B4 - Network Programming

B-NWP-400

Bootstrap

myTeams

0.4

Bootstrap

language: C

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

This bootstrap is an introduction for the myTeams project.
Please note that Jenkins tests are not available for now. download the bootstrap folder from the Intranet.
Note that for the following steps you should not use global or static variables
Handle all the errors gracefully (at least use a perror, preferably without exit).

Understand Select

The `select` function is used to monitor File Descriptors, please read **man 2 select** and take a look at this wikipedia article.

Step1: Read and select

In this step, you should always have the following output for every test.
Please complete the provided `main.c` file with your own code.
You can launch a set of tests with `./tests.sh`. The program will stop if a test fails and display some debugging information (if stdin is closed, stop the program).
Take a look inside the script so that you understand what is tested.
You can also take a look inside `on_command.c` but do not edit it.

```
> (echo -ne "hello\n"; sleep 0.5; echo -ne "world\n") | ./bin
COMMAND: "hello\n\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"
OK: the command seems all right.
COMMAND: "world\n\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"
OK: the command seems all right.
```

Here are some examples of data that could be sent to `on_command`. Please try these and look at the resulting errors.

```
  {
    char cmd[MAX_COMMAND_LENGTH + 1] = "hello\n";
    on_command(cmd);
  }
  {
    char cmd[MAX_COMMAND_LENGTH + 1] = "hello";
    on_command(cmd);
  }
  {
    char cmd[MAX_COMMAND_LENGTH + 1] = "hello\n\n";
    on_command(cmd);
  }
  {
    char cmd[MAX_COMMAND_LENGTH + 1] = "hello\nhel";
    on_command(cmd);
  }
  {
    char cmd[MAX_COMMAND_LENGTH + 1] = "hello\n\0\0hel";
    on_command(cmd);
  }
```

You need to reset every fd_set before calling select.

You may need a temporary buffer to store the data until you receive an entire command

Step2: Write and select

Compile this main function and run the binary with strace.

```
# include <sys/select.h>
# include <stddef.h>

int main(void) {
  fd_set read_fds;
  fd_set write_fds;
  fd_set except_fds;
  do {
    FD_ZERO(&read_fds);
    FD_ZERO(&write_fds);
    FD_ZERO(&except_fds);
    FD_SET(1, &write_fds);
  } while (select(4, &read_fds, &write_fds, &except_fds, NULL) != -1);
}
```

If you need more information about the fds look at **man 2 strace**, and find the flag `--decode-fds=set`.

As you can see something interesting happened !
The select never locks... which is not really nice for our cpu and battery consumption (#GreenCode)
This is because the fd 2 (stdout) is almost always ready to be written to.
Now, try to create a program that reads from `stdin`, and when a line break is detected (or the command buffer is full), write the content of the command (and not the whole buffer) in UPPER case on `stdout`.
Please remember to check if the fd is ready to be read from or written to before calling read or write.

create a datatype that contains:
- an input fd.
- an output fd.
- a buffer for the input.
- a buffer for your output.

Step3: Socket and select

Finally, try to create a server that does the same thing as in step 2 but with sockets.
This server must be able to handle multiple clients at the same time (without fork).
This time if the in_buffer is full, close the connection with the client and display an error on `stderror`.

Allocate all the resources needed by the client at its connection.

Remember about `nc` (man nc)