Soft Timers: Efficient Microsecond Software Timer Support for Network Processing

Mohit Aron
Peter Druschel

Presenter:
Christopher Head
Problem: Network Bandwidth

- Bandwidth: 1Gbps
- MTU: 1518B
- Rate: ~82000 pps
- Period: ~12μs ~= 36000instructions@3GHz
- Overhead of protection domain switch for interrupt handling too large!
Solution: Interrupt Thresholds

- Program NIC to interrupt on every $n$ received packets for $n>1$
- Problems:
  - Latency increased
  - Stream stops suddenly? Packets stuck in NIC?
  - Receive burstiness increases burstiness of transmit in reactive protocols (e.g. TCP)
  - Not supported by all NICs
Solution: Interrupts+Polling

- Disable interrupts when too much activity and poll instead
- Linux NAPI
- Problems:
  - When to poll? Latency increased
  - Only useful on receive
Better: Soft Timers

- Disable interrupts when too much activity and poll instead
- Poll at very low time intervals, but probabilistically
- Avoid interrupt overhead altogether by polling only when already interrupted
Performance

- Overhead for null event from 0.8µs to immeasurably small
- Overhead with cache/TLB-polluting event every 10µs from 5.1µs to 3.5µs
- Almost always very close to desired trigger time: mean triggers at ~2–30µs despite 1000µs hardware tick
Improvement: Cheap Timer

- APIC timer characteristics:
  - Adding event cheap (register access)
  - Firing event expensive (interrupt → context switch)

- Idea:
  - Add often: establish cap on timer period
  - Fire rarely: normally use soft timer; APIC fires only if soft timer too slow
Questions

• Is it effective in handling a lot of packets?
  • Yes: allows userspace to keep working even under heavy load
  • Receiving a lot of packets is useless if application can't run to deal with them!

• Can it be used for disk access as well?
  • NAPI/Polling Idea: OK to drop received packets under heavy load if it helps userspace keep running
  • Disk: not OK to drop IO requests!
  • Disk requests often synchronous
Questions

• Have soft timers been implemented for real?
  • Linux: not exactly, but some aspects are:
    - Network NAPI: polling (better than interrupts, not as good as soft timers (?))
    - Tickless system: APIC idea: use hardware timer to schedule next event at high resolution rather than periodic ticks at low resolution

• Hard real time semantics?
  • APIC enhancement: set \textit{deadline} for real time deadline, \textit{target} for earliest time event can be handled
Questions

• What if system load is low?
  • Soft timers handled by an idle CPU all the time
  • Switch NIC to interrupt mode if load is low (e.g. Linux NAPI)

• Dynamically adjust hardware timer rate to keep soft timers responsive?
  • APIC enhancement: guarantees *deadline* met
  • Authors' claim: apps making few syscalls are not doing heavy network IO
    – Shared machines?
Questions

- Why are cache/TLB pollution minimized?
  - Code to *enter* or *leave* interrupt context executed once instead of twice
- Given variability over 1ms, are soft timers still useful?
  - APIC enhancement
- CPUs are very fast; is it still worthwhile?
  - Long pipelines: context switches very expensive
Questions

- POSIX defines timers to ns granularity; how are these implemented today?
  - Hardware timers
  - Nanosecond *resolution* is *not* provided
  - Linux Tickless System: like the APIC idea
    - High resolution with infrequent firings
Questions

• Why isn't this done today? The old way still works?
  • Linux/NAPI: Polling
    – Not exactly the same as soft timers
    – Same goal: prevent livelock where kernel spends 100% of time handling interrupts and 0% in userspace getting useful work done

• Network utilization
  – NOT very good today
  – Networks must be very over-provisioned to avoid catastrophic chronic throughput breakdown
Questions

• Multicore: just throw a core an interrupt handling instead
  • Web servers are embarrassingly parallel
  • Soft timers cost nothing
  • Why waste CPU time, even if you have a lot?