# Assignment 2 – Can you Ping It?

Develop a program called udping that will act as either a udp ping server (which simply echoes udp ping packets back to the source), or a udp ping client which works similarly to the standard ping program.

The program must support the following command line options provided in any order:

| | | Default Value |
|---|---|---|
| -c ping-packet-count | (stop after sending this many) | 0x7fffffff |
| -i ping-interval | (interval in seconds between ping sends) | 1.0 |
| -p port number | (the port number the server is using) | 33333 |
| -s size in bytes | (of the application data sent) | 12 |
| -n no_print | (only print summary stats) | print all |
| -S Server | (operate in server mode) | client mode |

Example of use:

```
udping -S -p 50000
```

(starts the server and binds it to port 5000)

```
udping -i 0.1 -c 100 -s 200 -p 50000 <server_ip _address>
```

(send 100 pings of size 200 bytes at a rate of 10 pings per second to a server running on host with specified IP address. Options can be entered in any order)

Output notes:

(1) command line options should be echoed to the stderr stream.
(2) other output data should go to the stdout.
(3) in the default mode one line should be printed for every ping received by the client. The server should not print anything). The printed line should contain (see example below):
- The sequence number carried by the ping packet;
- The number of bytes of application data;
- The round trip time in milliseconds in the format shown.

(4) The statistics lines should be printed after the number of pings specified have been sent or the user terminates the client with a Ctrl-C.

```
tux$ ./udping -c 10 -s 300 -i 0.1 -p 33333 <srvr_ip_addr>

Count           10
Size           300   /____________________ Goes to stderr
Interval     0.100   \
Port         33333
Server_ip  <ip_addr>
(Notice the right-justified alignment)

   1    300    0.104
   2    300    0.035
   3    300    0.027
   4    300    0.035
   5    300    0.025  /____________________ Goes to stdout
   6    300    0.026  \                |
   7    300    0.025                   |
   8    300    0.050                   |
   9    300    0.025                   |
  10    300    0.025                   V
```

```
10 packets transmitted, 10 received, 0%  packet loss, time 1004 ms
rtt min/avg/max = 0.025/0.038/0.104 msec
```

If the -n option is used then printing of individual responses should be suppressed and you will print 10 asterisks instead.

```
tux$ ./udping -c 100 -s 300 -i 0.01   -n  <srvr_ip_addr>

Count           100
Size            300
Interval      0.010

**********

100 packets transmitted, 100 received, 0%  packet loss, time 1003 ms
rtt min/avg/max = 0.016/0.026/0.108 msec
```

<u>Implementation requirements:</u>

(0) You can use the getopt() function to parse the command line arguments. Use online manual to look up the arguments and use of this function.
(1) Your program MUST create two child threads. One will send pings and the other will receive them.
(2) The sender() thread MUST use the pthread_cond_timedwait() function to wait until it is time to send the next ping.  The proper time to send is:  start_time + (seq# - 1) x ping-interval. Use online manual to reseach this function.
(3) The receiver thread MUST compute the round trip time of the ping, the number of pings received, the min, max, and sum of the round trip times.
(4) Round trip times must be computed to microsecond level accuracy using the following function:
*int clock_gettime(clockid_t clk_id, struct timespec *tp)* where
clk_id is a macro set to CLOCK_REALTIME
(5) The signal() function must be used to set up handler that will print the statistics lines and the exit(0) if user enters CTRL-C.
(6) Your program should have the capability to ping more than one computer at a time.
(7) You can use Donahoo's UDPEchoClient.c and UDPEchoServer.c code as starter code.

<u>Other Requirements</u>

(0) This is a C program.
(1) Your program must unzip, compile and run on the SoC Linux machines.
(2) Programs that do not compile receive a 0 (yes, this will affect you course grade, so please make sure it compiles before you submit)
(3) Archives that are corrupt, damaged, or incomplete receive a 0 (and as above, this will also affect your course grade, so please pay attention at what you are submitting).
(4) Your archive should have no subdirectories, i.e. when "untarred" you should see only files, no directories (use *man* to lookup tar/gz commands o search engine)
(5) If you team did not make a submission due to any confusion, any circumstance, etc., there is no chance of resubmission past the late day. Communicate with your teammate. No exceptions. Better yet, check your grade book for a "to-be-graded" flag.

<u>Suggestion:</u>

Work in incremental manner.  Add 5-10 lines of code and compile. Frequent compilation saves a lot of headaches later.