Файл executor.c

```
#include "executor.h"

//control_struct control_word;

char preferences[PARAM_QUANTITY][PATH_MAX];
bool checkbox[CHECKBOXES_QUANTITY];

void read_settings() {
    FILE *settings = fopen(SETTINGS_PATH, "rb");
    if (settings == NULL)
        return;
    for (size_t i = 0;i < ARRAY_SIZE(preferences);i++) {
        fread(preferences[i], sizeof(char), PATH_MAX, settings);
    }
    fread(checkbox, sizeof(bool), CHECKBOXES_QUANTITY, settings);
}

void write_settings() {
    FILE *settings = fopen(SETTINGS_PATH, "wb");
    if (settings == NULL) {
        perror("Cannot open or create settings file");
        exit(errno);
    }
    for (size_t i = 0;i < ARRAY_SIZE(preferences);i++) {
        fwrite(preferences[i], sizeof(char), PATH_MAX, settings);
    }
    fwrite(checkbox, sizeof(bool), CHECKBOXES_QUANTITY, settings);
}

void create_exec_str(char* buf, char* path, char* query) {
    char parser_buf[PATH_MAX] = "";
    bool checkbox_flag = false;
    static char parser_query[PATH_MAX];
    FILE *fpipe;
    strcpy(buf, "find");

    strcpy(parser_query, "./parser ");
    if (path != NULL) {
        strcat(buf, " ");
        strcat(buf, path);
    }
    for (size_t i = 0;i < CHECKBOXES_QUANTITY;i++) {
        if (checkbox[i] == true) {
            if (!checkbox_flag) {
                checkbox_flag = true;
                strcat(buf, " -type ");
            }
            strcat(buf, checkboxes_tokens[i]);
            strcat(buf, ",");
        }
    }
    if (checkbox_flag)
        buf[strlen(buf) - 1] = '\0';

    if (query != NULL) {
        strcat(buf, " ");
        strcat(parser_query, query);
        if (0 == (fpipe = (FILE *)popen(parser_query, "r")))
        {
            perror("popen() failed");
```

```
            exit(EXIT_FAILURE);
        }
        fgets(parser_buf, sizeof parser_buf, fpipe);
        strcat(buf, parser_buf);
        pclose(fpipe);
    }
    strcat(buf, " 2>/dev/null");

}
FILE *get_query_result_file(char* path) {
    char command_buffer[PATH_MAX] = "";
    char query[PATH_MAX] = "";
    char buffer[256] = "";
    size_t current_pos = 0;
    size_t prev_pos = 0;
    FILE *fpipe;
    strcpy(query, "'");

    while (1) {
        if (preferences[get_index_by_param(QUERY_FORMAT)][current_pos] == '&'
            || preferences[get_index_by_param(QUERY_FORMAT)][current_pos] ==
'|'
            || preferences[get_index_by_param(QUERY_FORMAT)][current_pos] ==
'\0') {
            strncpy(buffer, preferences[get_index_by_param(QUERY_FORMAT)] +
prev_pos, current_pos - prev_pos);
            strcat(query, buffer);
            strcat(query, " ");
            for (size_t i = 0;i < ARRAY_SIZE(tokens);i++) {
                if (strstr(buffer, tokens[i])) {
                    strcat(query, preferences[i]);
                    strcat(query, " ");
                    break;
                }
            }
            if (preferences[get_index_by_param(QUERY_FORMAT)][current_pos] ==
'\0')
                break;
            prev_pos = current_pos;
        }
        current_pos++;
    }
    strcat(query, "'");


    create_exec_str(command_buffer, path, query);

    if (0 == (fpipe = (FILE *)popen(command_buffer, "r")))
    {
        perror("popen() failed");
        exit(EXIT_FAILURE);
    }

    return fpipe;
}
```

Файл executor.h

```
#ifndef EXECUTOR_H
#define EXECUTOR_H
#ifndef _DEFAULT_SOURCE
#define _DEFAULT_SOURCE
```

```
#endif
#include <linux/limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "../utility/utility.h"
#include "../config.h"
#include <errno.h>

static const char *const flag_str_name[] = {
    "-name",
    "-group",
    "-path",
    "-perm",
    "-regex",
    "-size",
    "-user",
    "-type",
};

typedef enum PARAMETR {
    NO_PARAM,
    NAME,
    GROUP,
    PATH,
    PERM,
    REGEX,
    SIZE,
    USER,
    UID,
    QUERY_FORMAT,
    PARAMETERS_END,
} PARAMETR;

typedef enum CHECKBOXES {
    TYPE_D,
    TYPE_F,
    TYPE_L,
    CHECKBOXES_END,
} CHECKBOXES;

static const char *const checkboxes_tokens[] = {
    "d",
    "f",
    "l",
};

static const char *const tokens[] = {
    "{name}",
    "{group}",
    "{path}",
    "{perm}",
    "{regex}",
    "{size}",
    "{user}",
    "{uid}",
};

#define get_index_by_param(__X) __X-1
#define get_str_opt_by_param(__X) flag_str_name[get_index_by_param(__X)]
#define PARAM_QUANTITY PARAMETERS_END-1
```

```
#define CHECKBOXES_QUANTITY CHECKBOXES_END

/* typedef struct parametr {
    PARAMETR flag;
    char *expr;
}parametr;

typedef struct control_struct {
    parametr options[PARAM_QUANTITY];
}control_struct; */

//Записать параметры в файл
void write_settings();

//Считать параметры из файла
void read_settings();

//Создать строку вызова find
void create_exec_str(char* buf, char* path, char* query);

//Получить результирующий файл
FILE *get_query_result_file(char* path);
#endif
```

Файл about.c

```
#include "about.h"


extern int ch;
extern screen_size scr_size;
extern bool exit_flag;
static WINDOW *box_win = NULL;

//Обработчик отрисовки окна "О программе"
static void render_about_window();

//Обработчик события изменения размера окна
static void on_about_resize_handler();

//Обработчик события выхода в основное окно
static void on_about_exit_handler();

//Порядок отрисовки окна "О программе"
static void about_refresher_handler();

/*Закрепление обработчиков за конкретными событиями(клавишами)*/
static const key_handler ABOUT_CONTROL_KEYS_HANDLERS[] = {
    {KEY_RESIZE, on_about_resize_handler},
    {KEY_F(3), on_about_exit_handler},
    {KEY_F(1),settings}
};

/*Констатный массив очереди отрисовки*/
static const render_routes ABOUT_RENDER_LIST[] = {
    render_key_map,
    render_about_window,
};

void about() {
    exit_flag = FALSE;
    refresh();
```

```
    about_refresher_handler();
    while ((ch = wgetch(box_win))) {
        default_key_handler(ABOUT_CONTROL_KEYS_HANDLERS,
ARRAY_SIZE(ABOUT_CONTROL_KEYS_HANDLERS));
        if (exit_flag) {
            delwin(box_win);
            box_win = NULL;
            return;
        }
    }
}

static void on_about_exit_handler() {
    exit_flag = TRUE;
}

static void about_refresher_handler() {
    curs_set(0);
    getmaxyx(stdscr, scr_size.max_y, scr_size.max_x);
    for (size_t i = 0; i < ARRAY_SIZE(ABOUT_RENDER_LIST);i++) {
        ABOUT_RENDER_LIST[i]();
    }
}

static void on_about_resize_handler() {
    delwin(box_win);
    box_win = NULL;
    about_refresher_handler();
}

static void render_about_window() {
    WINDOW *description_win = newwin(scr_size.max_y - 5, 3 * scr_size.max_x /
4, 2, scr_size.max_x / 8);
    if (box_win == NULL)
        box_win = newwin(scr_size.max_y - 1, scr_size.max_x, 0, 0);
    keypad(box_win, TRUE);
    wbkgd(box_win, COLOR_PAIR(3));
    wbkgd(description_win, COLOR_PAIR(3));
    box(box_win, 0, 0);
    mvwprintw(description_win, 0, 15, PROGRAM_NAME);
    mvwaddstr(description_win, 2, 0, DESCRIPTION);
    mvwaddstr(description_win, 4, 15, DESCRIPTION1);
    mvwaddstr(description_win, 6, 0, DESCRIPTION2);
    mvwaddstr(description_win, 8, 0, DESCRIPTION3);
    mvwaddstr(description_win, 12, 15, DESCRIPTION4);
    mvwaddstr(description_win, 14, 0, DESCRIPTION5);
    mvwaddstr(description_win, 16, 0, DESCRIPTION6);
    mvwaddstr(description_win, 18, 0, DESCRIPTION7);
    mvwaddstr(description_win, 20, 0, DESCRIPTION8);
    mvwaddstr(description_win, 22, 0, DESCRIPTION9);
    mvwaddstr(description_win, 24, 0, DESCRIPTION10);
    mvwaddstr(description_win, 26, 0, DESCRIPTION11);
    mvwaddstr(description_win, 28, 0, DESCRIPTION12);
    mvwaddstr(description_win, 30, 0, DESCRIPTION13);
    wrefresh(box_win);
    wrefresh(description_win);

    delwin(description_win);

}
```

Файл about.h

```
#ifndef ABOUT_H
#define ABOUT_H
#include "../utility/utility_gui_lib.h"
#include "settings.h"

//Точка входа в обработчик событий окна "О программе"
void about();
#endif
```

Файл main_screen.c

```
#include "main_screen.h"

//Инициализация форм
static void init_form();

//Очистка выделенных ресурсов
static void cleanup();

//Завершающая рутина
static void exit_route();

//Обработчик удаления символа в форме
static void on_del_char_handler();

//Обработчик изменения размера основного окна
static void on_resize_handler();

//Обработчик нажатия клавиши KEY_DOWN
static void on_key_down_handler();

//Обработчик нажатия клавиши KEY_UP
 static void on_key_up_handler();

//Обработчик события перехода к окну "О программе"
static void on_about_handler();

//Отрисовать строку поиска
static void render_search_bar();

//Предварительная инициализация основного окна
static void main_window_gui_init();

//Обработчик события подтверждения ввода в строку поиска
static void on_submit_handler();

//Порядок отрисовки
static void refresher_handler();

//Отрисовать основное окно
static void render_main_window();

//Обработчик события перехода к окну параметров
static void on_settings_handler();

//Обработчик нажатия клавиши KEY_RIGHT
static void on_scroll_right();

//Обработчик нажатия клавиши KEY_LEFT
static void on_scroll_left();

/*Закрепление обработчиков за конкретными событиями(клавишами)*/
```

```
static const key_handler CONTROL_KEYS_HANDLERS_LIST[] = {
   {KEY_F(3),exit_route},
   {KEY_F(2), on_about_handler},
   {KEY_F(1), on_settings_handler},
   {KEY_BACKSPACE, on_del_char_handler},
   {KEY_RESIZE,on_resize_handler},
   {ENTER_KEY,on_submit_handler},
   {KEY_DOWN,on_key_down_handler},
   {KEY_UP,on_key_up_handler},
   {KEY_RIGHT,on_scroll_right},
   {KEY_LEFT,on_scroll_left},
};
/*Констатный массив очереди отрисовки*/
static const render_routes RENDER_LIST[] = {
    render_search_bar,
    render_main_window,
    render_key_map,
};

extern int ch;
extern screen_size scr_size;
static FIELD *field[2];
static FORM *search_form = NULL;
static WINDOW *search_form_win = NULL;
static buf_t last_request;
static int current_pos;
static FILE *fpipe = NULL;
static int total_lines;

//Пользовательские параметры
extern char preferences[PARAM_QUANTITY][PATH_MAX];

void render_main_window_gui() {
    main_window_gui_init();

    while ((ch = wgetch(search_form_win))) {

        default_key_handler(CONTROL_KEYS_HANDLERS_LIST,
ARRAY_SIZE(CONTROL_KEYS_HANDLERS_LIST));
        if (ch != -1) {
            form_driver(search_form, ch);
            form_driver(search_form, REQ_VALIDATION);
            curs_set(1);
        }
    }
}
static void cleanup() {
    unpost_form(search_form);
    delwin(form_sub(search_form));
    free_form(search_form);
    free_field(field[0]);
    field[0] = NULL;
    delwin(search_form_win);
    search_form_win = NULL;
    search_form = NULL;
}
static void exit_route() {
    cleanup();
    endwin();
    exit(EXIT_SUCCESS);
}
static void on_del_char_handler() {
```

```
    form_driver(search_form, REQ_DEL_PREV);
    form_driver(search_form, REQ_VALIDATION);
    curs_set(1);
}
static void on_scroll_right() {
    form_driver(search_form, REQ_NEXT_CHAR);
    curs_set(1);
}
static void on_scroll_left() {
    form_driver(search_form, REQ_PREV_CHAR);
    curs_set(1);
}
static void on_about_handler() {
    about();
    cleanup();
    refresher_handler();
}
static void on_settings_handler() {
    settings();
    cleanup();
    refresher_handler();
}
static void on_resize_handler() {
    static screen_size prev;
    getmaxyx(search_form_win, prev.max_y, prev.max_x);
    strcpy(last_request, field_buffer(field[0], 0));
    if (search_form != NULL && (prev.max_x != scr_size.max_x ||
scr_size.max_y < 6)) {
        cleanup();
    }
    refresher_handler();
    //prev = scr_size;
}
static void on_key_down_handler() {
    if (current_pos < total_lines - 1)current_pos++;
    render_main_window();
    curs_set(0);
}
static void on_key_up_handler() {
    if (current_pos > 0)current_pos--;
    render_main_window();
    curs_set(0);
}

static void refresher_handler() {

    getmaxyx(stdscr, scr_size.max_y, scr_size.max_x);
    for (size_t i = 0; i < ARRAY_SIZE(RENDER_LIST);i++) {
        RENDER_LIST[i]();
    }
}
static void main_window_gui_init() {
    refresher_handler();
    curs_set(0);
}

static void on_submit_handler() {
    strcpy(last_request, field_buffer(field[0], 0));
    if (fpipe != NULL) {
        pclose(fpipe);
        fpipe = NULL;
    }
```

```
    current_pos = 0;
    refresher_handler();
}
static void init_form() {
    if (search_form == NULL) {
        int rows = 1, cols = 1;
        field[0] = new_field(1, scr_size.max_x - 2, 0, 0, 0, 0);
        field[1] = NULL;
        set_field_back(field[0], COLOR_PAIR(3));
        set_field_fore(field[0], COLOR_PAIR(3));
        field_opts_off(field[0], O_AUTOSKIP);
        search_form = new_form(field);
        scale_form(search_form, &rows, &cols);
        search_form_win = newwin(rows + 2, cols + 2, 0, 0);
        keypad(search_form_win, TRUE);
        wbkgd(search_form_win, COLOR_PAIR(3));
        set_form_win(search_form, search_form_win);
        set_form_sub(search_form, subwin(search_form_win, rows, cols, 1, 1));
        post_form(search_form);
    }
}
static void render_search_bar() {

    screen_size current_box_size;
    getmaxyx(search_form_win, current_box_size.max_y,
current_box_size.max_x);
    init_form();
    if (scr_size.max_x != current_box_size.max_x) {
        wresize(search_form_win, 3, scr_size.max_x);
    }
    box(search_form_win, 0, 0);
    set_field_buffer(field[0], 0, last_request);
    form_driver(search_form, REQ_END_LINE);
    wrefresh(search_form_win);

}

static void render_main_window() {

    static buf_t buf;
    static WINDOW *main_pad = NULL;
    WINDOW *box_win = NULL;
    size_t i = 0;
    screen_size current_box_size;
    if (main_pad == NULL) {
        main_pad = newpad(VISIBLE_MAX, scr_size.max_x);
        wbkgd(main_pad, COLOR_PAIR(3));
    }
    box_win = newwin(scr_size.max_y - 4, scr_size.max_x, 3, 0);
    wbkgd(box_win, COLOR_PAIR(3));
    getmaxyx(main_pad, current_box_size.max_y, current_box_size.max_x);
    if ((scr_size.max_y != current_box_size.max_y) || (scr_size.max_x !=
current_box_size.max_x)) {
        // wclear(box_win);
        wresize(main_pad, VISIBLE_MAX, scr_size.max_x);
    }

    if (fpipe == NULL) {

        fpipe = get_query_result_file(trimwhitespace(field_buffer(field[0],
0)));
        wclear(main_pad);
```

```
        total_lines = 0;
    }
    while (fpipe != NULL && (total_lines - current_pos) < scr_size.max_y - 6
&& fgets(buf, sizeof buf, fpipe) && total_lines < VISIBLE_MAX) {
        mvwaddstr(main_pad, total_lines, 0, buf);
        total_lines++;
    }
    wmove(main_pad, current_pos, 0);
    winstr(main_pad, buf);
    wattron(main_pad, COLOR_PAIR(1));
    mvwaddstr(main_pad, current_pos, 0, buf);
    wattroff(main_pad, COLOR_PAIR(1));
    wmove(main_pad, current_pos + 1, 0);
    winstr(main_pad, buf);
    mvwaddstr(main_pad, current_pos + 1, 0, buf);
    box(box_win, 0, 0);
    wrefresh(box_win);
    prefresh(main_pad, current_pos, 0, 4, 1, scr_size.max_y - 3,
scr_size.max_x - 2);
    delwin(box_win);

}
```

Файл main_screen.h

```
#ifndef SCREEN_H
#define SCREEN_H
#include "../utility/utility_gui_lib.h"
#include "settings.h"
#include <form.h>
#include <locale.h>
#include "../config.h"
#include "../executor/executor.h"
#include "about.h"

//Основная точка входа в цикл событий программы
void render_main_window_gui();
#endif
```

Файл settings.c

```
#include "settings.h"

extern int ch;
extern screen_size scr_size;
extern bool exit_flag;
extern char preferences[PARAM_QUANTITY][PATH_MAX];
extern bool checkbox[CHECKBOXES_QUANTITY];

static WINDOW *settings_form_win = NULL;
static FORM *settings_form = NULL;
static size_t current_setting = 0;
static screen_size prev_size;

//Очистка выделенных ресурсов
static void cleanup();

//Обработчик события изменения размера окна параметров
static void on_settings_resize_handler();

//Обработчик события выхода в основное окно
static void on_settings_exit_handler();
```

```
//Обработчик события переключения на следующее поле в форме
static void on_settings_next_field();

//Порядок отрисовки окна параметров
static void settings_refresher_handler();

//Отрисовать окно параметров
static void render_settings();

//Обработчик события переключения на предыдущее поле
static void on_settings_prev_field();

//Обработчик события удаления символа из поля
static void on_settings_del_char_handler();

//Переход на точку входа в цикл событий окна "О программе"
static void on_about_handler();

//Обработчик событий флажков
static void on_checkbox_handler();

//Обновить буфер полей
static void update_field_buffer();

//Обработчик перехода курсора внутри поля вправо
static void on_settings_scroll_right();

//Обработчик перехода курсора внутри поля влево
static void on_settings_scroll_left();

/*Структура буфера параметров*/
typedef struct buffer_settings {
    const PARAMETR flag;
    const char *ui_name;
    char *field_buffer;
}buffer_settings;

/*Структура буфера флажков*/
typedef struct buffer_checkbox {
    const CHECKBOXES flag;
    const char *ui_name;
    bool *checked;
}buffer_checkbox;

/*Привязка флажков к системной части*/
static buffer_checkbox checkboxes[] = {
    {TYPE_F,FILE_TYPE_GUI,&checkbox[TYPE_F]},
    {TYPE_D,DIR_TYPE_GUI,&checkbox[TYPE_D]},
    {TYPE_L,SYMLINK_TYPE_GUI,&checkbox[TYPE_L]},
};

/*Привязка параметров к системной части*/
static buffer_settings fields_buffer[] = {
   {NAME,NAME_GUI,preferences[get_index_by_param(NAME)]},
   {GROUP,GROUP_GUI,preferences[get_index_by_param(GROUP)]},
   {USER,USER_GUI,preferences[get_index_by_param(USER)]},
   {REGEX,REGEXP_GUI,preferences[get_index_by_param(REGEX)]},
   {PERM,PERM_GUI,preferences[get_index_by_param(PERM)]},
   {SIZE,SIZE_GUI,preferences[get_index_by_param(SIZE)]},
```

```
{QUERY_FORMAT,QUERY_STRING_GUI,preferences[get_index_by_param(QUERY_FORMAT)]}
,
};

static FIELD *settings_field[ARRAY_SIZE(fields_buffer) + 1];

static const key_handler SETTINGS_CONTROL_KEYS_HANDLERS[] = {
    {KEY_RESIZE, on_settings_resize_handler},
    {KEY_F(3), on_settings_exit_handler},
    {KEY_F(2),on_about_handler},
    {KEY_DOWN,on_settings_next_field},
    {KEY_LEFT,on_settings_scroll_left},
    {KEY_RIGHT,on_settings_scroll_right},
    {KEY_UP,on_settings_prev_field},
    {KEY_BACKSPACE,on_settings_del_char_handler},
    {ENTER_KEY,on_checkbox_handler},
};

/*Очередь отрисовки*/
static const render_routes SETTINGS_RENDER_LIST[] = {
    render_key_map,
    render_settings,
};

void settings() {
    exit_flag = FALSE;
    settings_refresher_handler();
    while ((ch = wgetch(settings_form_win))) {
        default_key_handler(SETTINGS_CONTROL_KEYS_HANDLERS,
ARRAY_SIZE(SETTINGS_CONTROL_KEYS_HANDLERS));
        if (ch != -1) {
            form_driver(settings_form, ch);
            form_driver(settings_form, REQ_VALIDATION);
            update_field_buffer();
            curs_set(1);
            wrefresh(settings_form_win);
        }
        if (exit_flag) {
            cleanup();
            return;
        }
    }
}

static void update_field_buffer() {
    if (current_setting < ARRAY_SIZE(settings_field) - 1) {
        strcpy(fields_buffer[current_setting].field_buffer,
field_buffer(settings_field[current_setting], 0));
        trimwhitespace(fields_buffer[current_setting].field_buffer);
    }

}

static void on_settings_scroll_right() {
    if (current_setting < ARRAY_SIZE(settings_field) - 1) {
        form_driver(settings_form, REQ_NEXT_CHAR);
        curs_set(1);
        wrefresh(settings_form_win);
    }
}
```

```
static void on_settings_scroll_left() {
    if (current_setting < ARRAY_SIZE(settings_field) - 1) {
        form_driver(settings_form, REQ_PREV_CHAR);
        curs_set(1);
        wrefresh(settings_form_win);
    }
}

static void on_settings_del_char_handler() {
    if (current_setting < ARRAY_SIZE(settings_field) - 1) {
        form_driver(settings_form, REQ_DEL_PREV);
        form_driver(settings_form, REQ_VALIDATION);
        update_field_buffer();
        curs_set(1);
        wrefresh(settings_form_win);
    }
}

static void on_about_handler() {
    cleanup();
    about();
    settings_refresher_handler();
}

static void on_checkbox_handler() {
    if (current_setting > ARRAY_SIZE(settings_field) - 2) {
        *(checkboxes[current_setting - ARRAY_SIZE(settings_field) +
1].checked) = !(*(checkboxes[current_setting - ARRAY_SIZE(settings_field) +
1].checked));
        render_settings();
    }
}

static void on_settings_next_field() {
    if (current_setting < ARRAY_SIZE(settings_field) - 2 +
ARRAY_SIZE(checkboxes)) {
        current_setting++;
        render_settings();
    }
}

static void on_settings_prev_field() {
    if (current_setting > 0) {
        current_setting--;
        render_settings();
    }
}

static void settings_refresher_handler() {
    getmaxyx(stdscr, scr_size.max_y, scr_size.max_x);
    for (size_t i = 0; i < ARRAY_SIZE(SETTINGS_RENDER_LIST);i++) {
        SETTINGS_RENDER_LIST[i]();
    }
}

static void render_settings() {

    static WINDOW *box_win = NULL;
    int rows, cols;


    if (box_win == NULL) {
```

```
        box_win = newwin(scr_size.max_y - 1, scr_size.max_x, 0, 0);
        wbkgd(box_win, COLOR_PAIR(3));
        wrefresh(box_win);
    }
    if (settings_form_win == NULL) {
        settings_form_win = newpad((int)(ARRAY_SIZE(settings_field) +
ARRAY_SIZE(checkboxes)), scr_size.max_x - 2);
        wbkgd(settings_form_win, COLOR_PAIR(3));
        keypad(settings_form_win, TRUE);
    }
    unpost_form(settings_form);
    for (size_t i = 0;i < ARRAY_SIZE(settings_field) - 1;i++) {
        if (settings_field[i] == NULL)
            settings_field[i] = new_field(1, 3 * scr_size.max_x / 4, i, 1, 0,
0);

        if (current_setting == i) {
            set_field_back(settings_field[i], COLOR_PAIR(1) | A_UNDERLINE);
            set_field_fore(settings_field[i], COLOR_PAIR(1));
            wattron(settings_form_win, COLOR_PAIR(1));
            field_opts_on(settings_field[i], O_ACTIVE);
        }
        else {
            set_field_back(settings_field[i], COLOR_PAIR(3) | A_UNDERLINE);
            set_field_fore(settings_field[i], COLOR_PAIR(3));
            field_opts_off(settings_field[i], O_ACTIVE);
        }
        field_opts_off(settings_field[i], O_AUTOSKIP);

        set_field_buffer(settings_field[i], 0,
fields_buffer[i].field_buffer);
        mvwaddstr(settings_form_win, 1 + i, 1, fields_buffer[i].ui_name);
        wattroff(settings_form_win, COLOR_PAIR(1));
    }
    for (size_t i = 0;i < ARRAY_SIZE(checkboxes);i++) {
        if (current_setting == ARRAY_SIZE(settings_field) - 1 + i)
            wattron(settings_form_win, COLOR_PAIR(1));
        mvwprintw(settings_form_win, i + ARRAY_SIZE(settings_field), 1, "%s
%s", *(checkboxes[i].checked) ? "[x]" : "[ ]", checkboxes[i].ui_name);
        wattroff(settings_form_win, COLOR_PAIR(1));
    }

    settings_field[ARRAY_SIZE(settings_field) - 1] = NULL;
    if (settings_form == NULL) {
        settings_form = new_form(settings_field);
        scale_form(settings_form, &rows, &cols);
        set_form_win(settings_form, settings_form_win);
        set_form_sub(settings_form, derwin(settings_form_win, rows, cols, 1,
scr_size.max_x / 4 - 3));
    }
    post_form(settings_form);
    if ((scr_size.max_y != prev_size.max_y) || (prev_size.max_x !=
scr_size.max_x) || (scr_size.max_y - 3 < (int)(ARRAY_SIZE(settings_field) -
1))) {
        wclear(box_win);
        wresize(box_win, scr_size.max_y - 1, scr_size.max_x);
        box(box_win, 0, 0);
        wrefresh(box_win);
    }
    prefresh(settings_form_win, (int)current_setting < scr_size.max_y - 3 ? 1
: current_setting + 1, 0, 1, 1, scr_size.max_y - 3, scr_size.max_x - 2);
    curs_set(0);
```

```
    if (current_setting < ARRAY_SIZE(settings_field) - 1) {
        set_current_field(settings_form, settings_field[current_setting]);
    }

    form_driver(settings_form, REQ_END_LINE);
    prev_size = scr_size;


}

static void cleanup() {
    if (settings_form != NULL) {
        unpost_form(settings_form);
        delwin(form_sub(settings_form));
        free_form(settings_form);
        for (size_t i = 0;i < ARRAY_SIZE(settings_field);i++) {
            free_field(settings_field[i]);
            settings_field[i] = NULL;
        }
        delwin(settings_form_win);
        settings_form_win = NULL;
        settings_form = NULL;
        prev_size = (screen_size){ 0,0 };
    }
}

static void on_settings_resize_handler() {
    screen_size temp;
    getmaxyx(settings_form_win, temp.max_y, temp.max_x);
    if (temp.max_x != scr_size.max_x) {
        cleanup();
    }
    settings_refresher_handler();
}

static void on_settings_exit_handler() {
    write_settings();
    exit_flag = TRUE;
}
```

Файл settings.h

```
#ifndef SETTINGS_H
#define SETTINGS_H
#include "../utility/utility_gui_lib.h"
#include "../executor/executor.h"
#include "about.h"
#include <form.h>

//Точка входа в цикл событий окна параметров
void settings();
#endif
```

Файл parser.c

```
#include "parser.h"

void process_input(const char input[256]) {
    char ptr[PATH_MAX];
    strcpy(ptr, input);
    int i = 0;
    while (ptr[i] != '\0') {
```

```
switch (ptr[i]) {
    case '!':
        strcat(current_operation, "-not ");
        break;
    case '&':
        size_flag = 0;
        if (first_flag) {
            if (braces_flag)
                strcat(output, "\\) ");
            strcpy(current_operation, "-and ");
        }
        break;
    case '|':
        size_flag = 0;
        if (first_flag) {
            if (braces_flag)
                strcat(output, "\\) ");
            strcpy(current_operation, "-o ");
        }
        break;
    case '\n':
        return;
    case '{': // Process tags
        {
            j = 0;
            while(ptr[i] != '}') {
                buffer[j] = ptr[i];
                i++;
                j++;
            }

            if (buffer[0] != '\0') {
                buffer[0] = '-';
                strcpy(current_field, buffer);
                if (strcmp(buffer, "-size")) {
                    size_flag = 1;
                }
                braces_flag = 0;
                token[0] = 0;
                memset(buffer, '\0', 128);
            }
        }
        break;
    case '"': // Process name.
        {
            i++;
            j = 1;
            buffer[0] = '"';
            while(ptr[i] != '"') {
                buffer[j] = ptr[i];
                i++;
                j++;
            }
            buffer[j] = ptr[i];
            if (buffer[0] != '\0') {
                first_flag = 1;
                if (!braces_flag) {
                    strcat(output, current_operation);
                    strcpy(current_operation, " ");
                }
                if (token[0] != 0 && !size_flag) {
                    strcat(output, "-o ");
```

```
                        } else if (token[0] != 0 && size_flag) {
                            strcat(output, "-and ");
                        } else
                            strcat(output, "\\( ");
                        sprintf(token, "%s %s ", current_field, buffer);
                        strcat(output, token);
                        braces_flag = 1;
                        memset(buffer, '\0', 128);
                    }
                }
                break;
            default:
                break;
        }

        i++;
    };
}

int main(int argc, char **argv) {
    if (argc != 2) {
        printf("Usage: %s <input_string>\n", argv[0]);
        return 1;
    }

    process_input(argv[1]);
    if (braces_flag)
        strcat(output, "\\) ");
    printf("%s", output);

    return 0;
}
```

Файл parser.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <linux/limits.h>

char current_field[128];
char token[PATH_MAX];
char output[PATH_MAX];
char braces_flag = 0;
char size_flag = 0;
char first_flag = 0;
char current_operation[20];
char buffer[128];
int j = 0;

//Парсер
void process_input(const char input[256]);
```

Файл utility.h

```
#ifndef UTILITY_H
#define UTILITY_H

#define ARRAY_SIZE(a) (sizeof(a) / sizeof(a[0]))

#endif
```

Файл utility_gui_lib.c

```
#include "utility_gui_lib.h"

int ch;
bool exit_flag = FALSE;
screen_size scr_size;

static const toolbar TOOLBAR_NAMES_AND_KEYS[] = {
    {EXECUTE_GUI,"Enter"},
    {ADDITIONAL_COMMAND_GUI,"F1"},
    {ABOUT_GUI,"F2"},
    {EXIT_GUI,"F3"},
};

void default_key_handler(const key_handler *restrict control_key_handlers,
size_t size) {
    for (size_t i = 0; i < size;i++) {
        if (control_key_handlers[i].key == ch) {
            control_key_handlers[i].handler();
            ch = -1;
            break;
        }
    }
}
void render_key_map() {
    static WINDOW *win = NULL;
    if (win == NULL) {
        win = newwin(1, scr_size.max_x, scr_size.max_y - 1, 0);
        wbkgd(win, COLOR_PAIR(1));
        wrefresh(win);
    }

    wclear(win);
    mvwin(win, scr_size.max_y - 1, 0);
    wrefresh(win);

    for (size_t i = 0; i < ARRAY_SIZE(TOOLBAR_NAMES_AND_KEYS);i++) {
        wattron(win, COLOR_PAIR(2));
        mvwprintw(win, 0, i * (scr_size.max_x /
ARRAY_SIZE(TOOLBAR_NAMES_AND_KEYS)), "%s",
TOOLBAR_NAMES_AND_KEYS[i].key_name);
        wattron(win, COLOR_PAIR(1));
        mvwprintw(win, 0, i * (scr_size.max_x /
ARRAY_SIZE(TOOLBAR_NAMES_AND_KEYS)) +
strlen(TOOLBAR_NAMES_AND_KEYS[i].key_name), " %s",
TOOLBAR_NAMES_AND_KEYS[i].name);
    }
    wrefresh(win);
}

char *trimwhitespace(char* str) {
    char *end;
    while (isspace((unsigned char)*str)) str++;
    if (*str == 0)
        return str;

    end = str + strlen(str) - 1;
    while (end > str && isspace((unsigned char)*end)) end--;
    end[1] = '\0';
    return str;
}
```

Файл utility_gui_lib.h

```
#ifndef UTILITY_GUI_LIB_H
#define UTILITY_GUI_LIB_H
#ifndef _DEFAULT_SOURCE
#define _DEFAULT_SOURCE
#endif
#include <ncurses.h>
#include <string.h>
#include <linux/limits.h>
#include <ctype.h>
#include "../config.h"
#include "utility.h"
#define ARRAY_SIZE(a) (sizeof(a) / sizeof(a[0]))
#define ENTER_KEY 10

typedef int control_key;

typedef struct screen_size {
    int max_x;
    int max_y;
}screen_size;

typedef struct toolbar {
    const char *name;
    const char *key_name;
}toolbar;

typedef char buf_t[PATH_MAX];
typedef void (*render_routes)();
typedef render_routes event_handler;
typedef struct key_handler {
    control_key key;
    event_handler handler;
}key_handler;

//Основной обработчки событий клавиатуры
void default_key_handler(const key_handler* control_key_handlers, size_t
size);

//Функция удаления пробельных символов из строки
char *trimwhitespace(char* str);

//Функция отрисовки подсказки с клавишами внизу окна
void render_key_map();
#endif
```

Файл config.h

```
#ifndef CONFIG_H
#define CONFIG_H

#define TITLE "find CURSES"
#define VERSION "v1.0"
#define DESCRIPTION "This program is a simple shell over the find utility."
#define DESCRIPTION1 "DOCUMENTATION"
#define DESCRIPTION2 "To record a parameter, the parameter is entered in
quotation marks (\"...\") \nin the field opposite the name"
#define DESCRIPTION3 "To use the parameter in the search, enter the name of
the flag in curly brackets ({...}) \nin the field opposite the name of the
\"Request format\""
#define DESCRIPTION4 "How to make \"Request format\":"
```

```
#define DESCRIPTION5 "Name format - {name}. Example: \"main.c\" or \".c\"."
#define DESCRIPTION6 "Group - {group}. Example: \"users\"."
#define DESCRIPTION7 "User - {user}. Example: \"amor\"."
#define DESCRIPTION8 "Regular expression - {regex}. Example: \".*\\.c\"."
#define DESCRIPTION9 "Access - {perm}. Example: \"644\"."
#define DESCRIPTION10 "Size - {size}. Examples: \"+1k\" or \"-1M\"."
#define DESCRIPTION11 "b -> 512-byte blocks (default), c -> bytes, w -> two-
byte words, k -> kilobytes, M -> megabytes, G -> gigabytes."
#define DESCRIPTION12 "Request format -> example: !{name} | {size} & {user}."
#define DESCRIPTION13 "\"!\' -> NOT, \"|\" -> OR, \"&\" -> AND."
#define FILE_TYPE_GUI "Display files"
#define DIR_TYPE_GUI "Display catalogs"
#define SYMLINK_TYPE_GUI "Display symbolic links"
#define EXECUTE_GUI "Perform"
#define ADDITIONAL_COMMAND_GUI "Parameters"
#define ABOUT_GUI "About programm"
#define SEARCH_GUI "Search"
#define REGEXP_GUI "Regular expression"
#define NAME_GUI "Name format"
#define GROUP_GUI "Group"
#define PERM_GUI "Access"
#define SIZE_GUI "File size"
#define USER_GUI "User"
#define QUERY_STRING_GUI "Request format"
#define EXIT_GUI "Exit/Back"

#define name_and_version_str(name, version) name " " version
#define PROGRAM_NAME name_and_version_str(TITLE,VERSION)

#define SETTINGS_PATH "./userdata"
#define VISIBLE_MAX 1024
#endif
```

Файл main.c

```
#include "./gui/main_screen.h"

int main() {
  setlocale(LC_CTYPE, "");
  initscr();
  start_color();
  use_default_colors();
  scrollok(stdscr, FALSE);
  cbreak();
  noecho();
  keypad(stdscr, TRUE);
  nodelay(stdscr, TRUE);
  init_pair(1, COLOR_BLACK, COLOR_CYAN);
  init_pair(2, COLOR_WHITE, COLOR_BLACK);
  init_pair(3, COLOR_WHITE, COLOR_BLUE);
  read_settings();
  render_main_window_gui();
  return 0;
}
```