Exercises

1. Implement the following server and client:
   - `./is_shell_sv server-port`
     The server creates a listening socket that listens on the specified port and accepts client requests containing shell commands. The server concurrently handles clients, executing each client's command, and passing the results back across the client's socket. (The code in `sockets/is_seqnum_sv.c` provides some pieces that are useful for coding the server.)
   - `./is_shell_cl server-host server-port 'shell command'`
     The client connects to the shell server, sends it a single shell command, reads the results sent back across the socket by the server, and displays the results on *stdout*. [template: `sockets/ex.is_shell_cl.c`]

   Some hints:
   - Easy execution of a shell command:
     `execl("/bin/sh", "sh", "-c", cmd, (char *) NULL);`
   - To have a shell 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`.

Exercises

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

1. `./is_shell_cl <host> <port> 'sleep 1'`
   Does this kill *accept()* in the server?
2. `./is_shell_cl <host> <port> 'rubbish'`
   Does a suitable error message appear for client?
3. `./is_shell_cl <host> <port> 'ls nonexistent-file'`
   Does the error message from *ls* appear for client?
4. `./is_shell_cl <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?
5. `while true; do ./is_shell_cl <host> <port> 'false'; done`
   If we create lots of children, is the server reaping the zombies?
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.
7. If your server correctly diagnoses a failed *fork()*, what happens if, having sent its command, the client exits before the server has a chance to send its response? To test this, modify the client so that, if it is supplied with a fourth command-line argument, it misbehaves in this fashion.