**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»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2**

«Лабораторная 2»

**Выполнили:**

Ахраров Али, студент группы N3250

**Проверил:**

Грозов Владимир Андреевич

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(отметка о выполнении)

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

# Вариант 18

# Задание:

Разработать на языке C для ОС Linux:

программу, позволяющую выполнять рекурсивный поиск файлов, начиная с

указанного каталога, с помощью динамических (разделяемых) библиотек-плагинов

(использовать в качестве основы программу, разработанную в лабораторной работе

1.1);

динамическую библиотеку, реализующую заданный вариантом лабораторной работы

из Табл. 4 критерий поиска файлов.

Программа должна представлять собой консольную утилиту, настройка работы

которой осуществляется путем передачи аргументов в строке запуска и/или с помощью

переменных окружения:

lab12abcNXXXXX [опции] [каталог]

Программа должна выполнять рекурсивный поиск файлов, отвечающих критериям,

которые задаются опциями в командной строке. Доступные критерии поиска (и,

соответственно, доступные опции) определяются наличием в заданном каталоге

динамических библиотек, расширяющих функциональность программы (далее — плагинов).

# Makefile

CFLAGS=-Wall -Wextra -Werror

TARGETS := lab12 lab12lib

all: $(TARGETS)

clean:

rm -rf \*.o lab12araN3250 libaraN3250.so

lab12: main.c plugin\_api.h

gcc $(CFLAGS) -o lab12araN3250 main.c -ldl -g

lab12lib: libaraN3250.c plugin\_api.h

gcc $(CFLAGS) -shared -fPIC -o libaraN3250.so libaraN3250.c -ldl -lm -g

# Отчет valgrind:

==8223== HEAP SUMMARY:

==8223== in use at exit: 0 bytes in 0 blocks

==8223== total heap usage: 20 allocs, 20 frees, 75,410 bytes allocated

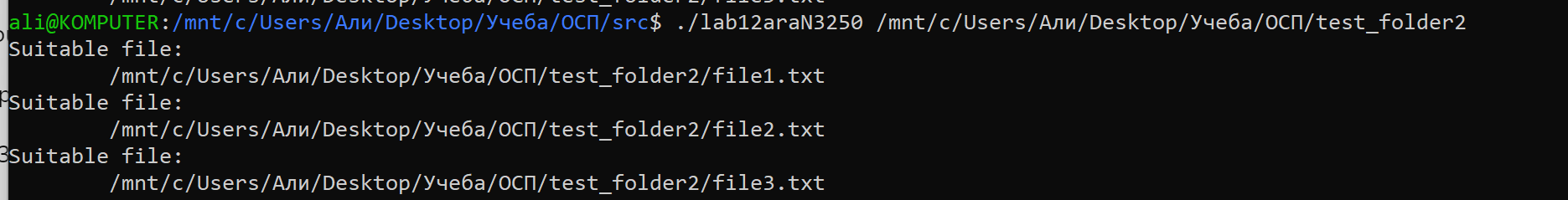
==8223==

==8223== All heap blocks were freed -- no leaks are possible

==8223==

==8223== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

# Скриншоты работы:



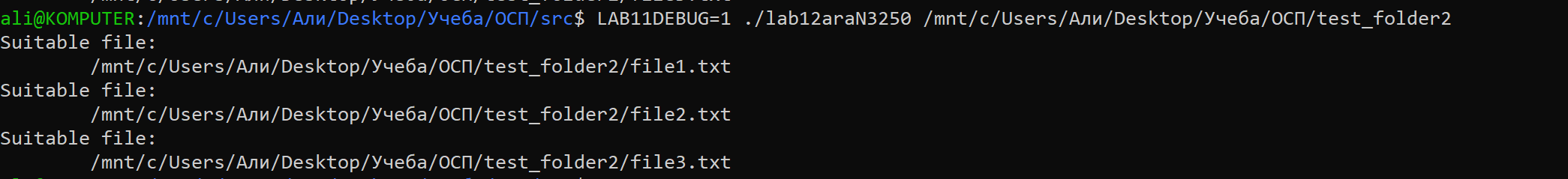


Рис.1 – скриншот работы программы

# Исходные тексты программ с комментариями.

#define \_XOPEN\_SOURCE 500

#define \_DEFAULT\_SOURCE

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include <ftw.h>

#include <stdint.h>

#include <string.h>

#include <getopt.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <linux/limits.h>

#include <dirent.h>

#include <stddef.h>

#include <sys/types.h>

#include <dlfcn.h>

#include "plugin\_api.h"

char\* DEBUG;

//0 - логическое И, 1 - логическое ИЛИ

int mode\_and\_or = 0;

//Логическое НЕ

int mode\_not = 0;

//Структура плагина

struct plugin\_helper {

void\* plugin\_so\_lib;

char\* plugin\_name;

int (\*plugin\_process\_file)(const char \*fname, struct option in\_opts[],size\_t in\_opts\_len);

struct plugin\_info\* p\_info;

struct option\* taken\_opts;

int taken\_opts\_counter;

};

//Список найденных плагинов

struct plugin\_helper\* found\_plugins;

//Счётчик найденных плагинов

int found\_counter = 0;

//Список всех опций от всех плагинов

struct option\* defined\_opts;

//Счётчик этих опций

int defined\_opts\_counter = 0;

//Путь до плагинов

char\* plugins\_dir\_path;

struct option long\_options[] = {

{"help", no\_argument, NULL, 'h'},

{"version", no\_argument, NULL, 'v'},

{NULL, 0, NULL, 0}

};

static int app\_long\_options\_len = sizeof(long\_options) / sizeof(long\_options[0]);

const char\* short\_options = "P:AONvh";

void free\_lists();

void print\_help();

int check\_plugins\_compatibility();

int load\_plugins(char\* plugins\_path);

int filter\_function(const char\* fpath, const struct stat\* sb, int typeflag, struct FTW\* ftwbuf);

int main(int argc, char\* argv[]){

DEBUG = getenv("LAB1DEBUG");

char plugins\_dir\_path[PATH\_MAX];

char\* r = getcwd(plugins\_dir\_path, PATH\_MAX);

r = r;

for(int i = 0; i < argc; i++){

if(argv[i][0] == '-' && argv[i][1] == 'P' && i + 1 < argc && argv[i + 1][0] != '-'){

strcpy(plugins\_dir\_path, argv[i + 1]);

break;

}

}

if(DEBUG){

fprintf(stderr, "Dir with plugins:\n\t%s\n", plugins\_dir\_path);

}

int l = load\_plugins(plugins\_dir\_path);

if(l != 0){

fprintf(stderr, "Error occured while importing plugins");

free\_lists();

exit(EXIT\_FAILURE);

}

defined\_opts = realloc(defined\_opts, sizeof(struct option) \* (defined\_opts\_counter + app\_long\_options\_len));

//Для работы с опциями плагинов в getopt\_long

//Необходимо установить идентификаторы опциям

int id = app\_long\_options\_len + 1;

for(int i = 0; i < defined\_opts\_counter; i++){

defined\_opts[i].val = id + i;

}

id = app\_long\_options\_len + 1;

for(int i = 0; i < found\_counter; i++){

for(int j = 0; j < (int)found\_plugins[i].p\_info->sup\_opts\_len; j++){

found\_plugins[i].p\_info->sup\_opts[j].opt.val = id;

id++;

}

}

defined\_opts[defined\_opts\_counter] = long\_options[0];

defined\_opts[defined\_opts\_counter + 1] = long\_options[1];

defined\_opts[defined\_opts\_counter + 2] = long\_options[2];

int opt;

int option\_index = -1;

while((opt = getopt\_long(argc, argv, short\_options, defined\_opts, &option\_index)) != -1){

switch (opt)

{

case 'h':

print\_help();

free\_lists();

return 0;

case 'v':

printf("lab12araN3250 version 1.\nAhrarov Ali Rustamovich N3250\n18-th variant\n\n");

free\_lists();

return 0;

case 'O':

mode\_and\_or = 1;

break;

case 'N':

mode\_not = 1;

break;

case 'A':

mode\_and\_or = 0;

break;

}

int found = 0;

for(int i = 0; i < found\_counter; i++){

for(int j = 0; j < (int)found\_plugins[i].p\_info->sup\_opts\_len; j++){

struct option expected\_option = found\_plugins[i].p\_info->sup\_opts[j].opt;

int expected\_option\_value = expected\_option.val;

if(expected\_option\_value == opt){

if(expected\_option.has\_arg > 0){

if(!optarg){

fprintf(stderr, "Option %s requires an argument\n", expected\_option.name);

free\_lists();

exit(EXIT\_FAILURE);

}

expected\_option.flag = (int\*)optarg;

}

if(found\_plugins[i].taken\_opts\_counter == 0){

found\_plugins[i].taken\_opts = malloc(sizeof(struct option));

}else{

found\_plugins[i].taken\_opts = realloc(found\_plugins[i].taken\_opts, sizeof(struct option) \* (found\_plugins[i].taken\_opts\_counter + 1));

}

memcpy(&found\_plugins[i].taken\_opts[found\_plugins[i].taken\_opts\_counter], &expected\_option, sizeof(struct option));

found\_plugins[i].taken\_opts\_counter++;

found = 1;

break;

}

if(found){

break;

}

}

}

if(opt == '?'){

fprintf(stderr, "Bad optinon\n\n");

print\_help();

free\_lists();

exit(EXIT\_FAILURE);

}

option\_index = -1;

}

char\* path = argv[optind];

if(!path){

print\_help();

free\_lists();

return 0;

}

nftw(path, filter\_function, 5, FTW\_DEPTH | FTW\_PHYS);

free\_lists();

return 0;

}

void free\_lists(){

for(int i = 0; i < found\_counter; i++){

dlclose(found\_plugins[i].plugin\_so\_lib);

if(found\_plugins[i].taken\_opts\_counter > 0){

free(found\_plugins[i].taken\_opts);

}

free(found\_plugins[i].plugin\_name);

free(found\_plugins[i].p\_info);

}

if(found\_counter > 0){

free(found\_plugins);

}

if(defined\_opts\_counter > 0){

free(defined\_opts);

}

}

void print\_help(){

printf("Usage: ./lab12araN3250 [options...] <dir\_path>\n\t%s: %s\n\t%s: %s\n\t%s: %s\n\t%s: %s\n\t%s: %s\n\t%s: %s\n",

"-h, --help",

"Get help for commands",

"-v, --version",

"Show version number and app information",

"-A",

"Use AND (by default)",

"-O",

"Use OR",

"-N",

"Use NOT",

"-P <dir>",

"Set folder with plugins to use"

);

printf("Imported options:\n");

for(int i = 0; i < found\_counter; i++){

struct plugin\_helper plugin = found\_plugins[i];

struct plugin\_info\* ppi = plugin.p\_info;

printf("\t(%d) Name: %s\n", (i + 1), plugin.plugin\_name);

for(int j = 0; j < (int)ppi->sup\_opts\_len; j++){

struct option opt = ppi->sup\_opts[j].opt;

const char\* description = ppi->sup\_opts[j].opt\_descr;

printf("\t\t(%d) %s - %s\n", (j + 1), opt.name, description);

if(opt.has\_arg){

printf("\t\tReqires an argument\n");

}

}

}

}

int load\_plugins(char\* plugins\_path){

DIR\* d;

struct dirent\* dir;

d = opendir(plugins\_path);

if(!d){

fprintf(stderr, "Impossible to open directory with plugins:\n\t%s\nSTOP\n", plugins\_path);

free\_lists();

return -1;

}

char buf[PATH\_MAX];

while((dir = readdir(d)) != NULL){

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

int dname\_len = strlen(dir->d\_name);

//Действительно ли файл - плагин?

if(dname\_len < 3 || dir->d\_name[dname\_len-3] != '.' || dir->d\_name[dname\_len-2] != 's' || dir->d\_name[dname\_len-1] != 'o'){

continue;

}

//Собираем путь до плагина

strcpy(buf, plugins\_path);

if(buf[strlen(buf) - 1] != '/'){

strcat(buf, "/");

}

strcat(buf, dir->d\_name);

void\* lib = dlopen(buf, RTLD\_NOW);

if(lib){

//Если действительно...

if(!found\_plugins){

found\_plugins = malloc(sizeof(struct plugin\_helper));

}

else {

found\_plugins = realloc(found\_plugins, sizeof(struct plugin\_helper) \* (found\_counter + 1));

}

int (\*plugin\_info)(struct plugin\_info\*);

plugin\_info = dlsym(lib, "plugin\_get\_info");

int (\*plugin\_process\_file)(const char \*fname, struct option in\_opts[],size\_t in\_opts\_len);

plugin\_process\_file = dlsym(lib, "plugin\_process\_file");

struct plugin\_helper temp;

temp.p\_info = malloc(sizeof(struct plugin\_info));

(\*plugin\_info)(temp.p\_info);

temp.plugin\_so\_lib = lib;

temp.plugin\_process\_file = plugin\_process\_file;

temp.taken\_opts\_counter = 0;

temp.taken\_opts = NULL;

//Копируем плагин в массив плагинов

memcpy(&found\_plugins[found\_counter], &temp, sizeof(struct plugin\_helper));

found\_plugins[found\_counter].plugin\_name = malloc(sizeof(char) \* (strlen(dir->d\_name) + 1));

strcpy(found\_plugins[found\_counter].plugin\_name, dir->d\_name);

found\_counter++;

if(DEBUG){

fprintf(stderr, "(%d) Imported plugin:\n\t%s\n", found\_counter, dir->d\_name);

}

}

}

}

closedir(d);

if(check\_plugins\_compatibility() == -1){

return -1;

}

for(int i = 0; i < found\_counter; i++){

struct plugin\_info\* plugin\_info = found\_plugins[i].p\_info;

if(!defined\_opts){

defined\_opts = malloc(sizeof(struct option) \* plugin\_info->sup\_opts\_len);

}

else{

defined\_opts = realloc(defined\_opts, sizeof(struct option) \* (defined\_opts\_counter + plugin\_info->sup\_opts\_len));

}

for(int j = 0; j < (int)plugin\_info->sup\_opts\_len; j++){

defined\_opts[defined\_opts\_counter + j] = plugin\_info->sup\_opts[j].opt;

}

defined\_opts\_counter += plugin\_info->sup\_opts\_len;

}

return 0;

}

int check\_plugins\_compatibility(){

for(int i = 0; i < found\_counter; i++){

struct plugin\_info\* plugin\_info\_1 = found\_plugins[i].p\_info;

for(int j = i + 1; j < found\_counter; j++){

struct plugin\_info\* plugin\_info\_2 = found\_plugins[j].p\_info;

for(int k = 0; k < (int)plugin\_info\_1->sup\_opts\_len; k++){

for(int s = 0; s < (int)plugin\_info\_2->sup\_opts\_len; s++){

if(strcmp(plugin\_info\_1->sup\_opts[k].opt.name, plugin\_info\_2->sup\_opts[s].opt.name) == 0){

fprintf(stderr, "Error! Two plugins have the same options:\n");

fprintf(stderr, "%s\n", found\_plugins[i].plugin\_name);

fprintf(stderr, "%s\n", found\_plugins[j].plugin\_name);

fprintf(stderr, "Remove one and restart\n");

free\_lists();

return -1;

}

}

}

}

}

return 0;

}

int filter\_function(const char\* fpath, const struct stat\* sb, int typeflag, struct FTW\* ftwbuf) {

fpath = fpath;

sb = sb;

typeflag = typeflag;

ftwbuf = ftwbuf;

if(typeflag == FTW\_F){

int suits = !mode\_and\_or;

int (\*plugin\_process\_file)(const char \*fname, struct option in\_opts[],size\_t in\_opts\_len);

for(int i = 0; i < found\_counter; i++){

if(found\_plugins[i].taken\_opts\_counter > 0){

plugin\_process\_file = found\_plugins[i].plugin\_process\_file;

int r = (\*plugin\_process\_file)(fpath, found\_plugins[i].taken\_opts, found\_plugins[i].taken\_opts\_counter);

if(r < 0){

fprintf(stderr, "Error in plugin %s\nStop app working\n", found\_plugins[i].plugin\_name);

return -1;

}

if(DEBUG){

fprintf(stderr, "Result for file:\n\t%s\nPlugin\n\t%s = %d\n", fpath, found\_plugins[i].plugin\_name, r);

}

if(mode\_and\_or == 0){

suits &= (!r && !mode\_not) || (r && mode\_not);

if(!suits){

break;

}

}

else{

suits |= (!r && !mode\_not) || (r && mode\_not);

}

}

}

if(suits){

printf("Suitable file:\n\t%s\n", fpath);

}

}

return 0;

}