Dublin City University

Electronic and Computer Engineering

EE562 Network Stack Implementation

Notes

Author

Michael Lenehan michael.lenehan4@mail.dcu.ie
Student Number: 15410402

Contents

1 Intro

1.1 Userspace vs Kernel

- Modern processors have 2 modes of operation
  - Intel - Ring 0/3
  - ARM - Supervisor/User
- Code in Ring 0 > Priority than Ring 3
  - More instructions from set
  - More resource access
- Allow for improved security
- System code = kernel

1.2 Monolithic vs Microkernel

- Monolithic
  - 1 multi-tasking prog. implements OS funcitons
    - Add new functionality requires kernel rebuild
    - Linux, FreeBSD
- Microkernel
  - Most services run in userlan, min. code is privileged
  - Enhances modularity/maintainability at expense of performance
  - Mach, GNU Hurd, Minix 3
- Hybrid Kernel/Macrokernel
  - Most services run on microkernel on Ring 0
  - Windows > NT

1.3 Loadable Modules

- Dynamically loaded at runtime
- Access kernel structures/methods which have been exported

1.4 OSI Model Limitations

- OSI Model is approx. guide
  - Session/Presentation not in most networked apps
  - Tunneling can result in multi. stacked L3 protocols
  - Ethernet Swithces nominally operate at L2, supposed to be "data-link" protocol layer.
  - Control plane does not feature in model

1.5 Bit Rates and Packet Rates

Packet Rates depend on:

- Physical Bit Rate
- OH
  - Extra bits + by lower layers/protocol features
- Pkt length
- Physical BR vary from V.Low to V.High
- Useable BR $<$ Nominal BR

– To inc. data transfer rate of X from Transport to higher layers, bit rate at phy layer must be $\gg X$

- Net SW must be written so net equip can operate at wire speed
  - HW and SW in net device rd/wr 2 pkts from link w/o dead time

1.6 Headers/Trailers

- Each pkt comprises of Protocol Data Unit and new layers head/tail
- Exception: Multiplexing, Fragmentation/Re-Assembly

1.7 Buffer Management

- Memory space ("socket buffer") req. to store data varies between layers
- Solutions:
  - Allocate new buffer
    - Buffer contents PBV, cp to larger buffer (with offset for header)
      - Lots of cp ops.

  - Oversized Linear Buffers (Linux)
    * How big skt buff is at lowest level implemented
    * Allocate buffer of this size, offset pointer to it by aggregate size of lower level headers
    * Good performance, breaches layering principle
  - Linked Minibuffers (BSD Unix)
    * Header/PDU/Tailer stored as linked list
    * Head/Tail + by inserting minibuffers at start/end of list
    * Memory-efficient, lower performance

2 Kernel Issues

2.1 Kernel Programming Issues

- Library functions not available
- Limited Stack - Small, fixed-size stack for each proc
  - kmalloc args 1=Size of block, 2=how kmalloc works
  - GFP_Kernel - kmalloc puts curnt. proc to sleep, waits for page when called in low mem situations
  - Funciton using GFP must be reentrant, otherwise use GFP_Atomic
- Kernel functions must be rentrant
- Linux kernel fully preemptible
  - kernel proc can be preempted by > priority proc
  - Should avoid global static vars in network code.
  - Need to use spinlock/semaphores to protect access to global structs
- Linux supports Symmetric Multiprocessor (SMP) architectures
  - Mutli. processors share common mem, potential issues
  - Harder debugging
  - Bugs cause catastrophic failure, require reboot

2.2 Memory Allocation in the Kernel

- Slabs ...

2.3 Kernel Scheduling

- Multi. Scheduling Classes
- Various policies within class
- Higher priority class served first
- Tasks can migrate between classes, policies, CPUs
- Completely Fair Scheduler
  - Decides which process is executed next
  - Uses red-black tree to decide on proc.
  - Implements Weighted Fair Queuing

Class	Policies
Stop	none
Deadline	SCHED_DEADLINE
Real Time (RT)	SCHED_FIFO, SCHED_RR
Fair (CFS)	SCHED_NORMAL, SCHED_BATCH, SCHED_IDLE
Idle	none

Class	Features
Stop	SMP-only, cannot be preempted, preemts all
Deadline	For periodic RT tasks
Real Time (RT)	for Posix "real-time" (low latency) tasks
Fair (CFS)	All other system tasks
Idle	Nothing to do - enter low power mode

2.4 Symmetric Multiprocessing (SMP)

- Like parallel processing
- Multi resources share single bus with multi. processors
- Contention for resources
  - N cores not N times faster than 1
  - Need to lock shared resources when in use

2.5 Reentrancy and Preemption

Preemption

- Stopping code to run other code

Reentrancy

- Preemting process might call preemted code
  - Preemnted code must support multi. execution instances

2.6 Writing reentrant code

- Avoid global vars, use automatic vars/kmalloc instead
- Use spinlock/semaphors to ensure global var access is atomic
- Do not call other functions, unless they are reentrant
- Accessing HW causes issues, use spinlock/semaphores to lock out other proc.
  - Causes bad RT performance
  - Fix - Disable interrupts
  - Most interrupt handlers do this

2.7 Spinlock

- Crude method to protect data struct
- Single int field as lock
  - Proc withing to access protected region must check lock val
  - If 1: proc. retries (spins) in tight loop of code
  - If 0: proc. enters region, sets value to 1
- Accessing mem location must be atomic - cannot be interrupted
- Winning proc is random/arbitrary

2.8 Semaphores

- Protect critical regions of code/data structs
- Queueing mechanism to give order to access
- Count
  - Num of proc waiting for resource
  - +: resource available
  - -/0: procs waiting
  - Initially set to 1
  - Requesting resource dec. count
  - Proc finished - inc. count
  - Abs val of count = num of sleepers
- Wait
  - Queue in which sleeping procs stored

2.9 HW Interrupts

- I/O Interrupts
  - Keyboard Press, USB bus has data, pkt arrives in NIC
- Timer Interrupts
  - Maintaining System Clock, Executing sched alg
- Interprocesspr Interrupts (IPI)
  - Only in SMP systems
  - On ARM processors
    - IPI_RESCHEDULE “Rescheduling interrupts“
    - IPI_CALL_FUNC “Function call interrupts“
    - IPI_CPU_STOP “CPU stop interrupts“
    - IPI_CPI_CRASH_STOP “CPU stop (for crash dump) interrupts“
    - IPI_TIMER “Timer broadcast interrupts”
    - IPI_IRQ_WORK “IRQ work interrupts”
    - IPI_WAKEUP “CPU wake-up interrupts”
    - Used for e.g. power management, moving procs between CPUs

2.10 Timer Interrupts

- Triggered by Programmable Interval Timer/High Precision Event Clock
  - ISR (interrupt service routine) implement jiffy clock, invoke proc scheduling
- SMP systems have local CPU timers
  - Used for profiling kernel code, timing procs.