SysProg Assignment 3: Ping

Arno Bakker Guillaume Pierre

Deadline: 10 October 2016 at 13:29

This assignment must be submitted via Blackboard.

Assignments *must* contain:

- Four source files called `pingserver.c`, `pingclient1.c`, `pingclient2.c` and `pingclient3.c`.
- A Makefile. If we type 'make' two times without editing the source files, then the second time no compilation should occur.
- A text file called `README` containing your full name, student number and email address.

Assignments must be packaged into an uncompressed `tar` file.

If your submission does not adhere to these requirements it will be refused!

Ping

The goal of this exercise is to write a program similar to the "ping" UNIX program. This program sends one message per second to another machine via the network. The "server" machine returns the message as soon as it receives it. The "client" machine measures the round-trip time (that is, the delay between sending the request and receiving the reply).

These programs are relatively complex, so we will build them in several phases. You will write one ping server program called `pingserver`, and three ping clients called `pingclient1`, `pingclient2` and `pingclient3`. In your assignments, please return all three of them.

Attention: for this exercise, you are **not** authorized to use multiple threads or multiple processes.

1. Write program `pingserver`. This program waits for UDP messages in an infinite loop. Whenever it receives a message, it re-sends the same message back to the sender. This program **must** not take any command-line argument or keyboard input.

2. Write program `pingclient1`. This program **must** take one parameter in its command line containing the host name of the server. It does not take any keyboard input. This program sends one UDP packet (containing any message) to the host name mentioned in the command line. It then waits until it receives the reply, and displays the duration of the round-trip time. This program does not need to handle lost packets.

   Hint: to measure the duration of the round-trip time, check the `man` page of function `gettimeofday()`.

   Output format: `"The␣RTT␣was:␣%f␣seconds.\n"`, and have no other output (␣ denotes a single space). The program must return 1 on error and 0 on successful execution.

3. Write program `pingclient2`. This is an improved version of `pingclient1` that detects lost packets. Instead of waiting forever for a reply to its message, `pingclient2` sets a timeout of 1 second. If a reply has not been received within this time, then it considers that the packet has been lost. It then displays a message and stops.

   Hint: to test your `pingclient2` program, you may want to update your `pingserver` program such that it responds only to half of the messages it receives. Do not forget to remove this change before submitting the assignment!

   Output format: As `pingclient1` or `"The␣packet␣was␣lost.\n"` on packet loss.

4. Write program `pingclient3`. This is an improved version of `pingclient2` which sends one message per second in an infinite loop. Each message must contain a counter value which is incremented each time. When a reply is received, `pingclient3` checks that the value it has received is indeed the one it is expecting.

   Output format:

   - `"Packet␣%u:␣%f␣seconds.\n"` when successful
   - `"Packet␣%u:␣lost.\n"` on packet loss
   - `"Packet␣%u:␣wrong␣counter!␣Received␣%u␣instead␣of␣%u.\n"` when packets are reordered

   The output then looks like this:

```
$ pingclient3 rattler.cs.vu.nl
Packet 484: 0.481000 seconds.
Packet 485: 0.481000 seconds.
Packet 486: lost.
Packet 487: wrong counter! Received 486 instead of 487.
Packet 488: lost.
Packet 489: lost.
Packet 490: wrong counter! Received 488 instead of 490.
^C
$
```

→ You can earn a bonus point which will be added to your grade for this exercise if your code also works with IPv6.