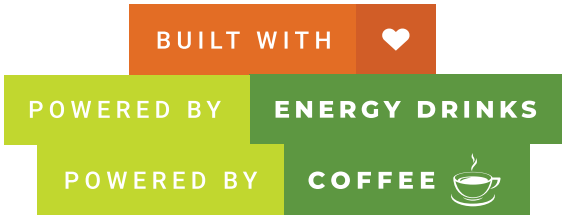
FS Browser. От бессонных ночей до пяти кружек кофе



С чего начать?

Любая работа начинается с какого-то плана (хоть какого-нибудь). Он базируется либо на задаче, либо в какой-либо потребности.

В нашем случае, у нас было конкретное задание реализовать интерактивный файловый менеджер в виде клиент-серверного приложения.

Из них я бы выделил несколько пунктов, с которыми надо было побороться:

- Сам клиент-серверный интерфейс
- Мониторинг изменения в рабочей директории
- Клиентский интерфейс
- Скачивание\Загрузка файла

Клиент-серверный интерфейс

В самом клиент-серверном интерфейсе ничего сложного при реализации не было, так как "уважаемый" динозавр (си) предоставлял в стандартной библиотеке методы для работы с сокетами. Да и слишком много раз мы их писали (писал на си http-server, исходный код: https://github.com/AppLoidx/c-http-server)

Поэтому я не хочу сильно на этом останавливаться, но некоторые ключевые моменты все-таки рассмотрю

Workflow

Основным методом запуска сервера является метод start_server

```
int run_server(unsigned int port, char *path);
```

В нем же происходит инициация сокета и его конфигурация:

```
int socket_desc = socket(AF_INET, SOCK_STREAM, 0);
if (socket_desc == -1) {
    perror("Can't create socket, dude. Sorry :(");
    exit(1);
}

server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = INADDR_ANY;
server_addr.sin_port = htons(port);

int opt = 1;
if (setsockopt(socket_desc, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt)) == -1)
perror("Error while calling setsockopt");

if (bind(socket_desc, (struct sockaddr *) &server_addr, sizeof(server_addr)) <
0) {
    perror("Can't bind socket");
    exit(1);
}

listen(socket_desc, 3);
```

Пропустим все это скучное и посмотрим интересные моменты:

- Создание отдельного треда, который будет смотреть за изменениями в директории

  ```
  pthread_t dir_state_thread;
  pthread_create(&dir_state_thread, NULL, on_dir_change, NULL);
  ```

- Назначение обработчика соединения, при подключении нескольких клиентов

  ```
  pthread_t sniffer_thread;
  pthread_create(&sniffer_thread, NULL, connection_handler, (void *)
  (new_sock));
  ```

- Хранение списка подключенных сокетов в виде связанного списка (нужна для нотификации об изменениях в директории)

  ```
  sockets = slist_append(sockets, client_sock);
  ```

Мониторинг изменения в директории

Это являлось довольно интересной задачей, которая была решена с помощью `<sys/inotify>`

Вот пример кода с inotify:

```
#include <sys/inotify.h>
```

```
wd = inotify_add_watch(fd, argv[1], IN_CREATE | IN_MODIFY | IN_DELETE);

if (wd == -1)
{
    printf("Не удалось добавить монитор %s\n",argv[1]);
}
else
{
    printf("Смотрим за %s\n",argv[1]);
}

/* do it forever*/
while(1)
{
    i = 0;
    length = read( fd, buffer, BUF_LEN );

    if ( length < 0 ) {
        perror( "read" );
    }

    while ( i < length ) {
        struct inotify_event *event = ( struct inotify_event * ) &buffer[ i ];
        if ( event->len ) {
            if ( event->mask & IN_CREATE) {
                if (event->mask & IN_ISDIR)
                    printf( "Директория %s была создана.\n", event->name );

                else
                    printf( "Файл %s был создан, WD %d\n", event->name, event-
>wd );
            }

            if ( event->mask & IN_MODIFY) {
                if (event->mask & IN_ISDIR)
                    printf( "Директория %s была изменена.\n", event->name );

                else
                    printf( "Файл %s был изменен, WD %d\n", event->name, event-
>wd );
            }

            if ( event->mask & IN_DELETE) {
                if (event->mask & IN_ISDIR)
                    printf( "Директория %s была удалена.\n", event->name );

                else
                    printf( "Файл %s был удален, WD %d\n", event->name, event-
>wd );
            }

            i += EVENT_SIZE + event->len;
        }
    }
}
```

При реализации выглядит это следующим образом:

```
void *on_dir_change(void *socket_desc) {
    int length, i = 0, wd;
    int fd;
    char buffer[BUF_LEN];

    char *working_dir = malloc(255 * sizeof(char));
    getwd(working_dir);


    /* Initialize Inotify*/
    fd = inotify_init();
    if ( fd < 0 ) {
    perror( "Couldn't initialize inotify");
    }

    /* add watch to starting directory */
    wd = inotify_add_watch(fd, working_dir, IN_CREATE | IN_DELETE);

    if (wd == -1)
    {
        printf("Couldn't add watch to %s\n", working_dir);
    }
    else
    {
        printf("Watching:: %s\n", working_dir);
    }

    while(1)
    {
        i = 0;
        length = read( fd, buffer, BUF_LEN );

        if ( length < 0 ) {
        perror( "read" );
        }

        while ( i < length ) {
        struct inotify_event *event = ( struct inotify_event * ) &buffer[ i ];
        if ( event->len ) {
            if ( event->mask & IN_CREATE) {
                slist_observer(sockets, send_notification);
            }
            if ( event->mask & IN_DELETE) {
                slist_observer(sockets, send_notification);
            }
            i += EVENT_SIZE + event->len;
        }
        }
    }
    /* Clean up*/
    inotify_rm_watch( fd, wd );
    close( fd );
```

```
    return 0;
}
```

Скачивание/Загрузка файлов

Скачивание (взгляд со стороны сервера)

```
do {
    int c = remain_data > BUFSIZE ? BUFSIZE : remain_data;
    char *data = malloc(c);

    pread(fd, data, c, offset);
    offset += c;

    write(sock, data, c);
    free(data);
    remain_data -= c;

} while (remain_data > 0);
```

Загрузка

```
while (remain_data > 0) {
    int len = recv(sock, file_response, BUFSIZE, 0);
    fwrite(file_response, len, 1, fp);
    remain_data -= len;
    memset(file_response, 0, BUFSIZE);
}
```

На клиенте и сервере методы загрузки\скачивания довольно похожи.

Клиентский интерфейс

Для начала добавил просто управляющие привязки клавиш

```
#define BACK_TO_HOME_LETTER 'H'
#define BACK_TO_HOME_LETTER_SPACED " H "

#define DOWNLOAD_LETTER 'D'
#define DOWNLOAD_LETTER_SPACED " D "

#define UPLOAD_LETTER 'U'
#define UPLOAD_LETTER_SPACED " U "

#define EXIT_LETTER 'Q'
#define EXIT_LETTER_SPACED " Q "
```

```
#define OPEN_LETTER 'O'
#define OPEN_LETTER_SPACED " O "

#define REFRESH_LETTER 'R'
#define REFRESH_LETTER_SPACED " R "

#define EXIT_VIEW_MODE_LETTER 'C'
#define EXIT_VIEW_MODE_LETTER_SPACED "C"

#define HELP_LETTER 'H'
#define HELP_LETTER_SPACED " H "
```

И обрабатываем их

```
if (c == KEY_UP) {
    if (pointer != 0) {
        cursor = 2;
        pointer--;
        up_down_event();
    }
}
if (c == KEY_DOWN) {
    if (pointer != files_count - 1) {
        cursor = 2;
        pointer++;
        up_down_event();
    }
}
if (c == 10) {
    old_pointer = pointer;
    File file = find_file_at(files, pointer)->file;
    on_select(file.name, strcmp(file.type, "DIR") != 0);
}
if (c == REFRESH_LETTER) {
    on_select(NULL, false);
}
if (c == HELP_LETTER) {
    on_select("", false);
}
if (c == DOWNLOAD_LETTER) {
    File file = list_at(files, pointer)->file;
    if (strcmp(file.type, "DIR") != 0) on_download(file.name);
}
```

Сам же интерфейс пишем в обычно ncurses стиле

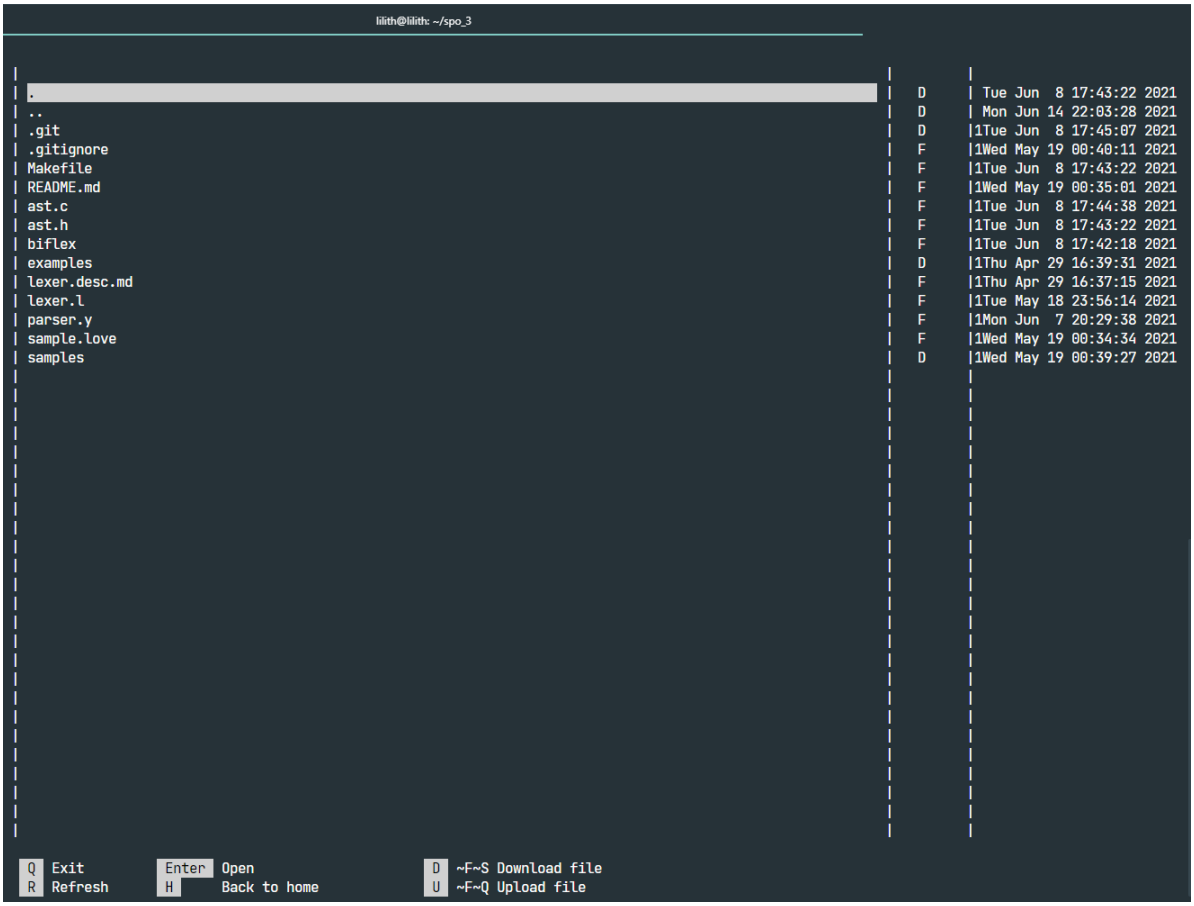
```
wattron(window_help, SELECTED);
wmove(window_help, 0, 1);
wprintw(window_help, EXIT_VIEW_MODE_LETTER_SPACED);

wmove(window_help, 1, 1);
```

```
wprintw(window_help, EXIT_LETTER_SPACED);

wattroff(window_help, SELECTED);

wmove(window_help, 0, 5);
wprintw(window_help, "Close file");
wmove(window_help, 1, 5);
wprintw(window_help, "Exit");
wmove(window_help, 0, 26);
wprintw(window_help, "Prev line");
wmove(window_help, 1, 26);
wprintw(window_help, "Next line");
```

Вывод файликов

```
wmove(window_main, cursor, 2);
wprintw(window_main, "%s", file->name);
if (strcmp(file->type, "FILE") == 0) {
    wmove(window_size, cursor, 3);
    wprintw(window_size, "%s", "F");
} else {
    wmove(window_size, cursor, 3);
    wprintw(window_size, "%s", "D");
}

wmove(window_date, cursor, 1);
wprintw(window_date, "%s", file->date);
```

Интересно, что при написании всего этого я вспомнил про Java Swing (не знаю почему, но напомнило про это)



Согласитесь выглядит красиво 😍

GITHUB