Solution to Socket Programming Tasks

Task1

Q: Does it work if we replace "serverName" from "localhost" to "127.0.0.1" in tcp/udp client?

Answer: Yes. “localhost” is the hostname format representation of “127.x.x.x”. All 127.0.0.0/8 addresses are reserved as loopback addresses.

Q: What will happen if we change "serverPort" to a number less than "1024" like "22" in both tcp/udp client and server? Why?

Answer: Nothing will happen as Normally we can use any port number, however port numbers between 0-1023 are reserved as well-know (system) ports to be used by well-known services. In most operating systems though, to bind a server address to such port numbers requires the process to be running with superuser privilege.

Q: Change the messageSize to "1" and see what will happen? Why?

Answer: Modifying the variable “messageSize” to “1” would limit the receive size to “1”. This way, the process can only deal with one byte at a time.

Q: Describe what needs to be changed if we want to implement a web server that handles one request at a time in the TCP code.

Answer: In our demo code, the server provides a service that converts received characters to uppercase. To implement a web server, the server should be able to react to client requests. First, it should parse the request message from the client. Then, the server should check the request for validity and generate an appropriate response. For example, the server receives a “GET / test.txt” message, then the server parses this message and knows that the client is requesting to get the “test.txt” file. If this file is accessible, the server will return it. Otherwise, the server returns the default “404 (page not found)” page.

Task2

Q: Describe the components of a socket address structure. Why do we use the data structure sockaddr_in instead of socketaddr?

Answer: The structure is available in the slides. Both structures can be used to assign a socket

address. However, we should use sockaddr_in instead, as this structure is more user-friendly and less error-prone than socketaddr.

Q: What is the purpose of the bind() function in socket programming? Does the client require it? Why?

Answer: Bind an address to the socket (man 2 bind). The client does not require it. The server should bind to a fixed address (including port number) known to clients to provide services, while the client does not provide services. An available address (including port number) is randomly assigned to the client when connecting to a server.

Q: Explain the purpose of the listen() function in socket programming. How does it work with the accept() function to establish connections between clients and servers? What does the argument “backlog” in the listen() function mean?

Answer: listen() marks the socket as a passive socket, this is, as a socket that will be used to accept incoming connection requests using accept(). The “backlog” argument defines the maximum length to which the queue of pending connections for a socket may grow (man 2 listen). In other words, it defines the maximum number of connections that haven’t been accepted yet.

Q: For the recv() and recvfrom() functions, what does the argument “size_t len” mean and what’s the difference between it and the return value?

Answer: The “len” parameter specifies the size of the buffer, that is, the maximum size of the received message. While the return value represents the length of the received message on successful completion (man 2 recv).

Task3

Q: Can the UDP server in Tutorial2 serves multiple UDP clients concurrently? Why?

Answer: Yes. Because the UDP protocol provides connectionless services. The UDP server will not be blocked by clients.

Q: What will happen if we do not close the server_socket/client_socket in client/server process?

Answer: If we don’t close the server socket in the forked client process, there will be multiple server sockets bound to the same socket address, which may introduce unknown errors. If we

don’t close the client socket in the server process, these client sockets will no longer be used or closed, which will occupy a lot of system resources.

Q: What’s the problem in non-blocking I/O?

Answer: Non-blocking I/O prohibits any clients from hogging the process forever by assigning a timeout value. However, if there are many clients to be served, these clients still need to wait for a long time. On the one hand, we should assign a small timeout value to provide timely services for waiting clients. On the other hand, a client may require a larger timeout to prepare requests (e.g., waiting for inputs from the keyboard).

Q: Find the problem of sigchld_handler function and give the right one.

Answer: Already covered in Tutorial 4.

Task4

Q: Describe the components of the structure “pollfd”. What’s the difference between “events” and “revents”?

Answer: The structure of “pollfd” can be found in the Tutorial 4 slides. “events” is a short type bitmap of the events we are interested in, while “revents” is the bitmap of the actual events that occurred (returned by the system call). Intuitively, “revents” is a subset of “events”.

Q: In the select() and poll(), how can we know which socket is ready for reading?

Answer: In select(), we have a “readfds” variable which is a set of all sockets. By using the macro FD_ISSET(i, readfds), we can check whether the socket “i” in the set “readfds” has been set. In poll(), we have a “pollfd” structure for each socket. After the poll() returns, we can check the returned values in “revents” element of the “pollfd” structure to check for the events.

Q: To enable the concurrency, what is the main difference between fork() and select()/poll()?

Answer: fork() enables concurrency by creating multiple processes that compete for the CPU. Instead, select() and poll() only use one process. At the same time, select() and poll() provide an overall blocking mechanism for checking all events. In the long run with a large number of clients fork() is expected to perform poorly compared to select /poll because of the context switching overhead in the OS.