

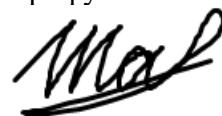
МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Санкт-Петербургский национальный исследовательский университет
ИТМО»

ФАКУЛЬТЕТ БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторным работам 1, 2

Работу выполнил
студент
группы N3149
очного отделения
Шарифуллин И.А.



Проверено преподавателем

(Грозов В. А.)



УНИВЕРСИТЕТ ИТМО

Санкт-Петербург, 2021

Задание:

Разработать на языке C для ОС Linux:

- программу, позволяющую выполнять рекурсивный поиск файлов, начиная с указанного каталога, с помощью динамических (разделяемых) библиотек-плагинов;
- динамическую библиотеку, реализующую заданный вариантом лабораторной работы из Табл. 4 критерий поиска файлов.

18	<i>Опция:</i> --mac-addr-bin <значение> <i>Назначение:</i> поиск файлов, содержащих заданное значение MAC-адреса в бинарной (little-endian или big-endian) форме. <i>Пример:</i> --mac-addr-bin a0:12:ce:78:9f:b4
----	---

Makefile:

all:

```
gcc -fPIC -shared lib.c -o lab1iasN3249.so -Wall -Werror -Wextra -O2
```

```
LD_LIBRARY_PATH=. gcc -o lab1iasN3249 lab1iasN3249.c -ldl -lm -Wall -Werror -Wextra -O2
```

clean:

```
rm lab1iasN3249.so lab1iasN3249
```

Отчет valgrind:

```
==3898== HEAP SUMMARY:
```

```
==3898==    in use at exit: 0 bytes in 0 blocks
```

```
==3898== total heap usage: 828 allocs, 828 frees, 815,900 bytes allocated
```

```
==3898==
```

```
==3898== All heap blocks were freed -- no leaks are possible
```

Скриншоты работы программы:

```
oem@ildan-vm:~/lab1iasN3249$ ./lab1iasN3249
Use: ./lab1 [[options] catalog]
Avaliable options:
-A - AND flag
-O - OR flag
-N - NOT flag
-h - help information
-v - version of the program
-p [DIR]- directory to find plugins
```

```
oem@ildan-vm:~/lab1iasN3249$ ./lab1iasN3249 -h
Plugin purpose:      Find files which contain given double number in binary form
Plugin author:       Daniil Smirnov
Supported options:
    --double-bin      Target double number

Plugin purpose:      Check if a checksum of the files in a directory calculated with CRC-32 algorithm is equal to a given number
Plugin author:       Polina Potapova
Supported options:
    --crc32           EXPECTED CRC-32

Plugin purpose:      Проверяет файл на содержание заданного значение MAC-адреса в бинарной (little-endian или big-endian) форме.
Plugin author:       Шарифуллин Ильдан Айдарович
Supported options:
    --mac-addr-bin    Принимает 1 аргумент - мак адрес в бинарной форме. Далее поиск будет осуществляться в том числе по заданному мак адресу.

-A - AND flag
-O - OR flag
-N - NOT flag
-h - help information
-v - version of the program
-p [DIR]- directory to find plugins
```

```
oem@ildan-vm:~/lab1iasN3249$ ./lab1iasN3249 --mac-addr-bin 23:23:23:23:23:23 .
Файл удовлетворяет условиям: ./test/111
Файл удовлетворяет условиям: ./test/123
Файл удовлетворяет условиям: ./lab1iasN3249.c
oem@ildan-vm:~/lab1iasN3249$ LAB1DEBUG=1 ./lab1iasN3249 --mac-addr-bin 23:23:23:23:23:23 .

    проверяем file 'libdasN3247.so' по opt mac-addr-bin
    проверяем file 'valgrind.txt' по opt mac-addr-bin
    проверяем file '111' по opt mac-addr-bin
    В файле ./test/111 найдена строка 23:23:23:23:23:23
Файл удовлетворяет условиям: ./test/111
    проверяем file '123' по opt mac-addr-bin
    В файле ./test/123 найдена строка 23:23:23:23:23:23
Файл удовлетворяет условиям: ./test/123
    проверяем file 'libpapN3251.so' по opt mac-addr-bin
    проверяем file 'lab1iasN3249.so' по opt mac-addr-bin
    проверяем file 'Makefile' по opt mac-addr-bin
    проверяем file 'lib.c' по opt mac-addr-bin
    проверяем file 'plugin_api.h' по opt mac-addr-bin
    проверяем file 'lab1iasN3249' по opt mac-addr-bin
    проверяем file 'lab1iasN3249.c' по opt mac-addr-bin
    В файле ./lab1iasN3249.c найдена строка 23:23:23:23:23:23
Файл удовлетворяет условиям: ./lab1iasN3249.c
```

```
oem@ildan-vm:~/lab1iasN3249$ ./lab1iasN3249 --mac-addr-bin 23:23:23:23:23:23 --m
ac-addr-bin 12:12:12:12:12:12 .
Файл удовлетворяет условиям: ./test/111
```

Исходный код программ:

lab1iasN3249.c

```
#include <stdlib.h>
#include <stdbool.h>
#include <stdio.h>
#include <unistd.h>
#include <dirent.h>
#include <errno.h>
#include "plugin_api.h"
#include <getopt.h>
#include <dlfcn.h>

int memcmp2(const void* ,
            const void* ,
            size_t );

void dirout(DIR*, char*, struct option*, int);
int (*proc_file)(const char*, struct option*, size_t);
int (*get_info)(struct plugin_info*);
char *var = NULL;
char *plugin_path = NULL;
int flag_OR = 0, flag_AND = 0, flag_NOT = 0, flag_wow = 0, res = 0;

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

    int option_index = 0, c = 0, help_flag = 0, p = 0, rez = 0, count = 0;
    char argv_copy = (char ) malloc(argc*sizeof(char *));
    DIR* dir = NULL;
    char* path = "";
```

```

struct option *opts = (struct option *) malloc(sizeof(struct option));
void *handle = NULL;
struct dirent *entry;
char *var2;
char *var3;
while (c != argc) {
    int j = 0;
    argv_copy[c] = (char *) malloc((j+1)*sizeof(char));
    while ((argv[c][j] != ' ') && (argv[c][j] != '\0')) {
        argv_copy[c][j] = argv[c][j];
        j++;
        argv_copy[c] = (char *) realloc(argv_copy[c], (j+1)*sizeof(char));
    }
    argv_copy[c][j] = '\0';
    c++;
}
c = 0;

struct plugin_info ppi = {0};

if (argc == 1) {
    printf("Use: ./lab1 [[options] catalog\n");
    printf("Avaliable options:\n");
    printf("-A - AND flag\n");
    printf("-O - OR flag\n");
    printf("-N - NOT flag\n");
    printf("-h - help information\n");
    printf("-v - version of the program\n");
    printf("-p [DIR]- directory to find plugins\n");
    goto metka;
}

```

```

const struct option long_options[] = {
    {NULL,0,NULL,0}
};

opterr = 0;

while ((c = getopt_long(argc, argv, "m:", long_options, &option_index)) != -1) {
    int curind = optind;
    if (!optopt) {
        c = 'm';
    }
    switch (c){
        case 'm':
            p = 2;
            while (argv[curind - 1][p]) {
                argv[curind - 1][p-2] = argv[curind - 1][p];
                p++;
            }
            argv[curind - 1][p-2] = argv[curind - 1][p];
            opts[count].name = argv[curind - 1];
            opts[count].has_arg = 1;
            opts[count].flag = (int*) argv[curind];
            count++;
            opts = (struct option *) realloc(opts, sizeof(struct option)*(count + 1));
            break;
    };
};

optind = 0;

while ( ( rez = getopt(argc, argv_copy, "AONvhp:")) != -1){
    switch (rez) {
        case 'A':

```

```

    opts[count].name = "A";
    count++;
    opts = (struct option *) realloc(opts, sizeof(struct option)*(count + 1));
    break;
case 'O':
    opts[count].name = "O";
    count++;
    opts = (struct option *) realloc(opts, sizeof(struct option)*(count + 1));
    break;
case 'N':
    opts[count].name = "N";
    count++;
    opts = (struct option *) realloc(opts, sizeof(struct option)*(count + 1));
    break;
case 'v': printf("version: 1.0. Sharifullin Ildan Aydarovich N3249\n"); goto metka; break;
case 'h': help_flag = 1; break;
case 'p':
    plugin_path = optarg;
    break;
} // switch
} // while

if (opts) {
    opts = (struct option *) realloc(opts, sizeof(struct option)*(count));
}

if (!plugin_path) {
    free(plugin_path);
    plugin_path = ".";
}

if ((dir = opendir(plugin_path)) == NULL) {
    printf("Вы ввели неверный каталог для поиска плагина: %s\n", plugin_path);
}

```

```
goto metka;  
}
```

```
if (errno == ENOTDIR){  
    perror("opendir");  
    printf("opendir error\n");  
    goto metka;  
}
```

```
while ((entry = readdir(dir)) != NULL){  
    var = (char*) malloc(90);  
    var2 = entry -> d_name;  
    var3 = plugin_path;  
    int co = 0;  
    while (*var3 != '\0') {  
        var[co] = *var3;  
        co++;  
        var3++;  
    }  
    var[co] = '/';  
    co++;  
    while (*var2 != '\0') {  
        var[co] = *var2;  
        co++;  
        var2++;  
    }  
}
```

```
if (((handle = dlopen(var, RTLD_LAZY)) != NULL) && (var[co - 1] == 'o') && (var[co - 2] ==  
's') && (var[co - 3] == '.')) { //23:23:23:23:23:23
```

```
if ((proc_file = dlsym(handle, "plugin_process_file")) == NULL) {  
    printf("error find function plugin_process_file in so\n");  
    goto metka;  
}
```



```

if ((get_info = dlsym(handle, "plugin_get_info")) == NULL) {
    printf("error find function plugin_get_info in so\n");
    goto metka;
}

if (help_flag && !(*get_info)(&ppi)) {
    printf("Plugin purpose:\t\t%s\n", ppi.plugin_purpose);
    printf("Plugin author:\t\t%s\n", ppi.plugin_author);
    printf("Supported options: ");
    printf("\n");
    for (size_t j = 0; j < ppi.sup_opts_len; j++) {
        printf("\t--%s\t\t%s\n", ppi.sup_opts[j].opt.name, ppi.sup_opts[j].opt_descr);
    }
    printf("\n");
}

if (handle) dlclose(handle);
}

if (var)
    free(var);
}

if (help_flag) {
    printf("-A - AND flag\n");
    printf("-O - OR flag\n");
    printf("-N - NOT flag\n");
    printf("-h - help information\n");
    printf("-v - version of the program\n");
    printf("-p [DIR]- directory to find plugins\n");
    goto metka;
}

for (size_t i = 0; i < (size_t) count; i++){
//    printf("%s\n", opts[i].name);

```

```

    if (opts[i].name != NULL) {
        if (*(opts[i].name) == 'O')
            flag_OR = 1;
        else if (*(opts[i].name) == 'A') {
            flag_AND = 1;
            res = 1;
        }
        else if (*(opts[i].name) == 'N')
            flag_NOT = 1;
    }
}

if ((!flag_OR) && (!flag_AND)) {
    flag_AND = 1;
}

path = argv[argc - 1];

if (dir) closedir(dir);

if ((dir = opendir(path)) == NULL) { //открытие каталога
    printf("Вы ввели неверный или вовсе не ввели каталог\n");
    goto metka;
}

if (errno == ENOTDIR){
    perror("opendir");
    printf("opendir error\n");
    goto metka;
}

printf("\n");
dirout(dir, path, opts, count);
metka:
if (dir) closedir(dir);

```

```

free(opts);
for (int i = 0; i < argc; i++)
    free(argv_copy[i]);
free(argv_copy);
return 0;
}

void dirout(DIR *cdir, char *path , struct option *opts, int count) {
    DIR *hdir, *jdir;
    char *url;
    char *var2;
    char *var3;
    char *var4;
    char *var5;
    char *name2 = "";
    void *handle = NULL;
    struct dirent *entry, *entry2;
    struct plugin_info ppi = { };
    int buf = 0, j = 0;
    struct option for_plugin[1] = { {"asd",0,0,0} };

    while ((entry = readdir(cdir)) != NULL){
        url = (char*) malloc(100);
        var2 = entry -> d_name;
        var3 = path;
        int co = 0;
        while (*var3 != '\0') {
            url[co] = *var3;
            co++;
            var3++;
        }
        url[co] = '/';
    }

```

```

    co++;
while (*var2 != '\0') {
    url[co] = *var2;
    co++;
    var2++;
}
url[co] = '\0';
entry2 = NULL;
jdir = opendir(plugin_path);
if ((entry -> d_type) == DT_REG) {
    while ((entry2 = readdir(jdir)) != NULL){
        var = (char*) malloc(90);
        var4 = entry2 -> d_name;
        var5 = plugin_path;
        int cou = 0;
        while (*var5 != '\0') {
            var[cou] = *var5;
            cou++;
            var5++;
        }
        var[cou] = '/';
        cou++;
        while (*var4 != '\0') {
            var[cou] = *var4;
            cou++;
            var4++;
        }
        var[cou] = '\0';
//    printf("\nres %s \n", var);
        if (((handle = dlopen(var, RTLD_LAZY)) != NULL) && (var[cou - 1] == 'o') && (var[cou
- 2] == 's') && (var[cou - 3] == '.')) { //23:23:23:23:23:23
            get_info = dlsym(handle, "plugin_get_info");
            int p = 0;
            if (!(*get_info)(&ppi)) {

```

```

name2 = (char*) ppi.sup_opts[0].opt.name;
while (name2[p])
    p++;
}
for (size_t i = 0; i < (size_t) (count); i++){
j = 0;
    if (opts[i].name != NULL) {
        while (opts[i].name[j])
            j++;
        if (!(j - p))
            if (!memcmp2(name2, opts[i].name, p)) {
                proc_file = dlsym(handle, "plugin_process_file");
                if (getenv("LAB1DEBUG")){
                    printf("\tproveryaem file '%s' po opt %s\n", entry -> d_name, name2);
                }
                for_plugin[0] = opts[i];
                buf = (*proc_file)(url, for_plugin, 1);
                if (!buf) {
                    if (flag_AND && !flag_wow) {
                        res = 1;
                        flag_wow = 1;
                    }
                    if (flag_OR)
                        res |= 1;
                    else if (flag_AND) {
                        res &= 1;
                        flag_wow = 1;
                    }
                } else {
                    if (flag_OR)
                        res |= 0;
                    else if (flag_AND) {
                        res &= 0;

```

```

        flag_wow = 1;
    }
}
buf = 0;
}
}
}
if (handle) dlclose(handle);
}
free(var);
}
if (flag_NOT) {
    if (res == 0) {
        printf("Файл удовлетворяет условиям: %s\n", url);    //2.1
    }
}
else {
    if (res != 0)
        printf("Файл удовлетворяет условиям: %s\n", url);
}
flag_wow = 0;
res = 0;
}
else if ((*entry -> d_name) != '.') {
    if ((hdir = opendir(url)) != NULL) {
//        printf("открываем каталог: %s\n\n", url);
        dirout(hdir, url, opts, count);
        closedir(hdir);
    }
}
if (url) free(url);
if (jdir)
    closedir(jdir);

```

```

    }
}

int memcmp2(const void* buf1,
            const void* buf2,
            size_t count)
{
    if(!count)
        return(0);

    while(--count && (*(char*)buf1 == *(char*)buf2) ) {
        buf1 = (char*)buf1 + 1;
        buf2 = (char*)buf2 + 1;
    }

    return(*((unsigned char*)buf1) - *((unsigned char*)buf2));
}

```

lab.c

```
#include <getopt.h>
```

```
#include <stdio.h>
```

```
#include "plugin_api.h"
```

```
#include <stdlib.h>
```

```
static struct plugin_option asd[] = { { {"mac-addr-bin", required_argument, 0, 0}, "Принимает 1 аргумент - мак адрес в бинарной форме. Далее поиск будет осуществляться в том числе по заданному мак адресу." } };
```

```
int plugin_get_info(struct plugin_info* ppi) {
```

```
    ppi->plugin_purpose = "Проверяет файл на содержание заданного значение MAC-адреса в бинарной (little-endian или big-endian) форме.";
```

```
    ppi->plugin_author = "Шарифуллин Ильдан Айдарович";
```

```
    ppi->sup_opts_len = 1;
```

```
    ppi->sup_opts = asd;
```

```

    return 0;
}

int memcmp2(const void* buf1,
            const void* buf2,
            size_t count)
{
    if(!count)
        return(0);

    while(--count && (*(char*)buf1 == *(char*)buf2) ) {
        buf1 = (char*)buf1 + 1;
        buf2 = (char*)buf2 + 1;
    }

    return(*((unsigned char*)buf1) - *((unsigned char*)buf2));
}

int BF(void *x, char *y, long unsigned int m) {
    int i;
    // printf("%s and %s\n", (char *)x, y);
    for(i=0; (*y != EOF) && (i <= 999); i++, y++) {
        if(memcmp2(y,x,m) == 0) {
            return 1;
        }
    }
    return 0;
}

int plugin_process_file(const char *fname,
                        struct option in_opts[],

```



```

size_t in_opts_len) {

FILE* fl;

char* strm = (char*) malloc(128*sizeof(char));
int* reversed = (int*) malloc(18*sizeof(int));
int* addrs_opts = (int*) malloc(in_opts_len*sizeof(int));
int flg = 0, res = 0;
int flag_OR = 0, flag_AND = 0, flag_wow = 0;

for (int i = 0; i < (int)in_opts_len; i++){
    addrs_opts[i] = 0;
}
if ((fl = fopen(fname, "r")) == NULL)
{
//    perror("openfile");
if (getenv("LAB1DEBUG"))
    printf("Не удалось открыть файл: %s\n", fname);
    return -1;
}
for (size_t i = 0; i < in_opts_len; i++){
//    printf("%s\n", in_opts[i].name);
if (in_opts[i].name != NULL) {
    if (*(in_opts[i].name) == 'O')
        flag_OR = 1;
    else if (*(in_opts[i].name) == 'A') {
        flag_AND = 1;
        res = 1;
    }
}
}
if ((!flag_OR) && (!flag_AND)) {
    flag_AND = 1;
}
}

```

```

while (fgets(strm, 128, fl)) {
    for (size_t i = 0; i < in_opts_len; i++){
        flg = 0;
        if (in_opts[i].name != NULL) {
            if (BF("mac-addr-bin", (char*)in_opts[i].name, 13)) {
                flg = BF(in_opts[i].flag, strm, 17);
            }
            /*
                if (flag_OR)
                    res |= flg;
                else if (flag_AND)
                    res &= flg;
                if (flg)
                    printf("!!! %s and %s\n", (char *)in_opts[i].flag, strm); */
                addrs_opts[i] += flg;
            }
        }
    }

    for (size_t i = 0; i < in_opts_len; i++){
        if (in_opts[i].name != NULL) {
            if (BF("mac-addr-bin", (char*)in_opts[i].name, 13)) {
                if (addrs_opts[i]) {
                    if (getenv("LAB1DEBUG")){
                        printf("\tB файле %s найдена строка %s \n", fname, (char*) in_opts[i].flag);
                    }

                    if (flag_AND && !flag_wow) {
                        res = 1;
                        flag_wow = 1;
                    }

                    if (flag_OR)
                        res |= 1;
                }
            }
        }
    }
}

```

```
        else if (flag_AND) {
            res &= 1;
            flag_wow = 1;
        }
    } else {
        if (flag_OR)
            res |= 0;
        else if (flag_AND) {
            res &= 0;
            flag_wow = 1;
        }
    }
}
}
```

```
fclose(fl);
free(strm);
free(reversed);
free(addr_opts);
```

```
if (res == 0)
    return 1;
else
    return 0;
}
```