Project 4

Multi-Core Network Honeypot
- Important for mutex locking/unlocking
- Crucial for synchronized data-structures
- Up to 32 cores in PA4
LL/SC Syntax

- LL a, off(b): loads M[b + off] into register a
- SC c, off(d):
  - Attempts to store the value of c into M[d + off].
  - If M[d + off] has changed since the last LL instruction then c = 0 and M[d + off] stays the same.
  - Otherwise M[d + off] = c and c = 1
You are to design a network *honeypot*:

- receives packets from a network device
- analyzes and classifies those packets
- tracks various statistics over time

Your honeypot will be simulated on a multi-core MIPS and simulated I/O devices.
Project Goal

- Bundles of data = “packet”
- Max size of packet is 4kB
- Receive packets as fast as possible (maximize throughput)
- Analyse all packets and gather statistics
Important Files

After you have read through most of the code that we give you, your focus should be on:

- kernel.h/c
- network.h/c
- honeypot.h/c

If you feel overwhelmed, don’t worry: you will not have to touch most of the other files.
Packets

Three categories:
- Vulnerable, spammer, evil
- Command
- Print

Detailed descriptions of each packet category is on the main project page.
Interrupts

- One of the two main ways to handle packet reception.
- Interrupt occurs when packet arrives
- Simple implementation (may or may not be easier than polling)
- Slow and will result in poor performance during network spikes
Polling

- This is the second way to receive packets
- Checks continuously if a packet arrived in the “packet ring” (explained later)
- Needs a core on polling duty
- Very fast and not a bottleneck if implemented correctly
Packet Ring

- Array of 16 (address, length) tuples in memory
- Has a “head” and “tail” (essentially a ringbuffer)
- When packet arrives,
  - it will be written to \textit{paddr} in the tuple under the head
  - head moves to the next tuple
- Make sure the memory where the packet arrives is allocated
Packet Ring (continued)

Packet Ring

(paddr, len)
DGB2 Hashing

- Same as for your hashtable
- Takes in a pointer to a sequence of bytes
- Returns an unsigned long
- You need to use this function on every packet
- Very time consuming
- Is the bottleneck in functioning systems
- Hard code some of it
- Unroll some of the loops (see FAQ)
- In sum: optimize it as much as you can
Milestones

- Start the project (*seriously* this time, start this early...we are not joking, and there are no slip days)
- Turn on simulated network card and receive/drop packets
- Prepare Design Doc meeting
- Receive first 17 packets without dropping
- Handle one of each type of packets
- Implement and synchronize shared data-structures
- Synchronize malloc / find a way around it
- Print out statistics
- Parallelize analysis of packets
- Optimize until due date
Expectations

- The measure of how good your final project is throughput (bits worth of packets you can analyse per unit time)
- You need to be dropping very few packets
- Aim for 10 Mb/s (don’t panic with lower throughput)
- Highest ever is 61 Mb/s
Due Dates

- Schedule Design Doc meeting - May 3rd
- Design Documentation - May 7th
- Schedule Final Presentation - May 10th
- Final Presentations/Demos - May 13-14th
- Final Code Submission - May 14th
1. We cannot emphasize this enough; get started early: there will be fewer people in office hours further away from the deadline.

2. Keep your code clean: your codebase will be significantly larger than any other project for this class. Make sure you can read through it.

3. Version control: this should be a reflex by now, there is no reason not to do it. Keep it private (e.g: bitbucket).