CSCE 3600: Systems Programming

Minor Assignment 4 – Using Linux Sockets

Due: 11:59 PM on Friday, November 1, 2024

PROGRAM DESCRIPTION:

In this assignment, you will write two complete C programs to support a client/server model using Linux sockets for a UDP "ping" utility, like the ping utility already available on our CELL machines.

- **Server** ○ The server program will be called with one command-line argument, the port number being used, such as `./minor4svr 8001`. If the user calls the server program with too few or too many arguments, you will print out a usage statement and terminate the program.
    - The server will set up a UDP socket on the Internet (i.e., INET) domain and then wait in an infinite loop listening for incoming UDP packets, specifically `PING` messages from a client.
    - *Packet Loss*

      UDP provides applications with an unreliable transport service. Messages may get lost in the network due to a variety of reasons. Since packet loss is rare or even non-existent in typical campus networks, the server in this lab will inject artificial loss to simulate the effects of network packet loss. The server will simulate 30% packet loss through generation of a seeded, randomized integer that will determine whether a particular incoming `PING` message is lost or not.
    - When a `PING` message comes in and if the packet is not lost, the server will print the client message to the terminal, record statistics about the message (e.g., client IP address, number of messages received from that client), and then send a PONG message back to the client. If the packet is determined to be lost, the server will print an appropriate message to the terminal and simply "eat" the message by not responding to the client.
- **Client** ○ The client program will be called with two command-line arguments, the hostname of the server and the port number being used, such as `./minor4cli cell06-cse 8001`. If the user calls the client program with too few or too many arguments, you will print out a usage statement and terminate the program.
    - The client will attempt to connect to the server using both IPv4 and IPv6 addresses, starting with IPv6 and falling back to IPv4 if IPv6 fails
    - The client will send 10 automated PING messages to the server on a UDP socket, where automated means the message is built in the code, not entered from the keyboard. Because UDP is an unreliable protocol, a packet sent from the client to the server may be lost in the network, or vice versa. For this reason,

the client cannot wait indefinitely for a reply to a `PING` message. You should get the client to wait up to one second for a reply – if no reply is received within one second, your client program should assume that the packet was lost during transmission across the network.

- Specifically, for each of the 10 `PING` messages, your client program should:
  - send the `PING` message using the UDP socket and print a status message;
  - if the response message is received from the server, calculate and print the roundtrip time (RTT) in milliseconds for each message; otherwise, print a status message that it timed out.
- After all of the PING messages have been sent (and responses received or timed out), the client program should report:
  - the number of messages sent, the number of messages received, and the message loss rate (as a percentage);
  - the minimum, maximum, and average RTTs for all of the `PING` messages in milliseconds.
  - The client should also handle errors gracefully, such as failed socket creation or connection attempts.

Your program should run on the INET or INET6 domain using SOCK_DGRAM (i.e., UDP) sockets, so that the server and the client execute on a different CELL machine.

Given the randomness of what messages get dropped, it could be that less than or greater than 30% of the messages are dropped.

SAMPLE OUTPUT (user input shown in **bold**)**:**

==> SERVER on cell06

```
sp1568@cell06:~/csce3600/fall24$          ./minor4svr
usage: ./minor4svr <port>
sp1568@cell06:~/csce3600/fall24$ ./minor4svr 8001 [server]:
ready to accept data...
[client]: PING
[client]: PING
[client]: PING
[client]: PING
[client]: PING
[client]: PING
[server]: dropped packet
[client]: PING
[client]:  PING  [client]:
PING
[server]: dropped packet
[client]: PING
[client]: PING
```

```
[client]: PING
[client]: PING
[client]: PING
[client]: PING
[client]: PING
[client]: PING
[client]: PING
[client]: PING
[client]: PING
[server]: dropped packet
^C
```

==> CLIENT on cell05

```
sp1568@cell05:~/csce3600/fall24$ ./minor4cli
usage : ./minor4cli <hostname> <port>
sp1568@cell05:~/csce3600/fall24$ ./minor4cli cell06
usage : ./minor4cli <hostname> <port>
sp1568@cell05:~/csce3600/fall24$ ./minor4cli cell06 8001
 1: Sent... RTT=0.983000 ms
 2: Sent... RTT=0.245000 ms
 3: Sent... RTT=0.208000 ms
 4: Sent... RTT=0.237000 ms
 5: Sent... RTT=0.199000 ms 6:
 Sent... Timed Out
 7: Sent... RTT=0.483000 ms
 8: Sent... RTT=0.352000 ms 9:
 Sent... Timed Out
10: Sent... RTT=0.515000 ms
10 pkts xmited, 8 pkts rcvd, 20% pkt loss min: 0.199000 ms,
max: 0.983000 ms, avg: 0.279875 ms
sp1568@cell05:~/csce3600/fall24$ ./minor4cli cell06 8001
 1: Sent... RTT=0.652000 ms
 2: Sent... RTT=0.368000 ms
 3: Sent... RTT=0.267000 ms
 4: Sent... RTT=0.291000 ms
 5: Sent... RTT=0.359000 ms
 6: Sent... RTT=0.406000 ms
 7: Sent... RTT=0.228000 ms
 8: Sent... RTT=0.358000 ms
 9: Sent... RTT=0.350000 ms
10: Sent... Timed Out
10 pkts xmited, 9 pkts rcvd, 10% pkt loss
min: 0.228000 ms, max: 0.652000 ms, avg: 0.291889 ms
```

REQUIREMENTS:

- Your code should be well documented in terms of comments. For example, good comments in general consist of a header (with your name, course section, date, and brief description), comments for each variable, and commented blocks of code.

- Your programs should be named "**`minor4svr.c`**" and "**`minor4cli.c`**", without the quotes, for the server and client code, respectively.
- Your program will be graded based largely on whether it works correctly on the CELL machines (e.g., `cell01`, `cell02`, …, `cell06`), so you should make sure that your program compiles and runs on a CELL machine before submitting.
- Please pay attention to the **SAMPLE OUTPUT** for how this program is expected to work. If you have any questions about this, please contact your instructor, TAs, or IA assigned to this course to ensure you understand these directions.
- This is an individual programming assignment that must be the sole work of the individual student. Any instance of academic dishonesty will result in a grade of "F" for the course, along with a report filed into the Academic Integrity Database.

SUBMISSION:

- You will electronically submit your two C source code files, `minor4svr.c` and `minor4cli.c`, to the **Minor Assignment 4** dropbox in Canvas by the due date.