     if (lib_name) free(lib_name);
221.     if (file_name) free(file_name);
222.     if (dl) dlclose(dl);
223.
224.     return 0;
225. }
```

Код программы(libknmN32511.c):

```c
#define _GNU_SOURCE // Определение _GNU_SOURCE добавляет определения функции, таких как функции getline() и asprintf()/
#define _XOPEN_SOURCE 500
#include <dirent.h>   // для работы с каталогами
#include <stdlib.h>   // для работы с функциями стандартной библиотеки
#include <string.h>   // для работы со строками
#include <math.h>     // для математических операций
#include <sys/mman.h> // для работы с отображением файлов в память
#include <sys/types.h>// для работы с типами данных, связанными с системными вызовами
#include <sys/stat.h> // для работы с информацией о файлах
#include <sys/param.h> // для использования макроса MIN()
#include <fcntl.h>    // для работы с файловыми дескрипторами
#include <unistd.h>   // для работы с системными вызовами Unix
#include <errno.h>    // для работы с кодами ошибок
#include <ftw.h>      // для работы с файловой системой
#include <stdio.h>    // для работы со стандартным вводом/выводом

#include "plugin_api.h"

#define MAX_INDENT_LEVEL 128 // Глубина рекурсии

static char *g_lib_name = "libknmN32511.so"; //Название библиотеки

static char *g_plugin_purpose = "Search for dynamic libraries containing all the specified symbols"; // Цель плагина

static char *g_plugin_author = "Kopylov Nikita"; // Имя автора плагина

#define OPT_DL_SYM "dl-sym" // Описание опции --dl-sym
const char* search_strings[10];
int lenght = 0;
char *token;
char temp[1024];
/*Массив структур для, каждая из которых содержит опцию плагина для программы. Она содержит два поля структуру option, а также поле *opt_descr
    struct plugin_option {
        struct option {
           const char *name;
           int         has_arg;
           int        *flag;
           int         val;
        } opt,
        char *opt_descr
    }
    Поле opt является структурой option, которая содержит информацию об опции:

            name - название опции;
            has_arg - указывает, требует ли опция аргумент;
            flag - указатель на переменную, которая будет установлена в значение val, если опция будет передана;
            val - значение, которое будет установлено в flag, если опция будет передана.
*/

static struct plugin_option g_po_arr[] = {
    {
        {
            OPT_DL_SYM,
            required_argument,
            0, 0,
        },
        "string"
    },
};

static int g_po_arr_len = sizeof(g_po_arr)/sizeof(g_po_arr[0]); // Длина массива структур
//
```

```c
//  Private functions
//
    void walk_dir(const char *dir); // Обход директории
    int search_function(const char* filepath, const char** search_strings, int num_strings); // Функция поиска подстроки
    struct flag_option flags = {0}; // структура для флагов опций
//
//  API functions
//
// Функция plugin_get_info заполняет структуру struct plugin_info переданными значениями,
//полученными из глобальных переменных g_plugin_purpose, g_plugin_author, g_po_arr_len и g_po_arr.
int plugin_get_info(struct plugin_info* ppi) {
    if (!ppi) {
        fprintf(stderr, "ERROR: invalid argument\n");
        return -1;
    }

    ppi->plugin_purpose = g_plugin_purpose;
    ppi->plugin_author = g_plugin_author;
    ppi->sup_opts_len = g_po_arr_len;
    ppi->sup_opts = g_po_arr;

    return 0;
}

// Макрос длля проверки //
int plugin_process_file(const char *fname,//  Указатель на строку, содержащую имя обрабатываемого файла.
        struct option in_opts[], //  Массив опций, которые будут использоваться для обработки файла
        size_t in_opts_len, // Размер массива
        struct flag_option* flag)  //  Указатель на структуру, содержащую флаги, которые будут использоваться для обработки файла
         {
// Инициализируем флаги, используя переданный указатель на структуру flag_option
    flags.op_and = flag->op_and;
    flags.op_or = flag->op_or;
    flags.op_not = flag->op_not;
    char *debug = getenv("LAB1DEBUG");
// Проверяем переданные параметры на валидность
    if (!fname || !in_opts || !in_opts_len) {
          errno = EINVAL;
          return -1;
     }

     if (debug) {
         for (size_t i = 0; i < in_opts_len; i++) {
             fprintf(stderr, "DEBUG: %s: Got option '%s' with arg '%s'\n",
                 g_lib_name, in_opts[i].name, (char*)in_opts[i].flag);
         }
     }
     int got_operator = 0;
 // Проверяем, была ли уже передана опция с заданным именем
     for (size_t i = 0; i < in_opts_len; i++) {
         if (!strcmp(in_opts[i].name, OPT_DL_SYM)) {

             if (got_operator) {
                 if(debug){
                         fprintf(stderr, "DEBUG: %s: Option '%s' was already supplied\n",
                         g_lib_name, in_opts[i].name);
                 }
                 errno = EINVAL;
                 return -1;

             }
             else {
```

```
125.                 // Если опция еще не была передана, копируем значение во временный
    буфер
126.             strcpy(temp, (char*)in_opts[i].flag);
127.             got_operator=1;
128.
129.         }
130.         }
131.
132.     }
133.     // Разбиваем значение, переданное в опции, на отдельные строки и сохраняем их в
    массив search_strings
134.     int i = 0;
135.     token = strtok(temp, ",");
136.     while (token != NULL){
137.         search_strings[i] = token;
138.         i++;
139.         token = strtok(NULL, ",");
140.     }
141.     lenght = i;
142.     // Обрабатываем файлы
143.     walk_dir(fname);
144.     return 0;
145. }
146.
147. void searching_func(const char* filepath, const char** search_strings, int num_strings)
    {
148.     FILE* fp = fopen(filepath, "r");
149.     char buffer[1024];
150.     char* debug = getenv("LAB1DEBUG");
151.     char* extension = strrchr(filepath, '.'); // Получаем расширение файла
152.     if (extension && !strcmp(extension, ".so")){ // Проверка на нужное расширение (.so)
153.         if (fp == NULL) {
154.             printf("Cannot open file %s\n", filepath);
155.             return;
156.         }
157.         int count = 0;
158.         int read_len = 0;
159.         if (flags.op_not == false){ // Проверка на флаг инфертирования
160.         while ((read_len = fread(buffer, 1, sizeof(buffer), fp)) > 0) { // Цикл для
    считывания данных из файла
161.             for (int i = 0; i < num_strings; ++i)
162.             {
163.                 int tmp_leng = strlen(search_strings[i]);
164.                 for (int j = 0; j < (read_len - tmp_leng); j++) { // Цикл для обработки
    данных из файла
165.                     if (memcmp(buffer + j, search_strings[i],tmp_leng) == 0) { //
    Сравнение данных из файла и нужной подстроки
166.                         count++; // Счетчик++
167.                         if (debug != NULL){ //
168.                             printf("Found string '%s' at offset %ld\n",
    search_strings[i], ftell(fp) - read_len);
169.                         }
170.                     }
171.             }
172.         }
173.         }
174.     }
175.     if (flags.op_not == true){
176.          while ((read_len = fread(buffer, 1, sizeof(buffer), fp)) > 0) { // Цикл для
    считывания данных из файла
177.             for (int i = 0; i < num_strings; ++i)
178.             {
179.                 int tmp_leng = strlen(search_strings[i]);
180.                 for (int j = 0; j < (read_len - tmp_leng); j++) { // Цикл для обработки
    данных из файла
181.                     if (memcmp(buffer + j, search_strings[i],tmp_leng) != 0) { //
    Сравнение данных из файла и нужной подстроки
182.                         //printf("%2s equal %s \n",buffer + j,search_strings[i]);
183.                         count++; // Счетчик++
```

```
                    }
                }
            }
            }
        }
            fclose(fp);
            if (debug != NULL){
                for (int i = 0; i < num_strings; i++) {
                    if (count == 0) {
                        printf("Did not find string '%s' in all libraries\n",
    search_strings[i]);
                        return;
                    }
                }
            }
            if (count != 0 && flags.op_not == false){
                printf("Strings ");
            for (int i = 0; i < num_strings; i++){
                printf("\"%s\" " , search_strings[i]);
            }
            printf(" was founded %d times in %s", count, filepath);
            }
            else if (count != 0 && flags.op_not == true){

                printf("The number of times how many rows were found that are not equal to
    the original %d in %s\n", count, filepath);
            }
        }
        else return;

    }




 void print_entry(int level __attribute__((unused)), int type __attribute__((unused)),
    const char *path) {
        if (!strcmp(path, ".") || !strcmp(path, "..")) // Проверяем, является ли запись
    текущей директорией или родительской директорией, если да, то пропускаем
            return;
        searching_func(path, search_strings, lenght); // Вызываем функцию поиска
    searching_func для поиска байтовой последовательности в содержимом файла
    }

    int walk_func(const char *fpath, const struct stat *sb __attribute__((unused)), int
    typeflag, struct FTW *ftwbuf) {
        print_entry(ftwbuf->level, typeflag, fpath); // Функция для обработки текущей
    записи
        return 0;
    }

    void walk_dir(const char *dir) {
        int res = nftw(dir, walk_func, 10, FTW_PHYS); // Вызов функции nftw() для обхода
    директории
        if (res < 0) { // Если директория не была найдена, выводим сообщение об ошибке
            fprintf(stderr, "ntfw() failed: %s\n", strerror(errno));
        }
    }
```

Код программы(plugin_api.h):

```
1.  #ifndef _PLUGIN_API_H
```

```
2.  #define _PLUGIN_API_H
3.
4.  #include <getopt.h>
5.  #include <stdbool.h>
6.
7.  /*
8.      Структура, описывающая опцию, поддерживаемую плагином.
9.  */
10. struct plugin_option {
11.     /* Опция в формате, поддерживаемом getopt_long (man 3 getopt_long). */
12.     struct option opt;
13.     /* Описание опции, которое предоставляет плагин. */
14.     const char *opt_descr;
15. };
16.
17.
18. struct flag_option {
19.     bool op_and;
20.     bool op_or;
21.     bool op_not;
22. };
23.
24. /*
25.     Структура, содержащая информацию о плагине.
26. */
27. struct plugin_info {
28.     /* Назначение плагина */
29.     const char *plugin_purpose;
30.     /* Автор плагина, например "Иванов Иван Иванович, N32xx" */
31.     const char *plugin_author;
32.     /* Длина списка опций */
33.     size_t sup_opts_len;
34.     /* Список опций, поддерживаемых плагином */
35.     struct plugin_option *sup_opts;
36. };
37.
38.
39. int plugin_get_info(struct plugin_info* ppi);
40.
41.
42.
43. int plugin_process_file(const char *fname,
44.         struct option in_opts[],
45.         size_t in_opts_len,
46.         struct flag_option* flags);
47. #endif
```

Скриншоты работы программы:

1. **Make all**

```
balora@root:~/OSP/lab1.2$ make all
gcc -Wall -Wextra -Werror -O3  -o lab12knmN32511 lab12knmN32511.c -ldl
gcc -Wall -Wextra -Werror -O3  -shared -fPIC -o libknmN32511.so lib12knmN32511.c -ldl -lm
```

2. **Make clean**

```
balora@root:~/OSP/lab1.2$ make all
gcc -Wall -Wextra -Werror -O3  -o lab12knmN32511 lab12knmN32511.c -ldl
gcc -Wall -Wextra -Werror -O3  -shared -fPIC -o libknmN32511.so lib12knmN32511.c -ldl -lm
balora@root:~/OSP/lab1.2$ make clean
rm -rf *.o lab12knmN32511 libknmN32511.so
```

3. **Опции (--dl-sym)**

```
balora@root:~/OSP/lab1.2$ ./lab12knmN32511 ./libknmN32511.so --dl-sym Nikita .
Strings "Nikita"  was founded 1 times in ./libknmN32511.so
plugin_process_file() returned 0
```

4. **Опции (-N)**

```
balora@root:~/OSP/lab1.2$ ./lab12knmN32511 ./libknmN32511.so -N --dl-sym Nikita .
The number of times how many rows were found that are not equal to the original 16897 in ./libknmN32511.so

plugin_process_file() returned 0
```

5. **Пример работы программы с переменной окружения**

```
balora@root:~/OSP/lab1.2$ export LAB1DEBUG=1
balora@root:~/OSP/lab1.2$ ./lab12knmN32511 ./libknmN32511.so --dl-sym Nikita,plugin .
DEBUG: opts_to_pass_len = 1
DEBUG: passing option 'dl-sym' with arg 'Nikita,plugin'
DEBUG: libknmN32511.so: Got option 'dl-sym' with arg 'Nikita,plugin'
Found string 'plugin' at offset 1024
Found string 'plugin' at offset 1024
Found string 'Nikita' at offset 8192
Found string 'plugin' at offset 13312
Found string 'plugin' at offset 14336
Strings "Nikita" "plugin"  was founded 5 times in ./libknmN32511.so
plugin_process_file() returned 0
```

Вывод: в ходе лабораторной работы создал программу для рекурсивного обхода директорий при помощи функции nftw(). Также познакомился с написанием собственного плагина на языке C. И научился пользоваться разделяемыми библиотеками.