Systems programming

Week 2 – Lab 3

Client-Server and ncurses

In this laboratory students will implement a simple distributed system, where the clients controls moving characters that show on the server screen.

Clients will send movement orders to the server and the server updates the position of the various characters on a windows.

In order to represent symbols in the screen the ncurses library will be used.

1 Ncurses

Ncurses is a library that allow the the reaction of text bases user interfaces. Ncurses provides an API to read from the keyboard and write with various degrees of control in the screen.

1.1 Ncurses installation

To use this library it is necessary to install it on Linux or mac OS X

1.1.1 Linux

The ncurses installation in Linux is straightforward and depend on the Linux version.

Fir Ubuntu it is necessary to issue the following commands

```
sudo apt-get install libncurses5-dev libncursesw5-dev
```

for other distributions the packages to install are different, as explained in the following page:

https://www.cyberciti.biz/faq/linux-install-ncurses-library-headers-on-debian-ubuntu-centos-fedora/

1.1.2 MAC OS X

To install the ncurses library in MAC OS X it is necessary to use the brew package manager for MAC OS X:

https://brew.sh/

after installing brew it is necessary to type the following command in the terminal:

```
brew install ncurses
```

1.2 Ncurses tutorial and documentation

The following site presents a detailed documentation of the various functions provided by ncurses:

https://invisible-island.net/ncurses/ncurses.html

https://invisible-island.net/ncurses/ncurses-intro.html

https://tldp.org/HOWTO/NCURSES-Programming-HOWTO/

1.3 Examples

The provides example files demonstrate the use of the fundamental ncurses functions.

To compile each example and all other programs that use ncurses it is necessary to use the **-lncurses** argument in the gcc

ncurses-example-1.c This program reads the key pressed by the user. If the user presses one of the key arrows it prints the direction. If any other key was presses it prints the corresponding character .	

`Initscr();`
`cbreak();`
`keypad(stdscr, TRUE);`
`noecho();`
	Initialization of the **ncurses** library
	From this point forward `printfs` / `fgets` stop working and all interaction with the screen/keyboard is done using the **ncurses** functions

`ch = getch();`	Get the pressed key. If the user keeps the key presses this function does not block and returns immediately
`switch (ch) {`	If the user presses a regular letter, digit or

`case KEY_LEFT:`
`break;`
`case KEY_RIGHT:`
`break;`
`case KEY_DOWN:`
`break;`
`case KEY_UP:`
`break;`
`}`
	symbol, the `ch` variable contains the corresponding ASCII code.
	In the case of special keys it is necessary to compare the variable with use specific constants:
	• `KEY_LEFT, KEY_RIGHT`
	• `KEY_UP, KEY_DOWN`

`mvprintw(0,0,`
`"%d :%c key was pressed",`
`n, ch);`
	The `mvprintw` function receives a string and further arguments like the `printf`, but allows to write such string in a specif position of the screen.
	The first arguments correspond to the location where this function starts to write the string.

ncurses-example-2.c – this program starts by drawing a windows and then draws a moving **x** on such windows.	

`initscr();`
`cbreak();`
`keypad(stdscr, TRUE);`
`noecho();`
	Initialization of the **ncurses** library
	From this point forward `printfs` / `fgets` stop working and all interaction with the screen/keyboard is done using the **ncurses** functions

`WINDOW * my_win = newwin(` `WINDOW_SIZE, WINDOW_SIZE,`	Creates and draws a windows starting at position 0, 0 and with dimension of

`0, 0);`
`box(my_win, 0 , 0);`
`wrefresh(my_win);`
	WINDOW_SIZE x WINDOW_SIZE
	
	The `my_win` variable will be used in other functions to write into this window These functions use relative (inside the window) coordinates

`wmove(my_win, pos_y, pos_x);` `waddch(my_win,ch| A_BOLD);`	These functions write one character on a specific position inside the `my_win` window. The `wmove` places the cursor at a specific position inside the `my_win` window. The `waddch` writes the character `ch` inside `my_win` at the previously defined cursor position
`wrefresh(my_win);`	After writing on a windows, it is necessary to call `wrefresh` to update the screen and show all the changes.

2 Client-server character remote control

In these exercise students will implement a system composed of a server and multiple clients that communicate using FIFOS.

The server will be similar to **ncurses-example-2.c** but will show on the screen various characters controlled by other programs. The movement of these characters will be received through a FIFO.

Some characters will be controlled by the user (using a program similar to **ncurses-example-1.c**), other characters will be controlled by a random movement generator.

The skeleton files for various program are available on the **exercise-1** directory:

- server.c

- human-control-client.c
- machine-control-client.c

When the server starts the server, he will wait for clients to start sending messages trough a FIFO.

Every client that wants to interact with the server needs to send a starting message that defines what character sch clinet will control, from this point forward the client will send messages stating what direction the character will move (left, right, up, down).

The server will receive each movement message and update the corresponding character on the screen.

3 Exercise 1

This version of the system will only allow one client (either human or machine) to run at the same time.

Students should follow the next steps to implement the exercise.

The various places to write code are identified in the supplied files.

3.1 TODO 1 (remote-char.h)

The first step is the definition of the messages that will be exchanged between the client and the server.

This should be done on **remote-char.h** file.

Take attention that there are two types of messages

- **connection** - the one sent when first connecting
- **movement** - the one that sends the movements.

Both types of messages should send the character selected by the user and the movement messages should contain the direction (UP, DOWN, LEFT, RIGHT).

3.2 TODO 2 (remote-char.h)

Since the clients and server will interact using a FIFO it is necessary to define such name, and make it available to all programs.

Define a constant with the name of the FIFO in the remote-char.h file. The the FIFO should be created in the /tmp directory.

3.3 TODO 3 (server.c)

The server should create and open a FIFO for reading

implement the code to create the FIFO (if it does not exist) and open it for reading.

3.4 TODO 4 (human-control-client.c)

The human-control-client should create and open a FIFO for writing.

implement the code to create the FIFO (if it does not exist) and open it for reading.

After implementing TODO_4 it is now possible to run the server and this client on different windows and understand if they are opening the same FIFO. If this happens we can now start to send data….

3.5 TODO 5 (human-control-client.c)

After opening the FIFO for writing the human-control-client.c should read from the keyboard what character the user wants to control and send such information to the server.

implement the code to read a character form the keyboard.

This should be done before initializing ncurses otherwise we need to use ncurses keyboard reading functions.

3.6 TODO 6 (human-control-client.c)

After reading the character the human-control-client.c should send the selected character to the server in a **connection** message.

Declare a variable of the message type (step TODO_1) and initialize with the correct information:

- message type (**connection**),
- selected character.

Write the message into the FIFO.

3.7 TODO 7 (server.c)

Implement on the server the code to read messages from the client.

On the **server.c** read a message from the FIFO.

3.8 TODO 8 (server.c)

Verify if the message read is of the **connection** type and extract from it the character, store it in a variable, define the initial position of the character, and draw it on the screen. The initial character position can be for instance (WINDOW_SIZE/2 , WINDOW_SIZE/2).

To store the character and its positions use the already declared variables:

- ch for the character;
- pos_x and pos_y for the position.

3.9 TODO 9 (human-control-client.c)

In the human-control-client.c, after reading the pressed key, verify if it is an arrow and fill the message to be sent to the server:

- assign the **movement** message type
- assign the direction of the messages

3.10 TODO 10 (human-control-client.c)

On the human-control-client, send the movement message to the server.

3.11 TODO_11 (server.c)

On the server verify if the message received on **TODO_7** is of type **movement**.

If the message is of **movement** type, erase the character from the old position, update the new character position, and redraw it on the screen.

4 Exercise 2

Modify the **machine-control-client.c** file with the correct code for it to connect to the server and control it own character. Follow the sets done in the **human-control-clinet.c** so that the FIFO connection and message transfer is done correctly.

After implementing the three programs try to execute them. You will need to terminate the server (Ctrl-C) to try a different client.

5 Exercise 3

The server implemented in exercise 1 works correctly only when a single client sends messages. If two (or more) clients send simultaneously messages to the server, only one character moves…... It would be nice if multiple characters moved (one for each client).

This happens because the serve only stores information about one character (the ASCII code on the ch variable and one position on the pos_x and pos_y), but receives **movement** messages from the various clients.

To allow the server to accept messages from multiple clients and have multiple characters moving on the screen, it is necessary to change the server so that it stores one character information (ch, pos_x and pos_y) for each client.

Modify the implemented server following the next steps in order to allow multiple characters at the same time.

5.1.1 STEP 1

Declare one datatype that will store each character information: the ASCII code, the x, y.

5.1.2 STEP 2

Declare an array of structures (declares in STEP 1) that will contain the information about the characters of the various clients.

5.1.3 STEP 3

Add one new structure to the array whenever the server receives a message of type **connection** (replace the code of TODO_8)

5.1.4 STEP 4

Whenever the server receives a message of type **movement**, the server should search for the corresponding entry on the array, calculate the new the position of such character and update the screen (replace the code of TODO_11).