In this exercise we study the effect of concurrent processing.

Excercise 2a (Chat program solution using Polling, 0,5p)

We discussed about the polling in the class in the context of simple chat program. Do it now in practice. You can use the function OpenChatFellow[1] that returns a file descriptor that simulates a serial line or network connection from a chat fellow. The simulated fellow writes characters a,b,c, ..., z one at a time in one second intervals.

Your program needs to give immediate answer to both inputs: your own keyboard input and the input from the chat fellow.

Write a program that continuously runs in a loop. It displays always immediately each character whether it comes from your keyboard or from the simulated chat fellow. The program terminates when the user presses Q key.

Hint 1.

You can read from the keyboard using the file descriptor 0 (STDIN_FILENO[2]). You can read from the chat fellow using the file descriptor that has been returned by the function OpenChatFellow(). Function OpenChatFellow() is used in the following way:

prototype is: `int OpenChatFellow();`
it is called this way:

```
int fellow_desc;
fellow_desc = OpenChatFellow();
```

then you can read a character from the fellow this way:

```
char chr_fellow;
read(fellow_desc, &chr_fellow, 1);
```

It is good practice to close the descriptor at the end of the program:

```
close(fellow_desc);
```

You also need to link the object file to your executable. Use the following command:

```
gcc ex2.c InputGenEdu.o -oex2.exe
```

in Edunix. And copy the file InputGenEdu.o to your working directory.

Both descriptors are initially in blocked mode, so that read blocks if the data is not ready.

Hint 2.

You can change the file descriptor from the blocked mode to the unblocked mode in the following way:

```
int file_flags;
file_flags = fcntl(file_descriptor, F_GETFL); // read current file flags
```

[1] OpenCharFellow() function is defined in InputGenEdu.o object file
[2] Defined in <unistd.h>

```
file_flags = file_flags | O_NONBLOCK; // add O_NONOBLOCK bit
fcntl(file_descriptor, F_SETFL, file_flags); // write new flags back
```

After that the read function does not block anymore. The read function call

```
char chr;
result = read(file_descriptor, &chr, 1);
```

always returns and behaves as follows: If the read operation succeeds (data is available), the read function returns 1 (number of bytes that were read). If data is not ready, the function returns -1 (error) and the error number errno value is EAGAIN.

Hint 3.

Keyboard input comes through terminal driver. Terminal driver uses buffering and echoing by default. To get immediate response to the keyboard input, set the terminal to non canonical mode (in unbuffered mode) by entering a command `stty -icanon` from the terminal before you start your program[3].

When you run (test) you program, make sure that the keyboard input appears twice on the screen, because the first output is echo from the terminal driver. If you see the keyboard input only once, it means that your program does not display it at all.

Excercise 2b (Chat program solution using two processes, 0,5p)

The chat application problem can be solved also by dividing the task to two separate processes. Write two programs. The first program is read_kb.c that reads keyboard only (in blocking mode) and displays always immediately all characters to the screen. The second program is read_cf.c that reads chat fellow descriptor only (in blocking mode) and displays always immediately all characters received to the screen. You need to link InputGenEdu.o to this latter program. Test these programs separately first. Then start them both using the following command:

```
./read_cf.exe& ./read_kb.exe
```

Put read_cf.exe to the background. See that these programs together behave in similar way than the program in part a.

Extra, not compulsory additional exercise (Using system calls and library calls, 0,25p)

Write a program that displays the date and time in the format usually used in Finland. That is in the format

dd.mm.yyyy hh:mm:ss

[3] Normally the input driver waits for the CR character to be received until it gives the input to the application (so that you able to edit the line before giving it to system).