Operating Systems
Jacobs University Bremen
Dr. Jürgen Schönwälder

Course: CO20-320202
Date: 2019-11-11
Due: 2019-11-18

OS 2019 Problem Sheet #5

Problem 5.1: *word guessing game server* (1+1+2+1+2+1+2 = 10 points)

The word guessing game challenges clients to guess words missing in dynamically created phrases. The game is implemented as a standalone server. After starting up and binding to a port specified on the command line, the server accepts incoming connections from clients and it greets them by sending a generic message (prefixed by `M: `). Afterwards, it creates a word guess challenge and it sends the challenge to the client as a challenge message (prefixed by `C: `). The client then sends a response message (prefixed by `R: ` in order to guess the word. If the client guessed the word correctly, the server sends an OK message (prefixed by `O: `). If the guessed word was wrong, the server sends a fail message (prefixed by `F: `). If the client does not comply to the protocol, the server may send generic messages (prefixed by `M: `). The client can send a quit message (prefixed by `Q: `) to leave the session at anytime.

An example exchange is shown below. The messages send by the client are the messages starting with `R: ` and `Q: `, all other messages are server generated messages.

```
$ nc localhost 1234
M: Guess the missing ____!
M: Send your guess in the form 'R: word\r\n'.
C: Is this ______ happening?
R: really
O: Congratulation - challenge passed!
C: You ___ standing on my toes.
R: are
O: Congratulation - challenge passed!
C: Are you _ turtle?
R: a
O: Congratulation - challenge passed!
C: You'll __ sorry...
R: be
O: Congratulation - challenge passed!
C: You are standing on __ toes.
R: my
O: Congratulation - challenge passed!
C: You should go ____.
R: party
F: Wrong guess - expected: home
C: There __ a fly on your nose.
Q:
M: You mastered 5/6 challenges. Good bye!
```

The server obtains the phrases via the program `fortune`, i.e., by running `fortune -n 32 -s` as a child process and reading the output produced by the child process from a pipe. The server than selects a random word, replaces it with underscores, and sends the challenge to the client.

The server should be able to handle an arbitrary number of clients and it should not block to wait for a `fortune` process to return a result. This means that accepting new clients, reading from connected clients, and reading from the pipes must all be driven from a single main event loop. The server must be able to handle multiple clients without forking a server for each connected client.

The assignment can be broken down into the following steps:

a) Write code to create a listening socket and a main event loop. You may use `libevent` for the main event loop or you may write your own event loop using the `select` system call.

b) Implement a callback to accept new incoming connections.

c) Write code to start `fortune` as a separate process with a pipe to read the output produced by `fortune`.

d) Implement a callback reading phrases produced by `fortune` from the pipe.

e) Write code to select a random word from a phrase and to hide the word using underscores.

f) Write code to send the challenge to the client.

g) Implement a callback to read and process messages received from clients.