

cmsc 417 programming assignment two

February 6, 2008

Your task in this assignment will be to build and maintain a neighbor table from multicast “hello” messages.

1 Context

A “hello” message tells the receiver “I’m alive.” A recipient uses this hello message to construct *soft state*: a cached view of reality that can be reconstructed should the machine fail or reboot. (Contrast “hard” state written to disk.) This soft state will consist of a list or table of immediate neighbors and how to reach them.

2 Deadline

DUE: Feb 29. That’s a friday. The late deadline will be Monday, March 3; you have only one late deadline to play with this semester.

3 Requirements

The program should do the following.

- Use as the source address on each packet the address: (`getpid() << 16`) + (last two digits of your account id `<< 8`) + (an arbitrary 8-bit number.), `ntohl`’d of course. (The destination address can remain zero; it does not matter.)
- Send a message every 25 seconds to the multicast address. It can be a protocol one message with the payload “hello” if you like. It doesn’t matter, so long as the packet is valid.¹
- Maintain a list of known and valid neighbors.
- Expire neighbors after 2 minutes of inactivity.
- Upon the command “print neighbor table” on stdin it should print, separated by commas and optional spaces, a table with the following fields in order:
IP address, Port (integer), Network address (integer from the header), Account ID, Is Alive? (Y or N), Time Remaining (seconds)
It is fine to print only “Y”s and never print N’s. N’s are optional, for your debugging of time comparisons.
- When stdin is closed, terminate.
- When the command “quit” is given, terminate.

¹Appreciation, but no bonus points, for writing parody lyrics inspired by pink floyd’s comfortably numb as payload.

The command “print neighbor table” may be sent at any time. Your code must respond within one second. That is, it is not okay to sleep for a predetermined time then check after waking up whether to print the table. “One second” is arbitrary.

Ignore extra newlines.

Stdin is line-buffered, you will not have to worry about reading one character or word at a time.

Ignore commands that are not “print neighbor table” or “quit” unless you want to engineer your own interpreter. (You’re welcome to add commands as you like.)

All printed values must be in host byte order (specifically, the port number of the remote machine must be correct).

Your executable must be named “two” (for the testing scripts to operate correctly). It must not require an argument.

Don’t even think about calling `sleep()`.

Expiration may be eager (set an event for when it times out, then reset that event on each hello) or lazy (note the time when it should be expired, then clean out any expired entries when the table is accessed).

Tests will be straightforward, whether the table gets entries it should, expires entries that are stale, tracks addresses properly, etc.

You may assume that network layer addresses are unique, and that ip address/udp port pairs are unique. A new network layer address for a known ip address and port should replace the old.

Ensure that your program does not leak memory.

4 Hints

Implement an event queue; at the moment, you have only one event (send the hello message each minute) and all other processing is input driven (receive a hello message or input from stdin). You may have to implement more periodic events for later assignments.

The `select` or `poll` system calls will be *necessary* for completing this assignment. While `poll` is more modern, `select` is more traditional. The file descriptor for stdin is 0 or `STDIN_FILENO` in `unistd.h`.

A `read()` or `fread()` or `fgets()` or `gets()` on stdin will return differently if “end of file” has been reached. If so, `exit`.

Do *not* use `O_NONBLOCK`, which would allow you to poll for new input. Polling leads to both wasted processor cycles (wasting battery and ultimately coal) and ugly design. It is possible to use non-blocking sockets effectively (especially for system calls like `connect()`), but not in this assignment.

Do *not* attempt to use threads or separate processes for different components of this assignment. Threads are nice abstractions that (sometimes) simplify concurrent programming when the amount of state required for a conversation dwarfs the shared state of the global program. Here, the global state (neighbor table) is much more important than the conceptual tasks that would have to lock the table on every access.

Do *not* use `sigaction()` and `alarm()` to interrupt slow system calls. That the kernel can interrupt a system call after a while is nice, but adds new ways for the call to fail and requires more complexity than you’d like (not to mention that signals and threads don’t get along).

5 Objectives

At the end of this assignment, you should understand:

- `select`.
- timers.
- Hello messages and the hello interval.
- Basic LLC behavior.