ФЕДЕРАЛЬНО ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Национальный исследовательский университет ИТМО**»**

Факультет безопасности информационных технологий

ITMO UNIVERSITY

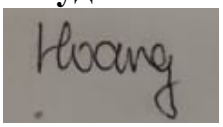
Дисциплина:

«Основы системного программирования»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1-2

Выполнил:

Студент:



Чан Ван Хоанг

Группы: N3249

Преподаватель:
Грозов В.А.

Санкт Петербург
2021

Вариант 14: Опция: --exe <значение>
Назначение: поиск исполняемых файлов в форматах ELF, PE32, a.out и COFF. Признаком формата считать соответствующее магическое число в заголовке файла. Значением опции является строка, в которой перечисляются через запятую без пробелов форматы, которые требуется найти.
Пример: --exe pe32,elf,a.out

I. Исходные тексты программ с комментариями

1. Makefile

```
CFLAGS=-Wall -Wextra -Werror -O2
TARGETS=lab1test lab1tvhN3249 libtvhN3249.so libtvhN3249-2.so

.PHONY: all clean

all: $(TARGETS)

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

lab1test: lab1test.c plugin_api.h
	gcc $(CFLAGS) -o lab1test lab1test.c -ldl

lab1tvhN3249: lab1tvhN3249.c plugin_api.h
	gcc $(CFLAGS) -o lab1tvhN3249 lab1tvhN3249.c -ldl

libtvhN3249.so: libtvhN3249.c plugin_api.h
	gcc $(CFLAGS) -shared -fPIC -o libtvhN3249.so libtvhN3249.c -ldl -lm

libtvhN3249-2.so: libtvhN3249-2.c plugin_api.h
	gcc $(CFLAGS) -shared -fPIC -o libtvhN3249-2.so libtvhN3249-2.c -ldl -lm
```

2. valgrind

```
==13090== HEAP SUMMARY:
==13090==     in use at exit: 0 bytes in 0 blocks
==13090==   total heap usage: 62 allocs, 62 frees, 219,215 bytes allocated
==13090==
==13090== All heap blocks were freed -- no leaks are possible
==13090==
==13090== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

3. lab1tvhN3249.c

```
#include <errno.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <dlfcn.h>
```

```
#include <dirent.h>
#include "plugin_api.h"

typedef int (*ppf_func_t)(const char*, struct option*, size_t); //for plugin_process_file
typedef int (*pgi_func_t)(struct plugin_info*); // for plugin_get_info;

struct longopt {
   struct option *all_opt ; //array for all longopt
   size_t all_opt_len ;
   struct option *opts_to_pass ;
   size_t opts_to_pass_len ;
   ppf_func_t func;
   pgi_func_t info;
};

int count_so (const char* dirname, int* len) {
   DIR* dir = opendir(dirname);
   if (dir == NULL) {
      fprintf(stderr, " ERROR: No directory %s\n", dirname);
      return -1;
   }
   struct dirent* entity;
   entity = readdir(dir);
   *len = 0;
   while (entity != NULL) {
      int flen = strlen(entity->d_name);
      if ((entity->d_type == DT_REG) && (entity->d_name[flen-1] == 'o') && (entity-
>d_name[flen-2] == 's') && (entity->d_name[flen-3] == '.')) {
         (*len)++;
         fprintf(stdout,"lib_name: %s\n",entity->d_name);
      }
      entity = readdir(dir);
   }
   closedir(dir);
   return 0;
}

int option_p (const char* dirname, void* dl[], int len) {
   DIR* dir = opendir(dirname);
   if (dir == NULL) {
      fprintf(stderr, " ERROR: No directory %s\n", dirname);
      return -1;
   }
   struct dirent* entity;
   entity = readdir(dir);
   int index = 0;
```

```
    while (entity != NULL && index < len) {
        int flen = strlen(entity->d_name);
        if ((entity->d_type == DT_REG) && (flen > 3) && (entity->d_name[flen-1] == 'o') &&
(entity->d_name[flen-2] == 's') && (entity->d_name[flen-3] == '.')) {
            size_t file_name_len = strlen(dirname) + strlen(entity->d_name) + 2 ;
            char* file_name = malloc(file_name_len);
            sprintf(file_name, "%s/%s", dirname, entity->d_name);
            dl[index] = dlopen(file_name, RTLD_LAZY);
            if (dl[index] == NULL) {
                fprintf(stderr, "ERROR: Failed to dlopen %s\n%s\n", entity->d_name, dlerror());
                return -1;
            }
            else {
                index++;
            }
            free(file_name);

        }
        entity = readdir(dir);
    }
    closedir(dir);
    return 0;
}

//fun file recursive search;
int res_file (const char* dirname, int tlen, struct longopt sup_all[], int is_or, int is_not) {
    DIR* dir = opendir(dirname);

    if(dir !=NULL ){

        struct dirent* entity;
        entity = readdir(dir);
        while (entity != NULL) {

            if(strcmp(entity->d_name, ".") != 0 && strcmp(entity->d_name, "..") != 0){
                // printf("lol : %s\n",entity->d_name);
                size_t path_len = strlen(dirname) + strlen(entity->d_name) + 2;
                char* path = malloc(path_len);
                snprintf(path, path_len, "%s/%s", dirname, entity->d_name);

                if(entity->d_type == DT_DIR){
                    int res = res_file(path, tlen, sup_all, is_or, is_not);
                    if (res){
                        free(path);
                        return -1;
                    }
```

```
            }

            if(entity->d_type == DT_REG) {
                int ret_true = 0; // if plugin retrun true ret++;
                int plugins_call = 0; //_count the number of plugins called
                for (int i=0; i < tlen; i++){
                    if(sup_all[i].opts_to_pass_len > 0) {
                        plugins_call++ ;
                        int ret_fun = sup_all[i].func(path, sup_all[i].opts_to_pass,
sup_all[i].opts_to_pass_len);
                     //   fprintf(stdout, "%d %d \n", i , ret_fun);
                        if (ret_fun == 0) ret_true++;
                        if (ret_fun < 0){
                            free(path);
                            errno = 0;
                            fprintf(stdout, "Error information: %s\n", strerror(errno));
                            return -1;
                        }
                    }
                }

                if(plugins_call){
                    // short_opt A and no opt;
                    if ( ret_true == plugins_call && is_or == 0 && is_not == 0) fprintf(stdout,
"%s\n", entity->d_name);
                    //short_opt O;
                    else if (ret_true > 0 && is_or == 1 && is_not == 0) fprintf(stdout, "%s\n", entity-
>d_name);
                    //short_opt NA;
                    else if (ret_true < plugins_call && is_or ==0 && is_not ==1) fprintf(stdout,
"%s\n", entity->d_name);
                    //short_opt NO;
                    else if (ret_true == 0 && is_or == 1 && is_not == 1) fprintf(stdout, "%s\n",
entity->d_name);

                }


            }

            free(path);
        }
        entity = readdir(dir);
    }

    closedir(dir);
```

```
    }
    return 0;
}

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

    struct longopt *sup_all = 0;
    char *f_name = 0;
    opterr = 0;

    int  is_o = 0, is_n = 0 , is_v = 0, is_h = 0 ,is_P = 0 ;
    // short_option A is_a = 1 (if is_o == 0 and is_n == 0)

    int len = 0;

    void** dl = 0;

    char **new_argv = (char**) malloc (argc * sizeof(char*));
    if(!new_argv){
        fprintf(stdout,"ERROR: could not allocate for argv copy\n");
    }

    memcpy(new_argv, argv, argc * sizeof(char*));

    // Minimum number of arguments is 2:
    // $ program_name --opts file to ch
    if (argc < 2) {
        fprintf(stdout, "Usage: ./main -short_opt --[options_for_lib] /path/to/file\n");
        fprintf(stdout, "Short_options:\n");
                fprintf(stdout, "\t\t-P: Plugin directory\n");
                fprintf(stdout, "\t\t-A: Combine plugin options using the 'AND' operation\n");
                fprintf(stdout, "\t\t-O: Combine plugin options using the 'OR' operation.\n");
                fprintf(stdout, "\t\t-N: Inverting the search term (after combining options plugins with -
A or -O)\n");
                fprintf(stdout, "\t\t-v: Displaying performer's full name, group number,lab version
number o\n");
                fprintf(stdout, "\t\t-h: Display help for options.\n");
        fprintf(stdout, "Long_options in plugin:\n");
        is_h = 1;
        goto START;

    }

    int ret_shrt = 0;
    while((ret_shrt = getopt(argc,new_argv, "P:vhAON"))!=-1) {
        switch(ret_shrt) {
```

```
case 'P':
   fprintf(stdout, "Short option -%c is detected!\n", ret_shrt);
   is_P = 1;
   if (count_so(optarg, &len)){
      fprintf(stderr, "ERROR: unable to count file.so\n");
      goto END;
   }

   dl = calloc (len, sizeof(void*));
   if (option_p(optarg, dl, len)){
      fprintf(stderr, "ERROR: unable to open file.so\n");
      goto END;
   }
   if( optind == argc ) {
      is_h = 1;
      goto START;
   }
   if(argv[optind][1] ==  '-') {

     goto START;
   }

   break;
case 'v':
   fprintf(stdout, "Short option -%c is detected!\n", ret_shrt);
   is_v = 1 ;
   if( optind == argc ) goto START;
   if( argv[optind][1] ==  '-' ) goto START;
   break;
case 'h':
   fprintf(stdout, "Short option -%c is detected!\n", ret_shrt);
   is_h = 1;
   if( optind == argc ) goto START;
   if( argv[optind][1] ==  '-' ) goto START;
   break;
case 'A':
   fprintf(stdout, "Short option -%c is detected!\n", ret_shrt);
   if(argv[optind][1] ==  '-') goto START;
   break;
case 'O':
   fprintf(stdout, "Short option -%c is detected!\n", ret_shrt);
   is_o = 1;
   if(argv[optind][1] ==  '-') goto START;
   break;
case 'N':
   fprintf(stdout, "Short option -%c is detected!\n", ret_shrt);
```

```
            is_n = 1;
            if(argv[optind][1] ==  '-') goto START;
            break;

    }

}


START:

if(!is_P) {
    if (count_so(".", &len)) {
        fprintf (stderr, "ERROR: unable to count in curren dir\n");
        goto END;
    }
    dl = calloc (len, sizeof(void*));
    if (option_p(".", dl, len)){
        fprintf (stderr,"ERROR: unable to dlopen libs in a current dir\n");
        goto END;
    }
}

sup_all = calloc (len, sizeof(struct longopt));

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

    // Check for plugin_get_info()
    sup_all[i].info = dlsym(dl[i], "plugin_get_info");
    if(!sup_all[i].info){
        fprintf(stderr, "ERROR: dlsym() failed: %s\n", dlerror());
        goto END;
    }
    struct plugin_info pi = {0};
    if (sup_all[i].info(&pi) < 0){
        fprintf(stderr, "ERROR: plugin_get_info() failed\n");
        goto END;
    }
    if (pi.sup_opts_len == 0){
        fprintf(stderr, "ERROR: library supports no options! How so?\n");
        goto END;
    }

    // Plugin info and printf short option v , h;

    if(is_v == 1){
```

```
    fprintf(stdout, "Plugin purpose:\t\t%s\n", pi.plugin_purpose);
    fprintf(stdout, "Plugin author:\t\t%s\n", pi.plugin_author);
    if (i == len -1)goto END;
}

if(is_h == 1){
    fprintf(stdout, "Supported options: ");
    if (pi.sup_opts_len > 0) {
        fprintf(stdout, "\n");
        for (size_t j = 0; j < pi.sup_opts_len; j++) {
            fprintf(stdout, "\t--%s\t\t%s\n", pi.sup_opts[j].opt.name, pi.sup_opts[j].opt_descr);
            }
        }
    else{
        fprintf(stdout, "none (!?)\n");
    }
    fprintf(stdout, "\n");
    if (i == len - 1) goto END;
}

// Get pointer to plugin_process_file()

sup_all[i].func = dlsym(dl[i], "plugin_process_file");
if(!sup_all[i].func) {
    fprintf(stderr, "ERROR: no plugin_process_file() function found\n");
    goto END;
}

// Prepare array of options for getopt_long

sup_all[i].all_opt_len = pi.sup_opts_len;
sup_all[i].all_opt = calloc(pi.sup_opts_len + 1, sizeof(struct option));

if (!sup_all[i].all_opt){
    fprintf(stderr, "ERROR: calloc() failed:%s\n", strerror(errno));
    goto END;
}

// copy option information

for (size_t j = 0; j < pi.sup_opts_len; j++) {
    memcpy(&sup_all[i].all_opt[j], &pi.sup_opts[j].opt, sizeof(struct option));
}

// Prepare array of actually used options that will be passed to
// plugin_process_file() (Maximum pi.sup_opts_len options)
```

```
        sup_all[i].opts_to_pass_len = 0;
        sup_all[i].opts_to_pass = calloc(pi.sup_opts_len, sizeof(struct option));
        if(!sup_all[i].opts_to_pass) {
            fprintf(stderr, "ERROR: calloc() failed: %s\n", strerror(errno));
            goto END;
        }
    }


    // Now process options for the lib


    for (int i = 0; i < len; i++) {
        optind = 1;
        memcpy(new_argv, argv, argc * sizeof(char*));
        while (1){
            int opt_ind = 0;
            int ret = getopt_long_only(argc, new_argv, "", sup_all[i].all_opt, &opt_ind);

            if (ret == -1) break;
            if(ret != 0) continue;

            // Check how many options we got up to this moment
            if ((size_t) sup_all[i].opts_to_pass_len == sup_all[i].all_opt_len){
                fprintf(stderr, "ERROR: too many options!\n");
                goto END;
            }

            // Add this option to array of options actually passed to plugin_process_file()
            memcpy(sup_all[i].opts_to_pass + sup_all[i].opts_to_pass_len, sup_all[i].all_opt +
opt_ind, sizeof(struct option));


            // Argument (if any) is passed in flag
            if ((sup_all[i].all_opt + opt_ind)->has_arg) {
                // Mind this!
                // flag is of type int*, but we are passing char* here (it's ok to do so).
                (sup_all[i].opts_to_pass + sup_all[i].opts_to_pass_len)->flag = (int*)strdup(optarg);
            }

            sup_all[i].opts_to_pass_len++;
        }


    }
```

```
if (getenv("LAB1DEBUG")) {
   for (int i = 0; i < len; i++){
      fprintf(stderr, "DEBUG: opts_to_pass_len = %ld\n", sup_all[i].opts_to_pass_len);
      for (size_t j = 0; j < sup_all[i].opts_to_pass_len; j++) {
      fprintf(stderr, "DEBUG: passing option '%s' with arg '%s'\n",
         (sup_all[i].opts_to_pass[j]).name,
         (char*)(sup_all[i].opts_to_pass[j]).flag);
      }
   }
}

fprintf(stdout, "The options are passed to libs!!! \n");
fprintf(stdout,"-------------------------------------\n");
fprintf(stdout,"The list of files that satisfy the requirements is:\n");

// Call fun recursive search and plugin_process_file()


errno = 0;
f_name = strdup(argv[argc-1]);
int ret_main = res_file(f_name, len, sup_all, is_o, is_n);
fprintf(stdout,"-------------------------------------\n");
fprintf (stdout, "fun res_file() returned %d\n",ret_main);
if(ret_main < 0){
   fprintf(stderr, "error infomation: %s\n", strerror(errno));
}

END:
if (sup_all){
   for (int i = 0; i < len; i++){
      for(size_t j = 0; j < sup_all[i].opts_to_pass_len; j++){
         if (sup_all[i].opts_to_pass[j].flag) free(sup_all[i].opts_to_pass[j].flag);
      }
      if (sup_all[i].opts_to_pass) free(sup_all[i].opts_to_pass);
      if (sup_all[i].all_opt) free(sup_all[i].all_opt);
   }
   free(sup_all);
}

if(new_argv) free(new_argv);
if(f_name) free(f_name);
if (dl){
   for (int i = 0; i < len; i++){
      if(dl[i]) dlclose(dl[i]);
   }
```

```
        free (dl);
    }

    return 0;
}
```

4. libtvhN3249.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <dirent.h>

#include "plugin_api.h"
static char *g_lib_name = "libtvhN3249.so";

static char *g_plugin_purpose = "Find type of file";

static char *g_plugin_author = "Tran Van Hoang - N3249";

#define OPT_exe "exe"
static struct plugin_option g_po_arr[] = {
/*
    struct plugin_option {
        struct option {
            const char *name;
            int         has_arg;
            int        *flag;
            int         val;
        } opt,
        char *opt_descr
    }
*/

    {
        {
            OPT_exe,
            required_argument,
            0, 0,
        },
```

```
        "File file with type elf, pe32, coff, a.out "
    },

};
static int g_po_arr_len = sizeof(g_po_arr)/sizeof(g_po_arr[0]);


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;
}

static char *input_type = NULL;


int plugin_process_file(const char *fname,
        struct option in_opts[],
        size_t in_opts_len) {

    int ret = -1;


    char *DEBUG = getenv("LAB1DEBUG");

    if (!fname || !in_opts || !in_opts_len) {
        errno = EINVAL;
        return -1;
    }
    int got_input_type = 0;
    int tmp_type = 0;

//check value op (type input)
#define OPT_CHECK(opt_var, tmp_type) \
    if (got_##opt_var) { \
        if (DEBUG) { \
            fprintf(stderr, "DEBUG: %s: Option '%s' was already supplied\n", \
                g_lib_name, in_opts[i].name); \
        } \
```

```
    errno = EINVAL; \
    return -1; \
} \
else { \
    char *endptr = NULL; \
    tmp_type = strtol((char*)in_opts[i].flag, &endptr, 10); \
    if(tmp_type!=0){\
        if (DEBUG) { \
                fprintf(stderr, "DEBUG: %s: Failed to convert '%s'\n", \
                g_lib_name, (char*)in_opts[i].flag); \
                } \
                errno = EINVAL; \
                return -1; \
    }\
    opt_var=endptr; \
    got_##opt_var = 1; \
}

for (size_t i = 0; i < in_opts_len; i++) {
    if (!strcmp(in_opts[i].name, OPT_exe)) {
        OPT_CHECK(input_type, tmp_type)
    }
    else {
        errno = EINVAL;
        return -1;
    }
}
FILE *fp;
// Get file name from user. The file should be
// either in current folder or complete path should be provided
// Open the file
fp = fopen(fname, "rb");
static unsigned char magic[4];
// Check if file exists
if (fp == NULL) {
        fprintf(stderr, "Error : Failed to open entry file - %s\n", strerror(errno));
        return -1;

}

size_t k = fread(magic,1,4,fp);
k++;
fclose(fp);
// Extract characters from file and store in character c
int loop;
int i;
```

```
   i=0;
   loop=0;
   static char output[9];
   while(magic[loop] != '\0')
   {
      sprintf((char*)(output+i),"%02x", magic[loop]);

      loop+=1;
      i+=2;
   }
   //insert NULL at the end of the output string
   output[i++] = '\0';
   char c[100];
   strcpy(c,input_type);
   if (strcmp(output,"7f454c46")==0 && strstr(c,"elf")!=NULL) {
         return ret = 0;
   }
   if (strstr(output,"ffd8ff")!=NULL && strstr(c,"jpeg")!= NULL) {
      return ret = 0;
   }
   if (strstr(output,"424d") != NULL && strstr(c,"bmp")!= NULL) {
      return ret = 0;
   }
   if (strstr(output,"47494638")!=NULL && strstr(c,"gif")!=NULL) {
      return ret = 0;
   }
   if (strstr(output, "89504e47") != NULL && strstr(c,"png") != NULL) {
      return ret = 0;
   }
   else if (strstr(output,"4d5a")!=NULL && strstr(c,"pe32")!= NULL) {
         return ret = 0;
   }
   // else if (strstr(output,"4c01")> 0) {
   //     if (strstr(input_type, "coff")!=NULL)
   //         return ret = 0;
   //     else return ret = 1;
   // }
   else if ((strstr(output,"0410")!=NULL || strstr(output,"0413")!=NULL) && strstr(c,"a.out")!=
NULL) {
         return ret = 0;
   }
   else return ret = 1;

}
```

5. libtvhN3249-2.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <dirent.h>
#include "plugin_api.h"
static char *g_lib_name = "libtvhN3249-2.so";

static char *g_plugin_purpose = "Find all file with size smaller than input size";

static char *g_plugin_author = "Tran Van Hoang - N3249";

#define OPT_size "size"

static struct plugin_option g_po_arr[] = {
/*
   struct plugin_option {
      struct option {
         const char *name;
         int         has_arg;
         int        *flag;
         int         val;
      } opt,
      char *opt_descr
   }
*/
   {
      {
         OPT_size,
         required_argument,
         0, 0,
      },
      "Find file with size smaller than input size"
   },
};

static int g_po_arr_len = sizeof(g_po_arr)/sizeof(g_po_arr[0]);

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;
}
static int input_size = 0;

int plugin_process_file(const char *fname,
        struct option in_opts[],
        size_t in_opts_len) {
    int ret = -1;

    char *DEBUG = getenv("LAB1DEBUG");

    if (!fname || !in_opts || !in_opts_len) {
        errno = EINVAL;
        return -1;
    }

    g_lib_name="libtvhN3249-2.so";

    int got_input_size = 0;

    #define OPT_CHECK(opt_var) \
    if (got_##opt_var) { \
        if (DEBUG) { \
            fprintf(stderr, "DEBUG: %s: Option '%s' was already supplied\n", \
                g_lib_name, in_opts[i].name); \
        } \
        errno = EINVAL; \
        return -1; \
    } \
    else { \
        char *endptr = NULL; \
        opt_var = strtol((char*)in_opts[i].flag, &endptr, 10); \
        if (strcmp(endptr,"")!=0) { \
                    if (DEBUG) { \
                    fprintf(stderr, "DEBUG: %s: Failed to convert '%s'\n", \
                    g_lib_name, (char*)in_opts[i].flag); \
```

```
                } \
                errno = EINVAL; \
                return -1; \
    } \
    got_##opt_var = 1; \
  }

  for (size_t i = 0; i < in_opts_len; i++) {
    if (!strcmp(in_opts[i].name, OPT_size)) {
      OPT_CHECK(input_size)
    }
      else {
      errno = EINVAL;
      return -1;
    }
  }

  if (!got_input_size) {
    if (DEBUG) {
      fprintf(stderr, "DEBUG: %s: The input size value was not supplied.\n",
        g_lib_name);
    }
    errno = EINVAL;
    return -1;
  }



  FILE *fp;
  fp = fopen(fname,  "rb");
  if (fp == NULL) {
      fprintf(stderr, "Error : Failed to open entry file - %s\n", strerror(errno));
      return -1;

  }
  fseek(fp, 0L, SEEK_END);
  int res = ftell(fp);
  fclose(fp);
  if (res < input_size)
    return ret = 0;
  else return ret = 1;

}
```

6. lab1test.c

```
// Test lab 1 .so files for formal conformance
```

```
//
// Compile with:
// gcc -o lab1test lab1test.c -ldl
//
// (c) Alexei Guirik, 2021
// This source is licensed under CC BY-NC 4.0
// (https://creativecommons.org/licenses/by-nc/4.0/)
//

#include <stdio.h>
#include <string.h>
#include <dlfcn.h>

#include "plugin_api.h"


int main(int argc, char *argv[]) {
    if (argc == 1) {
        fprintf(stdout, "Usage: lab1test /path/to/lib1.so ... libN.so\n");
        return 0;
    }

    // Try all names that are passed via argv
    for (int i = 1; i < argc; i++) {
        fprintf(stdout, "Trying %s:\n", argv[i]);

        struct plugin_info pi = {0};

        void *dl = dlopen(argv[i], RTLD_LAZY);
        if (!dl) {
            fprintf(stderr, "ERROR: dlopen() failed: %s\n", dlerror());
            continue;
        }

        // Check for plugin_get_info() func
        void *func = dlsym(dl, "plugin_get_info");
        if (!func) {
            fprintf(stderr, "ERROR: dlsym() failed: %s\n", dlerror());
            goto END;
        }

        typedef int (*pgi_func_t)(struct plugin_info*);
        pgi_func_t pgi_func = (pgi_func_t)func;

        int ret = pgi_func(&pi);
        if (ret < 0) {
```

```
            fprintf(stderr, "ERROR: plugin_get_info() failed\n");
            goto END;
        }

        // Plugin info
        fprintf(stdout, "Plugin purpose:\t\t%s\n", pi.plugin_purpose);
        fprintf(stdout, "Plugin author:\t\t%s\n", pi.plugin_author);
        fprintf(stdout, "Supported options: ");
        if (pi.sup_opts_len > 0) {
            fprintf(stdout, "\n");
            for (size_t i = 0; i < pi.sup_opts_len; i++) {
                fprintf(stdout, "\t--%s\t\t%s\n", pi.sup_opts[i].opt.name, pi.sup_opts[i].opt_descr);
            }
        }
        else {
            fprintf(stdout, "none (!?)\n");
        }

        // Warn if plugin_process_file() is not found
        func = dlsym(dl, "plugin_process_file");
        if (!func) {
            fprintf(stderr, "WARNING: no plugin_process_file() function found\n");
        }

        END:
        if (dl) dlclose(dl);
    }
    return 0;
}
```

7. plugin_api.h

```
#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 плагином нельзя).
*/

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

   Фунция, 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

II. Примеры работы программы:

1. LAB1DEBUG=1 ./lab1tvhN3249 --exe elf,pe32 ./file_test

```
└─$ LAB1DEBUG=1 ./lab1tvhN3249 --exe elf,pe32  ./file_test
lib_name: libtvhN3249-2.so
lib_name: libtvhN3249.so
DEBUG: opts_to_pass_len = 0
DEBUG: opts_to_pass_len = 1
DEBUG: passing option 'exe' with arg 'elf,pe32'
The options are passed to libs!!!
-------------------------------------
The list of files that satisfy the requirements is:
LOLPRO 11.24.2.exe
j6a9ot.exe
libnqtN3251-2.so
libnqtN3251.so
lab1test
libnqtN3251-2.so
-------------------------------------
fun res_file() returned 0
```

2. LAB1DEBUG=1 ./lab1tvhN3249 --size 1000 ./file_test

```
└─$ LAB1DEBUG=1 ./lab1tvhN3249 --size 1000 ./file_test
lib_name: libtvhN3249-2.so
lib_name: libtvhN3249.so
DEBUG: opts_to_pass_len = 1
DEBUG: passing option 'size' with arg '1000'
DEBUG: opts_to_pass_len = 0
The options are passed to libs!!!
-------------------------------------
The list of files that satisfy the requirements is:
123.txt
cde.txt
abc.txt
bcd.txt
-------------------------------------
fun res_file() returned 0
```

3. LAB1DEBUG=1 ./lab1tvhN3249 -A --exe elf,pe32 --size 20000 ./file_test

```
└─$ LAB1DEBUG=1 ./lab1tvhN3249 -A --exe elf,pe32 --size 20000 ./file_test
Short option -A is detected!
lib_name: libtvhN3249-2.so
lib_name: libtvhN3249.so
DEBUG: opts_to_pass_len = 1
DEBUG: passing option 'size' with arg '20000'
DEBUG: opts_to_pass_len = 1
DEBUG: passing option 'exe' with arg 'elf,pe32'
The options are passed to libs!!!
-------------------------------------
The list of files that satisfy the requirements is:
libnqtN3251-2.so
libnqtN3251.so
lab1test
libnqtN3251-2.so
-------------------------------------
fun res_file() returned 0
```

4. LAB1DEBUG=1 ./lab1tvhN3249 -O --exe elf,pe32 --size 1000 ./file_test

```
Short option -O is detected!
lib_name: libtvhN3249-2.so
lib_name: libtvhN3249.so
DEBUG: opts_to_pass_len = 1
DEBUG: passing option 'size' with arg '1000'
DEBUG: opts_to_pass_len = 1
DEBUG: passing option 'exe' with arg 'elf,pe32'
The options are passed to libs!!!
-------------------------------------
The list of files that satisfy the requirements is:
LOLPRO 11.24.2.exe
j6a9ot.exe
libnqtN3251-2.so
123.txt
cde.txt
abc.txt
bcd.txt
libnqtN3251.so
lab1test
libnqtN3251-2.so
-------------------------------------
fun res_file() returned 0
```

5. LAB1DEBUG=1 ./lab1tvhN3249 -A -N --exe elf,pe32 –size 20000 ./file_test

```
└─$ LAB1DEBUG=1 ./lab1tvhN3249 -A -N --exe elf,pe32 --size 20000 ./file_test
Short option -A is detected!
Short option -N is detected!
lib_name: libtvhN3249-2.so
lib_name: libtvhN3249.so
DEBUG: opts_to_pass_len = 1
DEBUG: passing option 'size' with arg '20000'
DEBUG: opts_to_pass_len = 1
DEBUG: passing option 'exe' with arg 'elf,pe32'
The options are passed to libs!!!
-------------------------------------
The list of files that satisfy the requirements is:
LOLPRO 11.24.2.exe
j6a9ot.exe
exJPEG.jpg
123.txt
cde.txt
abc.txt
expBMP.bmp
bcd.txt
expGIF.gif
expPNG.png
-------------------------------------
fun res_file() returned 0
```

6. LAB1DEBUG=1 ./lab1tvhN3249 -P ./file_test --lines-count 8 lines-count-comp ne file_test

```
└─$ LAB1DEBUG=1 ./lab1tvhN3249 -P ./file_test --lines-count 8 lines-count-comp ne file_test
Short option -P is detected!
lib_name: libnqtN3251.so
lib_name: libnqtN3251-2.so
DEBUG: opts_to_pass_len = 1
DEBUG: passing option 'lines-count' with arg '8'
DEBUG: opts_to_pass_len = 0
The options are passed to libs!!!
-------------------------------------
The list of files that satisfy the requirements is:
abc.txt
bcd.txt
-------------------------------------
fun res_file() returned 0
```