Exercises

1. Implement the following server [template: `sockets/ex.is_shell_sv.c`]:

   ```
   is_shell_sv <port>
   ```

   The server creates a socket that listens on the specified port and accepts client requests containing shell commands. (⚠ Each client sends just **one** command to the server.) The server concurrently handles clients, executing each client's command, and passing the results back across the client's socket.

   Some hints:

   - To keep things simple, the server should obtain the client command by doing a single *read()* (not my *readLine()* function!) of a large buffer, on the (imperfect) assumption that that will retrieve the largest command the client might send. A more sophisticated solution would involve the use of *shutdown(fd, SHUT_WR)* (covered later) in the client, and a loop which reads until end-of-file in the server.
   - Easy execution of a shell command:
     `execl("/bin/sh", "sh", "-c", cmd, (char *) NULL);`
   - To have the command send *stdout* (and *stderr*!) to the socket, use *dup2()*.
   - Checking all system calls for errors will save you a lot of grief (really!).
   - Need to write debugging output in the server? Open `/dev/tty`.
   - Even without writing a client (which is a following exercise), you can test the server using *ncat*: `ncat <host> <port-number> <<< "cmd"`

Exercises

Once you have a working server and client, you can make it more robust by checking the following test cases:

1. `while true; do ncat <host> <port> <<< 'false'; done`
   If we create lots of children, is the server reaping the zombies?
2. `ncat <host> <port> <<< 'sleep 1'`
   Does this cause *accept()* in the server to fail with an error?
3. `ncat <host> <port> <<< 'rubbish'`
   Does a suitable error message appear for the client?
4. `ncat <host> <port> <<< 'ls nonexistent-file'`
   Does the error message from *ls* appear for the client?
5. `ncat <host> <port> <<< "echo $(seq 1 1000000 | tr -d '\012')"`
   Does a very long command either get executed correctly or produce a suitable error message from the server?
6. Does your server handle the possibility that *fork()* may fail, by sending a suitable error message back to the client? Test this by modifying the code to replace the call to *fork()* with code that simply yields the value -1.

Note: "<<<" is *bash*-specific syntax meaning take standard input from the following command-line argument.