Program Assignment 4

CMSC 417 Fall 2011

Nov 07

1 Deadline

Nov 21, 2011.

2 Introduction

In this project, you will extend the previous project (Project 3) and implement a multicast program that maintains a list of neighbor nodes with their state.

A "hello" message tells the receiver that "I'm alive." Your program will send a "hello" message to the multicast address periodically. Your program will also receive "hello" messages from other multicast programs. Then you need to build a neighbor list using the received "hello" messages.

3 Team Project

You can build a team for this project. Number of members should be less than or equal to 3 unless you have any particular reason.

There is no incentive for small teams in this project.

Each team may submit one project. That is, only one of the team members may submit the project.

4 Requirements

1. Packet Format

    - Use the packet format described in the previous assignment.

- Source address = (getpid() << 16) + (last two digits of your account id << 8) + (an arbitrary 8-bit number.)
- The destination address remains 0.

2. Implementation

   - Send a message every TIME-X (default value: 25) seconds to the multicast address. Use the same protocol as the one in Project 3; use the same header 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, 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) >>
   - 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 (TIME-X:25, TIME-Y:100, Port Number:5050).

5 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"). Alternately, expiration may be lazy (note the time when it should be expired, then set "Is Alive" to N for any expired entries when the table is accessed).
  - That is, you don't have to expire a neighbor exactly when TIME-Y elapsed from the last "hello" message. You can set "Is Alive" to N when "print neighbor table" command is input.
- The select or poll system calls will be necessary for completing this assignment. While poll is more modern, select is more traditional.
- 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 may 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.