Eliminating Receive Livelock in an Interrupt-driven Kernel

Paper by
Jeffrey C. Mogul
mogul@wrl.dec.com
K. K. Ramakrishnan
AT&T Bell Laboratories
kkrama@research.att.com

1996 USENIX

Talk by
Prof Arthur Goldberg
NYU
2004
• **Problem: receive livelock**
  – the system spends all its time processing interrupts, to the exclusion of other necessary tasks

• **Solution: load dependant scheduling**
  – The OS lowers the priority of network interrupt handling as load increases
OS Scheduling Techniques

• Interrupts
  – When a task requires service, it generates an interrupt. The interrupt handler provides some service immediately.

• Polling
  – When a task requires service, it turns on a flag. The OS polls the flags periodically, and provides some service to the tasks with flags turned on.
Benefits of Scheduling Techniques

• Interrupts
  – Much lower overhead
  – Rapid response

• Polling
  – Perform better under heavy load
How interrupt-driven scheduling causes excess latency under overload

Timeline of Ethernet packet processing

- link-level processing at device IPL, which includes copying the packet into kernel buffers
- further processing following a software interrupt, which includes locating the appropriate user process, and queueing the packet for delivery to this process
- finally, awakening the user process, which then (in kernel mode) copies the received packet into its own buffer.
• **Under Load, Avoid Livelock**
  – Use interrupts only to initiate polling.
  – Use round-robin polling to fairly allocate resources among event sources.
  – Temporarily disable input when feedback from a full queue, or a limit on CPU usage, indicates that other important tasks are pending.
  – Drop packets early, rather than late, to avoid wasted work. Try to process packets that are received to completion.

• **Unloaded, Maintain High Performance**
  – Re-enable interrupts when no work is pending, to avoid polling overhead and to keep latency low.
  – Let the receiving interface buffer bursts, to avoid dropping packets.
  – Eliminate the IP input queue, and associated overhead.
Livelock in a Unix router
• filled circles: kernel-based forwarding performance
• open squares: the *screend* program, a firewall application

**Figure 6-1:** Forwarding performance of unmodified kernel
Why livelock occurs in the 4.2BSD Router

Figure 6-2: IP forwarding path in 4.2BSD
Fixing the livelock problem
- Do as much work as possible in a kernel thread
- Eliminate the IP input queue and its associated queue manipulations and software interrupt
Guaranteeing progress for user-level processes

Figure 7-1: User-mode CPU time available using cycle-limit mechanism