Exercise F: Graphical User Interfaces

We've got a chat program that works in the terminal. Now it's time to go from the 80's to the 90's: Let's build a graphical user interface for the chat! Btw.: From a GUI programmers point of view not much has changed since the 90's… So much for running in circles… er, I mean “innovation”. Anyway, this means that no matter if it's a web browser, a mobile application, a classical GUI or The Next Big Thing™, they all do more or less the same as we'll do. Usually just buried under an ever changing set of different buzz words.

We'll use the IUP library to build a GUI in C. You can find the documentation here: http://webserver2.tecgraf.puc-rio.br/iup/. Feel free to look up any functions or tutorials if you need to.

1. Hello GUI

To get a grasp on GUIs we'll first compile a little program that opens a window with a label in it.

- T Download gui.a and gui.h from the personal schedule.
- T Create the file hello_gui.c in the same directory with the following code in it:

```
#include <stdio.h>
#include "gui.h"

int main() {
    IupOpen(NULL, NULL);

    Ihandle* label = IupLabel("Hello GUI!");
    Ihandle* dialog = IupDialog(label);

    IupShow(dialog);
    IupMainLoop();

    IupClose();
    return 0;
}
```

- T Compile the program with the command:

```
gcc -Wall -std=c99 hello_gui.c -o hello_gui gui.a $(ar -p gui.a LDLIBS)
```

- T Run the program. Either from the terminal in the usual way or by double clicking it in the file browser.

Operating Systems

2. Creating the input controls

Ok, we have a basic GUI program. We'll transform it step by step into a GUI for our previous network chat program. First step: Create text fields and buttons for the information the user has to enter. The address and port and if the user wants to start a server or client. Previously we used command line arguments to get that information. Now we let the user enter them into our GUI.

- Create a new file named gui_chat.c with the same code in it as hello_gui.c.
- Replace the label with an IupHbox that contains two text fields and two buttons. The result should look like that:

```
+-----------------------------------------------+
|                     Dialog                    |
+---------+--------+--------------+--------------+
| address |  port  | Start Client | Start Server |
| (text)  | (text) |   (button)   |   (button)   |
+---------+--------+--------------+--------------+
```

  One text field to enter the address, another one to enter the port. The first button to start the client, the second to start the server. For now just build the GUI. We'll add the network stuff later.

  Remember: The last argument to the IupHbox function has to be NULL so it knows the list is finished.

- Compile and run the program to check if it works.
- Make the address text field expand when you resize the window. Do that by using IupSetAttribute() to set the "EXPAND" attribute to "HORIZONTAL".
- Optional: Set the "CUEBANNER" attribute of the address and port text controls so you know what to enter. The cuebanner is a gray help text that is shown when the control is empty. Similar to the placeholder attribute of HTML input elements.

3. It's good to be called back

We have controls to enter data. On to the next step: Creating functions that are executed when the buttons are pressed (so called “callbacks” because the GUI calls these functions for us when needed). For now we'll just have them print ToDo messages. Next week we'll fill them with the proper stuff.

- Add this function to your program:

```
int client_start_cb(Ihandle* self) {
    printf("TODO: start client\n");
    return IUP_DEFAULT;
}
```

- Use IupSetCallback() to set the “ACTION” callback of the “start client” button to client_start_cb.
- Create a function named server_start_cb similar to client_start_cb() above. This function should print the message “TODO: start server\n”. Then add that function as the ACTION callback of the “start server” button.
- Test the program: Check if the messages appear in the terminal when you press the buttons.

4. Adding controls for chat history and new messages

Basic GUI, check, functions that are called when buttons are pressed, check, a place to display old messages and a text field to type new ones, dang. Ok, then that's up next.

- Create a multiline text field for the chat history and a text field and button to enter a message and send it. At the end of this exercise it should look like this:

```
+-----------------------------------------------+
|                     Dialog                    |
+---------+--------+--------------+--------------+
| address |  port  | Start Client | Start Server |
| (text)  | (text) |   (button)   |   (button)   |
+---------+--------+--------------+--------------+
|                                               |
|                                               |
|                 chat history                  |
|                                               |
|                                               |
+-------------------------------+---------------+
| new message (text)            | send (button) |
+-------------------------------+---------------+
```

  For now just create the elements and put them into proper boxes. You'll need an IupHbox for the last row and an IupVbox to stack the three rows.

- Set the "MULTILINE", “EXPAND”, "READONLY" and "WORDWRAP" attributes of the chat history text field all to "YES". This will change the text field from a single line into multiple lines and make it use up all available space.
- Make the new message text field take up all the space of the last row it can get. Same as the address text control the first row (Tip: The "EXPAND" attribute again).
- Set the “MINSIZE” attribute of the dialog to “300x250”. This defines the minimum size of the window in pixels.
- Again, test the program: Resize the window to see if the chat history and new message text fields properly take up the new space.

5. Adding a message to the history

All the GUI elements are in place. Now for a bit of interaction. When the user presses the send button we'll take the text in the message text field and append it to the chat history. The actual sending of the message will have to wait until next week. Today we'll just do the GUI part of it.

- Add an “ACTION” callback for the send button similar to the callbacks for the start client and start server buttons. Name this callback “send_message_cb”.
- When the send button is pressed get the current value of the message text control. Use IupGetAttribute() to get the "VALUE" attribute of the message text field. Store it in a string variable.

  Tip: You'll need to use a global variable to store the Ihandle* of the message text field. Once the variable is global you can use it in your callback.
- Append the message string to the history text field in the format "me: %s" where the %s should be replaced with the message string. Use IupSetStrf() on the histories "APPEND" attribute for that. IupSetStrf() is similar to printf() in that it replaces placeholders like “%s” with the arguments before setting the attribute.

  Tip: You'll need to add another global variable for the history text field. Otherwise we can't access that variable from our callback function.
- Use IupSetAttribute(history_text, "SCROLLTO", "1000000000,1000000"); to scroll the history text field down to the latest message. Of course you have to replace “history_text” with your the variable of the chat history text field.
- Clear out the text in the message text control so we can type the next message (set the "VALUE" attribute to "").
- Optional: Ignore empty messages.

6. Optional: “send” a message when return is presses

This is a common feature for chat programs: When you type a message and press return the message is send instantly. No need to press the “send” button. Let's add this to our chat as well.

- Add a callback for "K_ANY" (any key press) to the message text field.

  Look up the signature in the documentation and define a matching callback function (Tip: Open the IUP documentation, expand the entire navigation and use the browser search via ctrl+f). When setting the callback you need to cast it to (Icallback) because of the K_ANY callbacks extra parameter.

- When the callback get the character K_CR directly call the send_message_cb() function your self. K_CR is a constant defined by IUP representing the return key. Also return IUP_IGNORE to prevent any default actions the return key would trigger.

- In all other cases just return IUP_CONTINUE (just let the GUI do it's job).