# SMPng Network Stack Update

11 May 2006

Robert N. M. Watson

Security Research
Computer Laboratory
University of Cambridge

UNIVERSITY OF
CAMBRIDGE

# Introduction

- SMPng
  - Move from Giant-locked kernel to more granular locking
- Network stack
  - One of the most complex subsystems in the FreeBSD kernel
  - Millions of lines of code
  - Hundreds of components

# Protocol Stack MPSAFEty

- As of FreeBSD 5.4, Giant disabled by default
- Giant selectively-reenabled at boot-time if certain features compiled in
    - NET_NEEDS_GIANT(“subsystem”);
    - I4b, netatm, ng_h4, ipsec, ipx_ip
- Covers the entire network stack including from sockets, netisr, ithread, callout
    - May be able to explore more granular acquisition of Giant if desired

# Device Drive MPSAFEty

- Giant acquired around certain interfaces
  - ifp->if_flags |= IFF_NEEDSGIANT;
  - Requires IF_LOCKGIANT() and deferred if_start
  - Significant performance issue for drivers
- if_ar, fi_arl, if_awi, smc90cx6, if_cnw, if_cp, if_ce, if_cs, if_ct, if_cx, if_ex, if_fe, if_fwe, if_fwip, if_ie, if_ic, if_lnc, if_pip, if_ray, if_sbni, if_sbsh, dp83932, if_sr, if_tx, if_aue, if_axe, if_cdce, if_cue, if_kue, if_rue, if_udav, if_ural, if_xe, if_ppp, if_sl

# Key Driver MPSAFEty Issues

- Certain frameworks must be made MPSAFE to remove Giant support for network interfaces
  - USB framework
  - Firewall framework
  - TTY framework
  - I4b

# 7.0 Giant Elimination Goals

- Eliminate NET_NEEDS_GIANT, some IFF_NEEDSGIANT
  - Replace KAME IPSEC with FAST_IPSEC once IPv6 support is complete (gnn)
  - Remove netatm, we already have at least two other ATM stacks that are MPSAFE (rwatson)
  - MPSAFE i4b (?)
- Fewer IFF_NEEDSGIANT consumers
  - TTY locking followed by SLIP/PPP locking (phk, rwatson)

# But There's More To Life Than Giant

- Removing Giant was good
  - Much improved parallelism, especially with respect to many processes reading from sockets at once
  - Improved latency for interrupt handling, etc
- Removing Giant is not enough!
  - A moderate number of locking loose ends
  - After that, you have to measure and optimize

# Significant Loose Ends

- Ifnet locking leaves much to be desired
  - Despite gradual improvement, locking conventions for ifnet still weak or not present
- Address list locking
  - As discussed yesterday, we most often don't
  - Netatalk prototype suggested that this was a significant amount of work
  - Races hardly ever exercised, cost of fixing high (real hours, overhead)

# Models for Parallelism

- Locking is about data structure and algorithm integrity in the face of parallelism
- However, you also need the opportunity for parallelism
  - Represented in our model by threads
- This means assigning work to different threads
  - We get quite effective parallelism between user processes, input vs other code, in ithread etc
  - Direct dispatch, fast forwarding further explot

# Models for Parallelism (2)

- However, a number of places we don't get effective parallelism
  - TCP/IP input processing due to single netisr
  - Callout wheels for protocols
  - Simultaneous send/send, send/receive, receive/receive on a socket
  - Single send pipeline
- Techniques for improving parallelism at one layer may reduce it at another
  - Direct dispatch

# Models for Parallelism (3)

- Lots of ideas being kicked around
  - Multiple netisrs, assigning work based on various things (source, IP layer characteristics, etc)
  - Netisr -> netisr_up, introduce netisr_down
  - Direct dispatch or fast forwarding by default
  - Break out higher level IP protocols from ip_input()
- What is needed is experimentation and extensive analysis/measurement
- Challenge: how to avoid optimizing just one application at the cost of many others

# More Loose Ends

- Ifnet queue dispatch model
- Socket upcalls
- Mbuf allocator race/bug

# Recent and Ongoing Work

- Socket/protocol API normalization
- True PCB reference model in TCP
- UNIX domain socket lock granularity
- Network stack consumer locking
- Netisr loopback traffic issue

# Socket/Protocol API Normalization

- API between socket layer and protocols left something to be desired
    - Reference model by which PCBs were torn down allowed PCBs to disappear "at any time"
    - Very memory efficient, but very weak invariants
    - Lead to lots of locking in protocols
- Strengthen invariants
    - so_pcb entirely owned by protocol, explicit strong reference to socket from protocol
    - so_pcb != NULL invariant in all protocols

# True PCB Reference Model in TCP

- Pcbinfo lock used to prevent PCB garbage collection during use by in-bound network path
  - Prevents any parallelism in tcp_input(), timers, etc.
  - Not such a problem with one netisr, but with direct dispatch, multiple netisrs, is an issue
- Permit pcbinfo lock to be released using true reference model
  - Being prototyped by mohans
  - Required socket/protocol sanitization

# UNIX Domain Socket Locking

- In original BSD/OS prototype, global lock ("pcbinfo") and per-PCB locks
  - However, global lock basically always necessary
  - We shipped with just subsystem lock in 5.x, 6.x
- Contention in SMP applications (MySQL)
  - Posted patch introduces PCB locks
  - Global lock often required, but held much less
  - Reasonable measured performance improvement
  - Concerned about overhead

# MPSAFE Network Stack Consumers

- Sendfile() now no longer acquires Giant
- NFS server now acquires Giant only for VFS
- MPSAFE NFS client patches from mohans now being tested by kris

# Netisr loopback issue

- With PREEMPTION, netisr will wake up and preempt sending threads for if_loop prematurely
  - Results in significantly degraded loopback performance
- Discussed extensively with jhb and others
- Currently investigating deferred wakeup model, which would reintroduce coalescing of wakeups
- Interested in bulk handoff for transmit

# Performance Generally

- Looking much better in 6.x
  - Critical section/mbuf changes
  - Most important things MPSAFE
  - PREEMPTION reduces latency for interrupts
- Still a lot of work to do
  - In some cases improve locking granularity
  - In others, decide if we have too much
- TCP performance, PPC performance much lower than we'd like

# Things That Need Owners

- Device driver frameworks
  - USB, Firewire
- Many device drivers
- Protocol stack consumers
  - netncp, netsmb
- Holding a stick to phk over ttys
- Continued performance measurement and optimization throughout the stack

# Critical Application Performance Targets

- LAMP
  - HTTP performance
  - MySQL performance
- Netperf
  - Raw TCP performance, UDP performance
- Bridging/forwarding rate
- Your items here...?

# Conclusion

- Vast majority of SMPng goal accomplished
  - Almost all of network stack is Giant-free
- Now it's time for measurement, inspection, refinement
  - Moderate sized tasks remaining with reasonable payoff in performance, architectural improvement
  - Lots of room for aggressive benchmarking, profiling, optimization