Operating Systems Lab
Jacobs University Bremen
Dr. Jürgen Schönwälder

Course: 320232
Date: 2016-11-17
Deadline: 2016-11-23

Problem Sheet #11

Problem 11.1: *guess word game TCP server using libevent* (1+1+2+1+2+1+2 = 10 points)

The word guessing game challenges clients to guess words in dynamically created phrases. The game is implemented as a standalone TCP server. After starting up and binding to a port specified on the command line, the TCP server accepts incoming connections from TCP 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 from a pipe. The server than identifies 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 the libevent mainloop.

The assignment can be broken down into the following steps:

a) Write code to create a listening socket and a main event loop.
b) Implement a callback to accept new incoming connections.
c) Write code to start `fortune` in the background.
d) Implement a callback reading phrases produced by `fortune` from a 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.