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

File and methods

pixel.c

pixel_t pixel (const uint16_t color)

```
constructs pixel from uint16_t color
@param color | uint16_t color
@return pixel
```

mzapo_parlcd.c

socket_rocket.c

void _int_tbetb (int x, char buffer, int offset)*

```
Puts int into the buffer (converts it to Big-Endian)
@param x - number
@param buffer - buffer to populate
@param offset - offset in buffer
```

void _uint32_t_tbetb (uint32_t x, char buffer, int offset)*

```
Puts uint32_t into the buffer (converts it to Big-Endian)
@param x - number
@param buffer - buffer to populate
@param offset - offset
```

void _int16_t_tbetb (int16_t x, char buffer, int offset)*

```
Puts int16_t into the buffer (converts it to Big-Endian)
@param x - number
@param buffer - buffer to populate
@param offset - offset
```

void _color_tbetb (pixel_t c, char buffer, int offset)*

```
Puts color into the buffer
@param c - color
@param buffer - buffer to populate
@param offset - offset in buffer
```

void _name_tbetb (const char c[], char buffer, int offset)*

```
Puts light admin unit name into the buffer
@param c - light admin unit name
@param buffer - buffer to populate
@param offset - offset in buffer
```

void _icon_tbetb (const uint16_t icon[], char buffer, int offset)*

```
Puts light admin unit icon into the buffer on given offset, converts to Big-Endian
@param icon - icon array
@param buffer - buffer to populate
@param offset - offset in buffer
```

uint32_t _bt_uint32_t (char buf, int offset)*

```
Gets uint32_t from buffer
@param buf - buffer
@param offset - offset in buffer
```

```
@return uint32_t
```

pixel_t _bt_color (char buf, int offset)*

```
Gets color from buffer (should be saved as uint32_t, first byte empty, then red,
green, blue)
@param buf - buffer
@param offset - color offset in buffer
@return pixel_t color
```

void _bt_name (char buf, int offset, char* uname)*

```
Gets light admin unit name from buffer
@param buf - buffer
@param offset - name offset in buffer
@param uname - array to populate
```

uint16_t _bt_uint16_t (char buf, int offset)*

```
Gets uint16_t from buffer
@param buf - buffer
@param offset - offset in buffer
@return uint16_t
```

int16_t _bt_int16_t (char buf, int offset)*

```
Gets uint16_t from buffer
@param buf - buffer
@param offset - offset in buffer
@return uint16_t
```

void _bt_icon (char buf, int offset, uint16_t icon[256])*

```
Gets light admin unit icon from buffer
@param buf - buffer
@param offset - icon offset in buffer
@param icon - array to populate
```

void _sr_modify_lau (lau_t lu, char* buf, pthread_mutex_t* local_lau_mutex)*

```
Updates light admin unit
@param lu - light admin unit
@param buf - buffer
```

void _sr_set_lau (lau_t lu, char* buf, pthread_mutex_t* local_lau_mutex)*

```
Sets light admin unit to received status
@param lu - light unit
@param buf - buffer
```

void sr_updater (void args)

```
Runs as threads
Sends update about local unit every one second
@param passer struct with all necessary arguments
```

void sr_init (void args)

```
Handles incoming messages, updates, etc.
@param passer struct with all necessary arguments
```

void send_modify (

```
Send update packet
@param sockfd | socket file descriptor
@param out_addr | address to send at
@param cr | change in ceiling red
@param cg | change in ceiling green
@param cb | change in ceiling blue
@param wr | change in walls red
@param wg | change in walls green
@param wb | change in walls blue
```

void send_set (

```
Send set packet
@param sockfd | socket file descriptor
@param out_addr | address to send at
@param cr | ceiling red
@param cg | ceiling green
@param cb | ceiling blue
@param wr | walls red
@param wg | walls green
@param wb | walls blue
```

devices_list.c

void dl_init (devices_list_t devices_list)*

```
init device list in provided pointer to devices_list_t
```

```
@param devices_list
```

int dl_push_back (devices_list_t devices_list, sockaddr_in addr, lau_t lau)*

```
Push sock address : device pair on end of the devices list
@param devices_list | devices list
@param addr | sockaddr_in struct with device
@param lau | light admin unit
@return 0 if failed, 1 if OK
```

unsigned int dl_size (devices_list_t devices_list)*

```
Return count of devices in list as unsigned integer
@param devices_list | list of devices
@return count of devices as unsigned int
```

void dl_destroy (devices_list_t devices_list)*

```
Destroy the devices list struct
@param devices_list | pointer to devices list
```

void dl_delete (devices_list_t devices_list, unsigned int index)*

```
Delete device from list
@param devices_list | list of devices
@param index | index of device to be deleted
```

console_info.c

void console_info (void args)

```
Runs in separate thread, show updates about current application state
@return NULL
```

display.c

void put_char_there (char c, int row, int column, uint16_t color, uint16_t background)

```
Puts char on specific place on display
@param c | char
@param row | 0 <= row <= 20
@param column | 0 <= column <= 59
@param color
@param background
```

void put_string_on_line (char c, int row, int offset, uint16_t color, uint16_t background_color)*

```
Draws string on line
@param c | string
@param row | 0 <= row <= 19
@param offset | 0 <= offset in row <= 60
@param color | font color
@param background_color | background color
```

void redraw (unsigned char parlcd_mem_base)*

```
Redraw the display
@param parlcd_mem_base
```

void one_device_draw_init ()

```
Inits static elements on display
```

void one_device_draw (lau_t lu, int selected_row, unsigned char parlcd_mem_base)*

```
Draw given light admin unit info to display
@param lu | light unit o be displayd
@param selected_row | selected row (go back, colors, etc.)
@param parlcd_mem_base | memory base for parlcd
```

void all_devices_draw_init ()

```
Init all devices list
nulls the display
```

void all_devices_draw (devices_list_t devices_list, int curr_device_in_list, unsigned char* parlcd_mem_base)*

```
Draw devices list on display
@param devices | vector of all connected devices
@param knob_change | change of knob of currently selected device
```

void read_knobs (uint8_t k1, uint8_t* k2, uint8_t* k3, uint8_t* b1, uint8_t* b2, uint8_t* b3, unsigned char* knobs_mem_base)*

```
Reads knobs to provided variables
@param k1 | knob1
```

```
@param k2 | knob2
@param k3 | knob3
@param b1 | button1
@param b2 | button2
@param b3 | button3
@param knobs_mem_base | memory base for knobs and buttons
```

void par_lcder (void args)

```
Handles display and hardware inputs
@param passer with all necessary data
```

font_rom8x16.c

mzapo_phys.c

app.c

void _start_threads (passer_t passer)*

```
starts all threads
@param passer
```

void _stop_threads ()

```
stops all threads
```

int _load_lau (const char file_name, lau_t* lu)*

```
loads configuration of light admin unit
@param file_name | configuration file name
@param lu | light admin unit
@return 1 on error 0 on success
```

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

```
main thread, runs everything, needs 1 argument: configuration file as specified in
manual
@param argc
@param argv
@return
```