

Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИТМО»

Факультет безопасности информационных технологий
Направление подготовки 10.03.01 «Информационная безопасность»

Отчет по лабораторной работе № 1.2
по дисциплине «Основы системного
программирования»
по теме: «Использование динамических
библиотек»

Студент:

Самайкин Е.С.

Подпись:



Группа:

N32491

Оценка (баллы):

Преподаватели:

Грозов В. А., Гирик. А. В.

Санкт-Петербург 2023

СОДЕРЖАНИЕ

1 Задание	3
2 Make-file	5
3 Отчет valgrind	6
4 Работа программы	7
5 Тексты программ	9

1 Задание

Разработать на языке C для ОС Linux:

1. 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);
2. 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).

При запуске без имени каталога для поиска программа выводит справочную информацию по опциям и доступным в момент запуска плагинам. Поддерживаемые программой опции перечислены в Табл. 2. Выводятся все опции, поддерживаемые плагинами, и их описание. По умолчанию плагины ищутся в том же каталоге, где расположен исполняемый файл программы, а если задана опция -P, то в каталоге, указанном в этой опции. В случае запуска программы с несколькими опциями, задающими критерии поиска, эти критерии объединяются логической операцией «И» (то же самое, если задана опция -A) или логической операцией «ИЛИ» (если задана опция -O). Если задана опция -N, то после объединения всех условий поиска по «И» или «ИЛИ», оно меняется на противоположное.

Плагины представляют собой динамические библиотеки в формате ELF с произвольным именем и расширением .so и интерфейсными функциями, перечисленными в Табл. 1. Подробное описание API плагинов содержится в файле plugin_api.h.

При обнаружении файла, отвечающего заданным критериям поиска, в стандартный поток вывода выводится полный путь к этому файлу. При определении переменной окружения LAB1DEBUG в стандартный поток ошибок должна выводиться информация о том, что и в каком месте файла нашлось (чтобы было легче понять, почему файл отвечает

критериям поиска), а также может выводиться любая дополнительная отладочная информация. Переменные окружения, которые должны поддерживаться программой и библиотекой, приведены в Табл. 3.

Обход осуществляется в соответствии с вариантом 3 – `ftw()`.

Выполняется поиск заданной последовательности байтов. Аргумент `цель_поиска` имеет формат строки в кодировке UTF-8.

2 Make-file

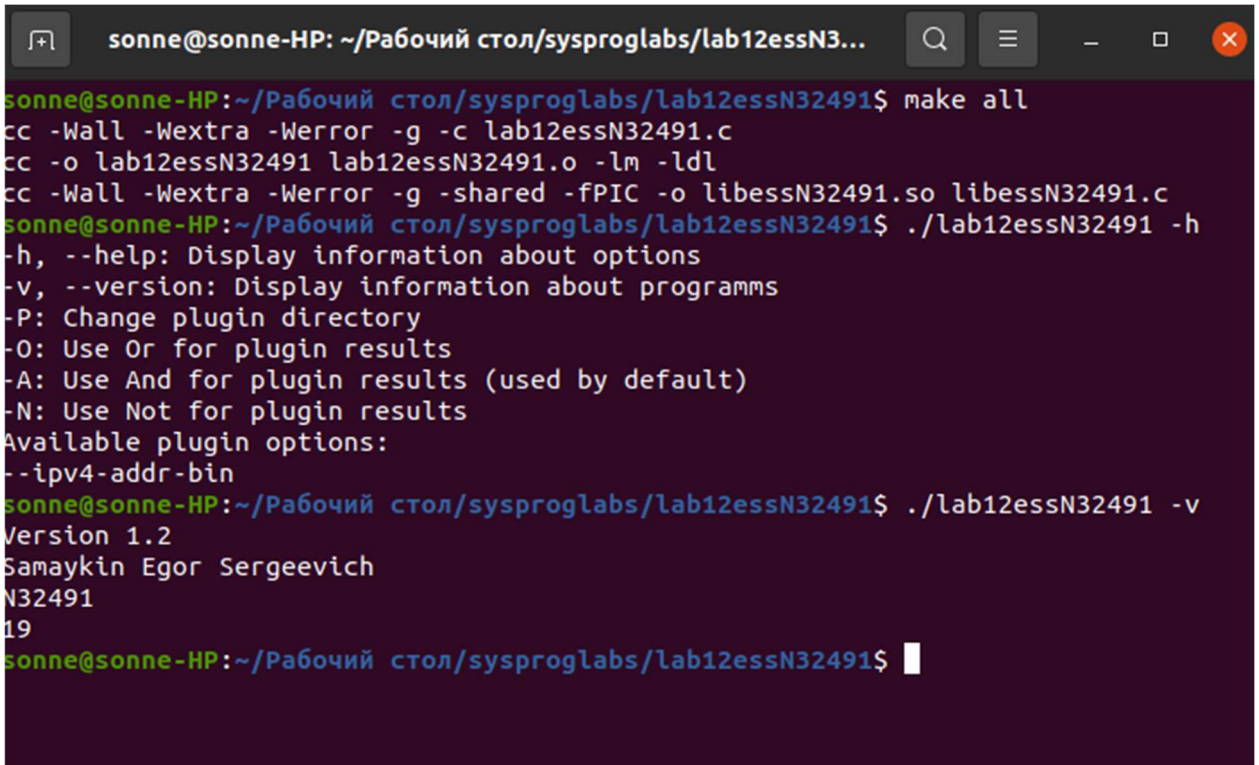
```
APP=lab12essN32491
SRCS=lab12essN32491.c
LIB=libessN32491.so
OBS=lab12essN32491.o
CC=gcc
CFLAGS=-Wall -Wextra -Werror -g
LDFLAGS=-lm -ldl
LIBFLAGS=-shared -fPIC

.PHONY: all clean
all: $(APP) $(LIB)
clean:
    rm -rf *.o $(APP) $(LIB)
$(APP): $(OBS)
    $(CC) -o $@ $^ $(LDFLAGS)
$(OBS): $(SRCS)
    $(CC) $(CFLAGS) -c $^
$(LIB): libessN32491.c
    $(CC) $(CFLAGS) $(LIBFLAGS) -o $@ $^
```

3 Отчет valgrind

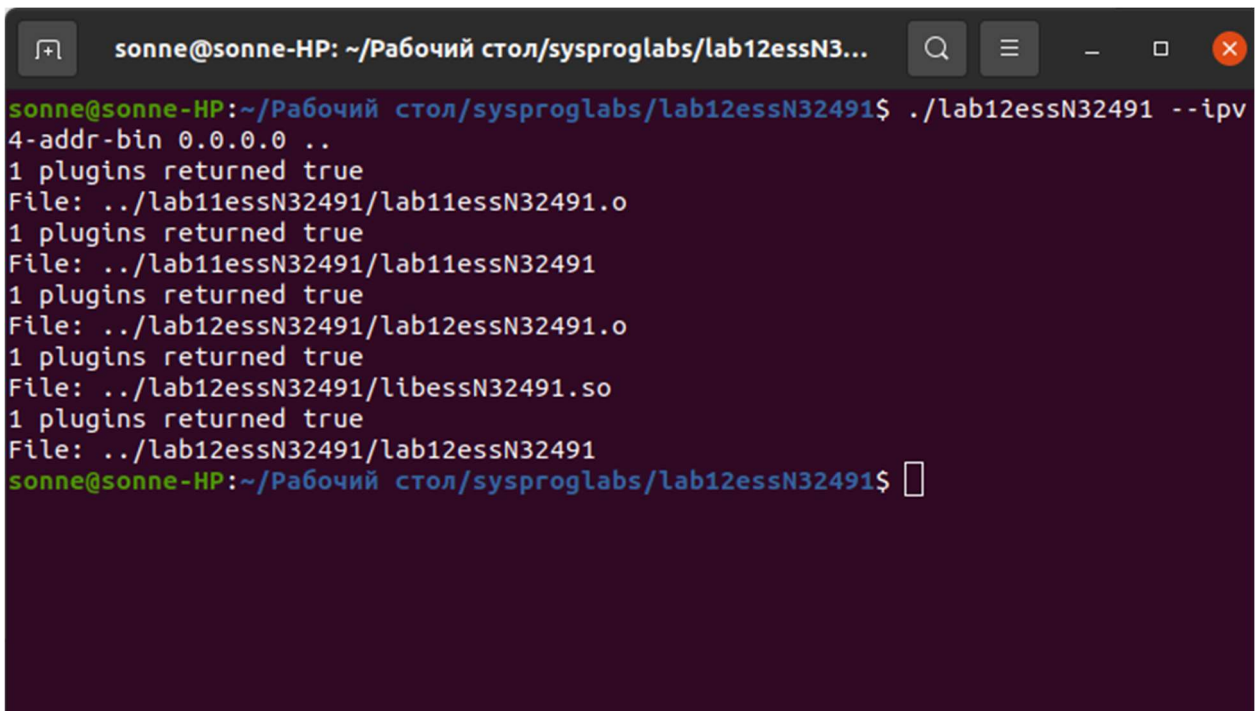
```
==152845== HEAP SUMMARY:
==152845==      in use at exit: 0 bytes in 0 blocks
==152845==    total heap usage: 919 allocs, 919 frees, 2,339,694 bytes
allocated
==152845==
==152845== All heap blocks were freed -- no leaks are possible
==152845==
==152845== For lists of detected and suppressed errors, rerun with: -s
==152845== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from
0)
```

4 Работа программ



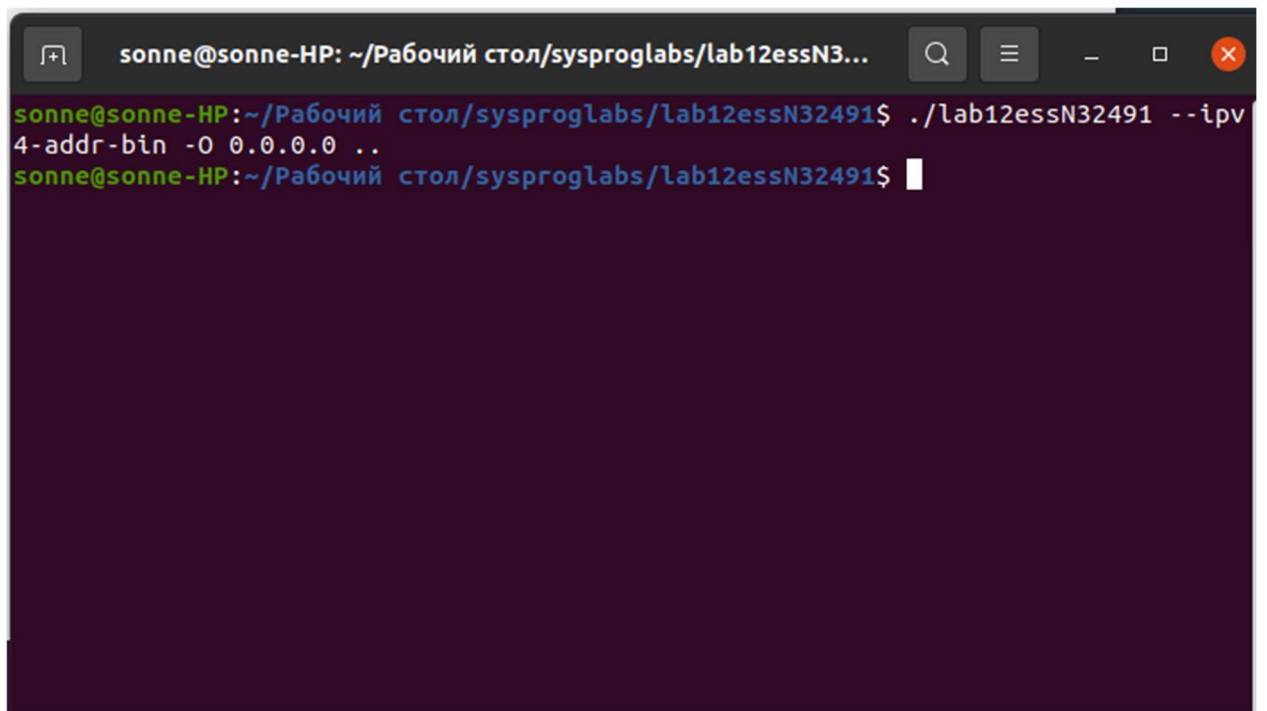
```
sonne@sonne-HP: ~/Рабочий стол/sysproglabs/lab12essN3...
sonne@sonne-HP:~/Рабочий стол/sysproglabs/lab12essN32491$ make all
cc -Wall -Wextra -Werror -g -c lab12essN32491.c
cc -o lab12essN32491 lab12essN32491.o -lm -ldl
cc -Wall -Wextra -Werror -g -shared -fPIC -o libessN32491.so libessN32491.c
sonne@sonne-HP:~/Рабочий стол/sysproglabs/lab12essN32491$ ./lab12essN32491 -h
-h, --help: Display information about options
-v, --version: Display information about programmes
-P: Change plugin directory
-O: Use Or for plugin results
-A: Use And for plugin results (used by default)
-N: Use Not for plugin results
Available plugin options:
--ipv4-addr-bin
sonne@sonne-HP:~/Рабочий стол/sysproglabs/lab12essN32491$ ./lab12essN32491 -v
Version 1.2
Samaykin Egor Sergeevich
N32491
19
sonne@sonne-HP:~/Рабочий стол/sysproglabs/lab12essN32491$
```

Рисунок 1 – пример работы программы



```
sonne@sonne-HP: ~/Рабочий стол/sysproglabs/lab12essN3...
sonne@sonne-HP:~/Рабочий стол/sysproglabs/lab12essN32491$ ./lab12essN32491 --ipv4-addr-bin 0.0.0.0 ..
1 plugins returned true
File: ../lab11essN32491/lab11essN32491.o
1 plugins returned true
File: ../lab11essN32491/lab11essN32491
1 plugins returned true
File: ../lab12essN32491/lab12essN32491.o
1 plugins returned true
File: ../lab12essN32491/libessN32491.so
1 plugins returned true
File: ../lab12essN32491/lab12essN32491
sonne@sonne-HP:~/Рабочий стол/sysproglabs/lab12essN32491$
```

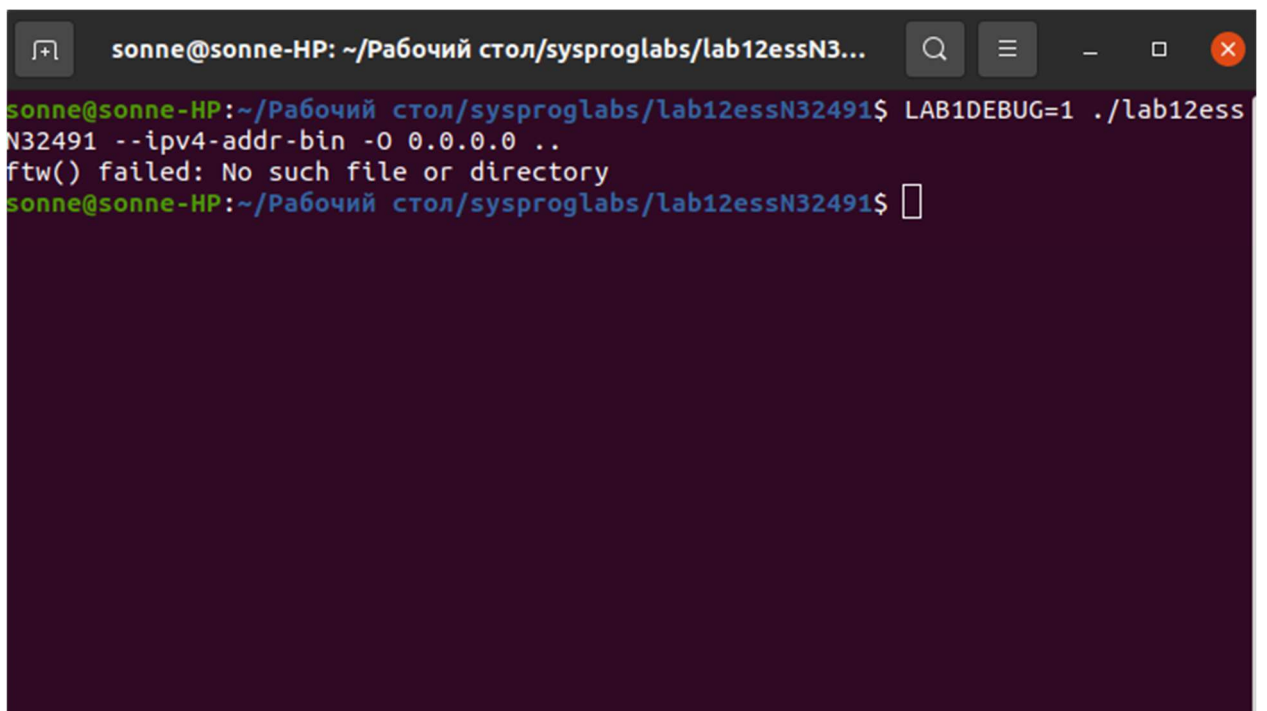
Рисунок 2 – пример работы программы



A terminal window with a dark purple background. The title bar shows the user 'sonne@sonne-HP' and the current directory '~/Рабочий стол/sysproglabs/lab12essN3...'. The prompt is 'sonne@sonne-HP:~/Рабочий стол/sysproglabs/lab12essN32491\$'. The command './lab12essN32491 --ipv4-addr-bin -O 0.0.0.0 ..' has been entered and executed. The prompt is now 'sonne@sonne-HP:~/Рабочий стол/sysproglabs/lab12essN32491\$' with a cursor at the end.

```
sonne@sonne-HP: ~/Рабочий стол/sysproglabs/lab12essN3...
sonne@sonne-HP:~/Рабочий стол/sysproglabs/lab12essN32491$ ./lab12essN32491 --ipv4-addr-bin -O 0.0.0.0 ..
sonne@sonne-HP:~/Рабочий стол/sysproglabs/lab12essN32491$
```

Рисунок 3 – пример работы программы



A terminal window with a dark purple background. The title bar shows the user 'sonne@sonne-HP' and the current directory '~/Рабочий стол/sysproglabs/lab12essN3...'. The prompt is 'sonne@sonne-HP:~/Рабочий стол/sysproglabs/lab12essN32491\$'. The command 'LAB1DEBUG=1 ./lab12essN32491 --ipv4-addr-bin -O 0.0.0.0 ..' has been entered and executed. The output is 'ftw() failed: No such file or directory'. The prompt is now 'sonne@sonne-HP:~/Рабочий стол/sysproglabs/lab12essN32491\$' with a cursor at the end.

```
sonne@sonne-HP:~/Рабочий стол/sysproglabs/lab12essN32491$ LAB1DEBUG=1 ./lab12essN32491 --ipv4-addr-bin -O 0.0.0.0 ..
ftw() failed: No such file or directory
sonne@sonne-HP:~/Рабочий стол/sysproglabs/lab12essN32491$
```

Рисунок 4 – пример работы программы

5 Тексты программ

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <dirent.h>
#include <string.h>
#include <getopt.h>
#include <dlfcn.h> //Хед для dlopen()
#include <errno.h>
#include <ftw.h> //Хед для ftw обхода
#include <sys/param.h>
#include <malloc.h>

#include "plugin_api.h"

//типы-указатели на функции plugin_api
typedef int (*ppf_func_t)(const char *, struct option *, size_t);
typedef int (*pgi_func_t)(struct plugin_info *);

//Структура, в которой будет храниться все для запуска плагинов
struct plugin
{
    ppf_func_t process_file;
    struct option *in_opts;
    size_t in_opts_len;
};
void **plugin_handles = NULL;
struct plugin *plugin_array = NULL;
struct plugin_info *info_array = NULL;

//гп для хранения директории поиска
char *dir_search;

//гп для хранения пути поиска плагинов
char *plug_path = "./";

//гп для сохранения количества плагинов
int plugin_cnt = 0;

//Флаги для отладки, проверки и запуска плагинов соответственно
int Search_done_FLAG = 0;
int Debug_FLAG = 0;
int Or_FLAG = 0;
int Not_FLAG = 0;

//Запуск plugin_process_file
void CheckFilePlugins(const char *fpath)
{

```

```

        //Сколько плагинов сказали что файл подходит, и сколько плагинов успешно
        запустились
        int result = 0;
        int used = 0;
        for (int i = 0; i < plugin_cnt; i++)
        {
            if (plugin_array[i].in_opts_len > 0)
            {
                used++;
                int t = plugin_array[i].process_file(fpath,
                plugin_array[i].in_opts, plugin_array[i].in_opts_len);
                if (t < 0)
                {
                    used--;
                    fprintf(stderr, "Error in plugin №%d! %s", i,
                    strerror(errno));
                    if (errno == EINVAL)
                        fprintf(stderr, "errno is EINVAL, skipping this plugin
                        for next files\n");
                }
                else if (t == 0)
                    result++;
            }
        }
        if ( //Если файл подходит с учетом флагов Or_FLAG и Not_FLAG выводим
        (Not_FLAG == 0 && ((Or_FLAG == 1 && result > 0) || (Or_FLAG == 0 &&
        result == used))) ||
        (Not_FLAG == 1 && ((Or_FLAG == 1 && result == 0) || (Or_FLAG == 0 &&
        result < used))))
        {
            fprintf(stdout, "%d plugins returned true\n", result);
            fprintf(stdout, "File: %s\n", fpath);
        }
    }

    /*Принимает инфу о текущем пути файла fpath, структуру stat и флаг typeflag,
    указывающий тип файла. Если файл является обычным файлом FTW_F, то функция
    вызывает CheckFileLine() для анализа содержимого файла */
    int CheckFile(const char *fpath, const struct stat *sb, int typeflag)
    {
        (void)sb;
        if (typeflag == FTW_F)
            CheckFilePlugins(fpath);
        return 0;
    }

    /*Принимает инфу о текущем пути файла, структуру stat и флаг typeflag. Если
    файл - обычный FTW_F, и имеет в названии ".so", функция открывает эту
    библиотеку и проверяет на соответствие plugin_api*/
    int CheckPlugin(const char *fpath, const struct stat *sb, int typeflag)
    {
        (void)sb;
        if (typeflag == FTW_F && strstr(fpath, ".so") != NULL)
        {
            void *Lib_handle = dlopen(fpath, RTLD_LAZY);
            if (!Lib_handle)
            {
                if (Debug_FLAG)
                    fprintf(stderr, "%s: dlopen() failed!: %s", fpath,
                    dlerror());
            }
        }
    }

```

```

        return 0;
    }
    else
    {
        pgi_func_t pgi = dlsym(Lib_handle, "plugin_get_info");
        ppf_func_t ppf = dlsym(Lib_handle, "plugin_process_file");
        if (!pgi || !ppf)
        {
            fprintf(stderr, "%s: dlsym for plugin_get_info or
plugin_process_file failed: %s\n", fpath, dlerror());
            dlclose(Lib_handle);
            return 0;
        }
        struct plugin_info pi = {0};
        int res = pgi(&pi);
        if (res < 0)
        {
            fprintf(stderr, "%s: plugin_get_info returned %d, skipping
it\n", fpath, res);
            dlclose(Lib_handle);
            return 0;
        }
        if (pi.sup_opts_len == 0)
        {
            fprintf(stderr, "%s: plugin supports no options, skipping
it\n", fpath);
            dlclose(Lib_handle);
            return 0;
        }
        plugin_cnt++;
        info_array = realloc(info_array, sizeof(struct plugin_info) *
plugin_cnt);
        info_array[plugin_cnt - 1] = pi;
        plugin_array = realloc(plugin_array, sizeof(struct plugin) *
plugin_cnt);
        plugin_array[plugin_cnt - 1].process_file = ppf;
        plugin_array[plugin_cnt - 1].in_opts_len = 0;
        plugin_array[plugin_cnt - 1].in_opts = NULL;
        plugin_handles = realloc(plugin_handles, sizeof(void *) *
plugin_cnt);
        plugin_handles[plugin_cnt - 1] = Lib_handle;
    }
}
return 0;
}

/*Начинает обход ФС, если еще не проверяли плагины, запускаем с функцией
CheckPlugin в директории plug_path, иначе проходимся по dir_search функцией
CheckFile*/
void WalkDirectory()
{
    if (Search_done_FLAG == 0)
    {
        Search_done_FLAG = 1;
        int result = ftw(plug_path, CheckPlugin, FTW_NS);
        if (result < 0 && Debug_FLAG)
        {
            fprintf(stderr, "ftw() failed: %s\n", strerror(errno));
        }
        return;
    }
    else
    {

```

```

    int result = ftw(dir_search, CheckFile, FTW_NS);
    if (result < 0 && Debug_FLAG)
    {
        fprintf(stderr, "ftw() failed: %s\n", strerror(errno));
    }
}

/*Функция для разбора аргументов ком строки, getopt нужен для обработки
опций*/
void OptionDo(int argc, char *const *argv)
{
    int value = 0;
    opterr = 0;
    //Считаем сколько опций надо выделить (+2 для version и help)
    size_t opt_cnt = 2;
    for (int i = 0; i < plugin_cnt; i++)
        opt_cnt += info_array[i].sup_opts_len;
    struct option *long_options = calloc(opt_cnt + 1, sizeof(struct option));
    long_options[0] = (struct option){ "help", 0, 0, 'h' };
    long_options[1] = (struct option){ "version", 0, 0, 'v' };
    size_t opt_copy_idx = 2;
    //Копируем опции в long_options
    for (int i = 0; i < plugin_cnt; i++)
    {
        for (size_t j = 0; j < info_array[i].sup_opts_len; j++)
        {
            long_options[opt_copy_idx] = info_array[i].sup_opts[j].opt;
            opt_copy_idx++;
        }
    }

    int found_plug_opts = 0;
    while (1)
    {
        int found_cnt = 0;
        int optindex = 0;
        value = getopt_long(argc, argv, "hvP:OAN", long_options, &optindex);
        if (value == -1)
            break;
        //Выводимая инфа
        switch (value)
        {
        case 'h':
            fprintf(stdout, "-h, --help: Display information about options\
\n-v, --version: Display information about programmes\
\n-P: Change plugin directory\
\n-O: Use Or for plugin results\
\n-A: Use And for plugin results (used by default)\
\n-N: Use Not for plugin results\n");
            fprintf(stdout, "Available plugin options:\n");
            for (size_t i = 2; i < opt_cnt; i++)
            {
                fprintf(stdout, "--%s\n", long_options[i].name);
            }

            //Освобождаем память перед выходом
            for (int i = 0; i < plugin_cnt; i++)
            {
                if (dlclose(plugin_handles[i]) != 0)
                    fprintf(stderr, "dlclose error: %s\n", dlerror());
                if (plugin_array[i].in_opts)
                    free(plugin_array[i].in_opts);
            }
        }
    }
}

```

```

    }
    if (plugin_handles)
        free(plugin_handles);
    if (plugin_array)
        free(plugin_array);
    if (info_array)
        free(info_array);
    exit(EXIT_SUCCESS);

case 'v':
    fprintf(stdout, "Version 1.2\
        \nSamaykin Egor Sergeevich\
        \nN32491\
        \n19\n");
    //Освобождаем память перед выходом
    for (int i = 0; i < plugin_cnt; i++)
    {
        if (dlclose(plugin_handles[i]) != 0)
            fprintf(stderr, "dlclose error: %s\n", dlerror());
        if (plugin_array[i].in_opts)
            free(plugin_array[i].in_opts);
    }
    if (plugin_handles)
        free(plugin_handles);
    if (plugin_array)
        free(plugin_array);
    if (info_array)
        free(info_array);
    exit(EXIT_SUCCESS);

case 'P':
    //Уже рассмотрел отдельно, чтобы не приходилось освобождать
    память и искать плагины дважды
    break;

case 'O':
    Or_FLAG = 1;
    break;

case 'A':
    Or_FLAG = 0;
    break;

case 'N':
    Not_FLAG = 1;
    break;

case '?':
    if (Debug_FLAG)
        fprintf(stderr, "Inputed unknown option: %c\
            \nUsage: %s [options] directory target\n",
            optopt, argv[0]);
    exit(EXIT_FAILURE);

case 0:
    //Длинная опция для плагина
    for (int i = 0; i < plugin_cnt; i++)
    {
        for (size_t j = 0; j < info_array[i].sup_opts_len; j++)
        {
            if (strcmp(long_options[optindex].name,
info_array[i].sup_opts[j].opt.name) == 0){
                found_plug_opts++;
                found_cnt++;
            }
        }
    }

```

```

        plugin_array[i].in_opts_len++;
        plugin_array[i].in_opts =
realloc(plugin_array[i].in_opts, sizeof(struct option) *
plugin_array[i].in_opts_len);
        plugin_array[i].in_opts[plugin_array[i].in_opts_len-
1] = long_options[optindex];
        if(long_options[optindex].has_arg)
plugin_array[i].in_opts[plugin_array[i].in_opts_len-1].flag = (int*)optarg;
    }
}
    if(found_cnt > 1){
        fprintf(stderr, "Error: 2 options with the same name in
plugins\nOpt name:%s\n",long_options[optindex].name);
        exit(EXIT_FAILURE);
    }
    break;
default:
    break;
}
}
free(long_options);
if (argc - optind < 1 || found_plug_opts == 0)
{
    if (Debug_FLAG)
        fprintf(stderr, "Usage: %s [options] directory\n", argv[0]);
    exit(EXIT_FAILURE);
}
dir_search = argv[optind];
}

int main(int argc, char *const *argv)
{ //(int argc, char **argv)
    //Проверка флага переменной окружения
    if (getenv("LAB1DEBUG"))
    {
        Debug_FLAG = 1;
    }
    //Проверка опции -P перед запуском поиска плагинов
    for (int i = 0; i < argc - 1; i++)
    {
        if (strcmp(argv[i], "-P") == 0)
        {
            if (Debug_FLAG)
                fprintf(stderr, "-P found\n");
            plug_path = argv[i + 1];
            break;
        }
    }
    WalkDirectory();
    OptionDo(argc, argv);

    //Информация была нужна для получения опций, теперь нужно освободить
    if (info_array)
        free(info_array);
    WalkDirectory();

    for (int i = 0; i < plugin_cnt; i++)
    {
        if (dlclose(plugin_handles[i]) != 0)
            fprintf(stderr, "dlclose error: %s\n", dlerror());
        if (plugin_array[i].in_opts)
            free(plugin_array[i].in_opts);
    }
}

```

```

    }
    if (plugin_handles)
        free(plugin_handles);
    if (plugin_array)
        free(plugin_array);
    return 0;
}

```

Листинг 1 – lab12essN32491.c

```

#define _GNU_SOURCE      //Для для strdup
#include <stdio.h>
#include <string.h>
#include <getopt.h>
#include <errno.h>
#include <stdlib.h>

#include "plugin_api.h"

static char *g_lib_name = "libessN32491.so";
static char *g_plugin_purpose = "Search for ipv4 addr in binary form";
static char *g_plugin_author = "Samaykin Egor Sergeevich";
static size_t g_opts_len = 1;
static struct plugin_option g_sup_opts[1] ={
    {
        {"ipv4-addr-bin", required_argument, 0, 0},
        "IP address in XXX.XXX.XXX.XXX format"
    }
};

int plugin_get_info(struct plugin_info *ppi)
{
    //Если ppi - NULL возвращается ошибка
    if (!ppi)
        return -1;

    //Заполняем ppi и возвращаем успех
    ppi->plugin_author = g_plugin_author;
    ppi->plugin_purpose = g_plugin_purpose;
    ppi->sup_opts_len = g_opts_len;
    ppi->sup_opts = g_sup_opts;
    return 0;
}

int plugin_process_file(const char *fname,
                        struct option in_opts[],
                        size_t in_opts_len)
{
    //Если не получили какой то из аргументов выходим с ошибкой
    if (!fname || !in_opts || !in_opts_len)
        return -1;
    int ret = -1;
    char *debug = getenv("LAB1DEBUG");
    /*Получение IP из входных опций*/
    char *ip = NULL;
    for (size_t i = 0; i < in_opts_len; i++)

```

```

{
    if (strcmp(in_opts[i].name, "ipv4-addr-bin") == 0)
    {
        ip = strdup((char *)in_opts[i].flag);
    }
    else
    {
        errno = EINVAL;
        return -1;
    }
}
if (!ip || strlen(ip) > 15 || strlen(ip) < 7 || !strstr(ip, "."))
{
    /*IP адрес это от 7 символов (0.0.0.0), и до 15 (255.255.255.255)
    поэтому если длина не входит в этот промежуток, то входные данные
    неправильные*/
    errno = EINVAL;
    free(ip);
    return -1;
}
//Разбиваем входной IP на 4 байта, и отдельно сохраняем в обратном
порядке для big-endian и little-endian проверки
unsigned char byte_array[4];
unsigned char byte_array_rev[4];
for (int i = 0; i < 4; i++)
{
    char *t;
    if (i == 0)
        t = strtok(ip, ".");
    else
        t = strtok(NULL, ".");
    if (t == NULL)
    {
        //Если не нашелся какой-то токен из четырех, т.е. их слишком
        мало, то неверные входные данные
        if (debug)
            fprintf(stderr, "%s: tokens failed\n", g_lib_name);
        errno = EINVAL;
        free(ip);
        return -1;
    }
    unsigned char value = strtoul(t, NULL, 10);
    if (value == 0 && strcmp(t, "0") != 0)
    {
        //Если ошибка с конвертацией char* в значение, то значит в токене
        не было числа
        if (debug)
            fprintf(stderr, "%s: strtoul() failed\n", g_lib_name);
        errno = EINVAL;
        free(ip);
        return -1;
    }
    byte_array[i] = (char)value;
    byte_array_rev[3 - i] = (char)value;
}
free(ip);
//Проверка файла
FILE *fd = fopen(fname, "rb");
if (!fd)
{
    //Если не получилось открыть файл fopen() сам выставляет errno
    return -1;
}

```



```

while (!feof(fd))
{
    unsigned char buffer[4];
    //Считываем из файла по 4 байта и сверяем
    int t = fread(buffer, sizeof(unsigned char), 4, fd);
    if (t < 4)
    {
        break;
    }
    else if ((memcmp(buffer, byte_array, 4) == 0) || (memcmp(buffer,
byte_array_rev, 4) == 0))
    {
        //Если нашли в прямом или обратном порядке
        ret = 0;
        if (fclose(fd) < 0)
        {
            fprintf(stderr, "Failed to fclose() file!!\n");
        }
        return 0;
    }
}
ret = 1;
if (fclose(fd) < 0)
{
    fprintf(stderr, "Failed to fclose() file!\n");
}
//Освобождаем то, что выделили
return ret;
}

```

Листинг 2 – libessN32491.c

```

#ifndef _PLUGIN_API_H
#define _PLUGIN_API_H

#include <getopt.h>

/*
    Структура, описывающая опцию, поддерживаемую плагином.
*/
struct plugin_option {
    /* Опция в формате, поддерживаемом getopt_long (man 3 getopt_long). */
    struct option opt;
    /* Описание опции, которое предоставляет плагин. */
    const char *opt_descr;
};

/*
    Структура, содержащая информацию о плагине.
*/
struct plugin_info {
    /* Назначение плагина */
    const char *plugin_purpose;
    /* Автор плагина, например "Иванов Иван Иванович, N32xx" */
    const char *plugin_author;
    /* Длина списка опций */
    size_t sup_opts_len;
    /* Список опций, поддерживаемых плагином */
    struct plugin_option *sup_opts;
};

int plugin_get_info(struct plugin_info* ppi);

```

```

/*
plugin_get_info()

Функция, позволяющая получить информацию о плагине.

Аргументы:
    ppi - адрес структуры, которую заполняет информацией плагин.

Возвращаемое значение:
    0 - в случае успеха,
    < 0 - в случае неудачи (в этом случае продолжать работу с этим
платином нельзя).
*/

int plugin_process_file(const char *fname,
    struct option in_opts[],
    size_t in_opts_len);
/*
plugin process file()

Функция, позволяющая выяснить, отвечает ли файл заданным критериям.

Аргументы:
    fname - путь к файлу (полный или относительный), который проверяется
на
        соответствие критериям, заданным с помощью массива in_opts.

    in_opts - список опций (критериев поиска), которые передаются
плагину.
        struct option {
            const char *name;
            int         has_arg;
            int         *flag;
            int         val;
        };
        Поле name используется для передачи имени опции, поле flag - для
передачи
        значения опции (в виде строки). Если у опции есть аргумент, поле
has_arg
        устанавливается в ненулевое значение. Поле val не используется.

    in_opts_len - длина списка опций.

Возвращаемое значение:
    0 - файл отвечает заданным критериям,
    > 0 - файл НЕ отвечает заданным критериям,
    < 0 - в процессе работы возникла ошибка

В случае, если произошла ошибка, переменная errno должна устанавливаться
в соответствующее значение.
*/

#endif

```

Листинг 3 – plugin_api.h