CS 415: Operating Systems Practicum

Project 3- Implementing UDP

Oliver Kennedy
okennedy@cs.cornell.edu

Updates to project 2

- New threads should come in at high priority
- The project 2 section notes say to use time().
  - For project 3, replace calls to time() with accesses to the global variable ticks
    - You will need to add the following line to clock_handler: ticks += PERIOD;

Goals

- We have a threading package
  - What are threads used for?
    - Complex calculations (sieve)
    - Servers!
- Let’s do some networking.

What do you get to work with?

- You get a means of...
  - identifying a host (network_address_t).
  - sending data to a particular host.
    - network_send_pkt()
  - being alerted when new data arrives.
    - network_initialize(handler);
- All defined in network.h

Why isn't this good enough?

- We're multiplexing the processor, we also need to multiplex the network card.
- The network card sends and receives messages.
  - Create multiple virtual network cards.
  - Call them "ports".
  - A port can send messages to another port.
    - ... on the same machine or another

What goes into a port?

- Two types of Ports: Local and Remote
- Local and Remote should store
  - Port #
  - Host Address
- Local should also store
  - Message queue

The Basics

- minimsg_initialize()
- miniport_local_create()
  - allocates a local port and prepares it for receive operations.
- miniport_remote_create()
  - allocates a remote port and prepares it for send operations
- miniport_destroy()

Anatomy of a Packet

- Header
  - Who sent it (address, port)
  - Who is it meant for (port, address?)
  - Body Size
- Body

Working with memory

- sizeof() returns the number of bytes used by a datastructure
- char *foo1 = malloc(4) is the same as
  - int *foo2 = malloc(sizeof(int))
- ((int*)foo1) can be used as if it were foo2

Working with memory

- Basic types
  - char *buffer = [headerbytes][bodybytes]
  - header_t header; char *body;
- Extracting data from buffer
  - Header: (header_t*)&(buffer[0])
  - Body: &(buffer[sizeof(header_t)])

minimsg_send()

- Check for error cases (invalid params)
- Create the header
- For a remote port
  - network_send_pkt()
- For a local port
  - Create a fake network_interrupt_arg_t
  - Enqueue it

minimsg_receive()

- Check for error cases
- Block until the local port has something in its queue
  - Semaphores!
- Extract the header, body, and length
- Create a new port representing the sender and save it in ‘remote’
  - Either a remote port or a “virtual port”
  - Application responsible for cleaning this up
- free() the packet contents

Interrupt Handler

- Interrupt handler should be fast!
  - Pull processing out into another thread
- Create a global to-be processed queue.
- Make a thread like the garbage collector
  - Peek at the header and get the port
  - Enqueue the packet on the port's queue.
- Need a datastructure to map ports to miniports.

Gotchas

- Synchronization
  - Interrupt handler should not need to obtain locks.
  - Be careful which synchronization method you use.
- Semaphores
  - V() should NOT block!

Other notes

- Local ports all need an ID.
  - Use hard-coded ports provided by the app.
- Test with network[1-6].c
- 3 test cases included for alarms/preemption
- Message sizes/Body sizes are capped!

Good Luck