Project Overview

1. Create topology (make udp sockets, one for routing and one for data)
2. Implement routing (control) thread
   - Track neighbors: Send hello packets to your direct neighbors, receive their hellos, keep track of neighbors using TTL
   - Implement your routing protocol – e.g., if distance vector, periodically send your routing table to your neighbors; receive theirs and update your table if necessary
3. Implement your forwarding (data) thread
   - When you receive a data packet, check destination
   - If you are destination, receive it
   - If not, check routing table for next hop and forward the packet
Sleeping/Implementing timers

- You will have to use threads to implement your program
  - One thread for control (routing/hello) and one for data (forwarding packets)
- Use `usleep` and it sleeps only one thread (example to follow)
- I am guessing `sleep` works the same way
- use `usleep`; it gives you finer control over time (microseconds instead of seconds)
- Need to `#include <unistd.h>`
usleep() Example

```c
#include <pthread.h>
#include <unistd.h>
#include <stdio.h>

void *slowthread(void * i) {
    int j=0;
    while(++j) { //print Hi once every 5 seconds
        usleep(5000000);
        printf("Hi %d from slowthread\n",j);
    }
}

void *fastthread(void * i) {
    int j=0; //print Hi once every 1 second
    while(++j) {
        usleep(1000000);
        printf("Hi %d from fastthread\n",j);
    }
}

main() {
    pthread_t *thread;
    int dummy;
    pthread_create(thread, NULL, slowthread, (void *) &dummy);
    fastthread((void *) &dummy);
    pthread_exit(NULL);
}
```
Output

Hi 1 from fastthread
Hi 2 from fastthread
Hi 3 from fastthread
Hi 4 from fastthread
Hi 1 from slowthread
Hi 5 from fastthread
Hi 6 from fastthread
Hi 7 from fastthread
Hi 8 from fastthread
Hi 9 from fastthread
Hi 2 from slowthread
Hi 10 from fastthread
Hi 11 from fastthread
...

- Each sleeps independently without blocking the other
Recall Overall structure of Program

- Two threads, one for routing/control and one for forwarding/data

- Routing thread: (1) Exchanges hello messages; (2) Exchanges routing messages; (3) Builds routing table; (4) Interacts with control client (optional)

- Data thread: (1) Receives incoming data packets; (2) Forwards it if we are not destination according to routing table; (3) If you do not implement control client, should with low probability generate packets to random destinations
Data Thread – Event Driven

1. In a loop, send out any locally generated packets

2. Wait for incoming packets (use select, with a reasonable timeout value, say 0.1 sec)

3. If you receive an incoming data packet, check if you need to forward it or receive it locally and do so
Routing Thread – Time Driven

• Also runs in a loop
  – Use usleep with a suitable period (say 0.1 sec) to control loop interval
  – Things here happen periodically (e.g., should send hello packets every $n$ seconds); keep a count for every event in terms of the loop granularity (a count of 50 will give you 5 seconds if your period is 0.1 seconds)

• Every loop
  1. Check if you need to generate control packets (hello or routing) based on their count expiring
  2. Check for control packets (hello, or routing) and update your routing table accordingly
    – Behavior will vary depending on whether you use distance vector or link state
Packets and Bits

• Three types of packets:
  – Data packets do the work of carrying the data.
  – Hello packets (or heartbeat packets) keep neighbors informed that the link between them exists.
  – Routing packets, exchange routing info according to your routing protocol.
  – For the optional client part, you have other control packets.

• Data packets are received on the data port, everything else on the routing port.
Data Packet Format

- Should have fields for:
  - Packet Id, so that we can track it
  - Source and Destination Id
  - Previous hop (this allows us to tell what link it came on since all of the links share the same port)
  - Protocol Id (necessary to implement upper layer protocols used in bonus parts)
  - Time to Live (TTL)
  - Data

- Like project 1, you can use memcpy to assemble/disassemble packets in character buffers to send them over

- You should generate packets periodically to random destinations if you do not implement the optional control client
Hello Packet

- Hello protocol simply keeps hosts aware of their neighbors
  - Each host should have a data structure (array or linked list) of its immediate neighbors
  - Everytime they receive a hello packet from them, they refresh the TTL entry on the neighbor list (lets say starting value is 1000 timer pulses)
  - Everytime the routing thread wakes up reduce TTL by 1, if it reaches 0 remove the neighbor
  - If a link goes down, we stop receiving packets

- What should the packet format be? Hello Opcode and Source Id is enough
Routing Packet

- Depends on your routing protocol
  - Distance Vector: should include your full routing table
    * Example: [Opcode—Source—host1—Cost1—host2—cost2—...]
  - Link State: Send only your neighbor list
    * Example: [Opcode—Source—SequenceNumber—Neighbor1—Neighbor2—...]
    * Note that you have to flood this (send it to all other neighbors) the first time you receive this sequence number

- When you receive a routing packet, you should update your routing table according to your protocol