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. Requirements

Use the packet format described in the previous assignment. The program should do the following:

- Source address = ( getpid() << 16 ) + (last two digits of your account id << 8) + ( an arbitrary 8-bit number. )
- The destination address remains 0
- Send a message every TIME-X (default value: 25) seconds to the multicast address. It should be a protocol one message with the payload “hello”.
- Maintain a list of known and valid neighbors.
- Expire neighbors after TIME-Y (default value: 100) seconds of inactivity.
- Upon the command “print neighbor table" on stdin it should print on a line, separated by commas and optional spaces, 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)
- You may assume that network addresses are unique, and that ip address/udp port pairs are unique.
  - A new network address for a known ip address and port should replace the old.
- When the command “quit" is given, terminate.
- Ignore commands that are not “print neighbor table" or “quit"
- When stdin is closed, terminate.
- Ensure that your program does not leak memory.
- Your executable must be named “two" (for the testing scripts to operate correctly).

./two [TIME-X TIME-Y port]
If no arguments are provided, the defaults should be used (same address and port as previous project).

3. Hints

- Ignore extra newlines.
- Stdin is line-buffered, so you will not have to worry about reading one character or word at a time.
- 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 set Is Alive to N for any expired entries when the table is accessed).
- The poll() or select() system call will be necessary to complete this project.
- 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 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.
- You could implement an event queue. For this assignment you have only one event (send the hello message every TIME-X seconds) 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.