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;

}