APO-LAU

Apo lau is semestral project for APO on CTU. LAU stands for Light admin unit. Goal of semestral project was to create C/CPP program which handles lights in individual rooms, communicates with other devices and displays data to display.

Usage

- download this project
- `make`
- `./app conf_files/unit1.txt`

Info

Project is split up into individual files. All files are written in C. Pogram runs in 4 threads and everything in code is commented. You will find following files in the project: - app - file with main, loads configuration, runs threads - devices list - C alternative to vector of pair of socket address : light admin unit, used for saving all currently broadcasting unit and info about them - light_admin_unit - light admin unit struct and related methods - passer - contains only one struct, which is used to pass arguments to all 4 threads - pixel - pixel struct and related methods - mzapo_* - files provided by our teacher, methods and constants to access all peripherals - font_* - files provided by our teacher, methods and structs with fonts - socket_rocket - all stuff necessary for UDP LAN communication with other lau's, handles update sending and update receiving, dynamically creates list of all connected units - console_info - displays everything that's happening into the console, mainly for debugging or if the unit doesn't have a display - display - handles everything related to peripherals, from displaying data to reading input from knobs

Download

```
git clone https://github.com/matoous/apo_lau

cd apo_lau/
```

Compile

```
make
```

Apo-lau uses gcc for its compilation, feel free to change makefile as much as you want, since I am not sure about its quality. Do not forget to keep/add threads flag.

Run

```
./app configuration_file
```

Configuration file is used to load initial data of the unit. Configuration file is structured as follows:

`unit name` // unit name on separate line, the first line, required

`ceiling color r g b` // 3 integer values => 0 and <= 256 defining the color of ceiling light, required

`walls color r g b` // 3 integer values => 0 and <= 256 defining the color of walls light, required

`icon 256*uint16_t` // 256 uint16_t values defining the 16x16 icon of the unit in rgb565 format, required

To safely stop the application all you need is to press any button, all the threads one by one will get killed and the application will end.

What happens when I run the app

- configuration file gets loaded and local light admin unit is set
- list of devicis is initialized
- first the udp_listener thread gets started, this thread listens to any incoming traffic and updates devices list accordingly
- second the udp_updater threads gets started, this thread sends updates about local light admin unit and then sleeps for 1 sec
- third the parlcd_displayer thread gets started, this thread handles all peripherals, inputs and ouputs to display. This thread fails silently and if it fails, it keeps the device running so you can set its parameters remotely
- fourth the console_info thread gets started, this thread is not required to run, its main purpouse is to serve the developer during debuging
- application waits for key input
- all threads join the main and end
- all things get freed
- application end