opensolaris

Chapter 2 Process， thread, and scheduling

—— *kernel services*

Zhao Xia
zhaoxia@os.pku.edu.cn

Outline

- Kernel Services
- System call
- Trap
- Interrupt
- kernel callout
- system clock

Access to Kernel Services

- User mode
- kernel mode
  - Access Kernel data structures and hardware devices
- When a user process needs to access kernel system services
  - thread within the process transitions from user mode to kernel mode through a set of interfaces known as *system calls*

Enter the kernel mode

- system call
  - user process requests a kernel service
- processor trap
  - vectored transfer of control into the kernel, initiated by the processor
- interrupt
  - vectored transfer of control into the kernel, typically initiated by a hardware device

Context of thread

- describes the environment for a thread of execution
- Execution Context
  - thread stacks, open file lists, resource accounting, etc.
- virtual memory context
  - set of virtual-to-physical address translations
  - Each process has its own virtual memory context
  - each process context has kernel's virtual memory mapped within it

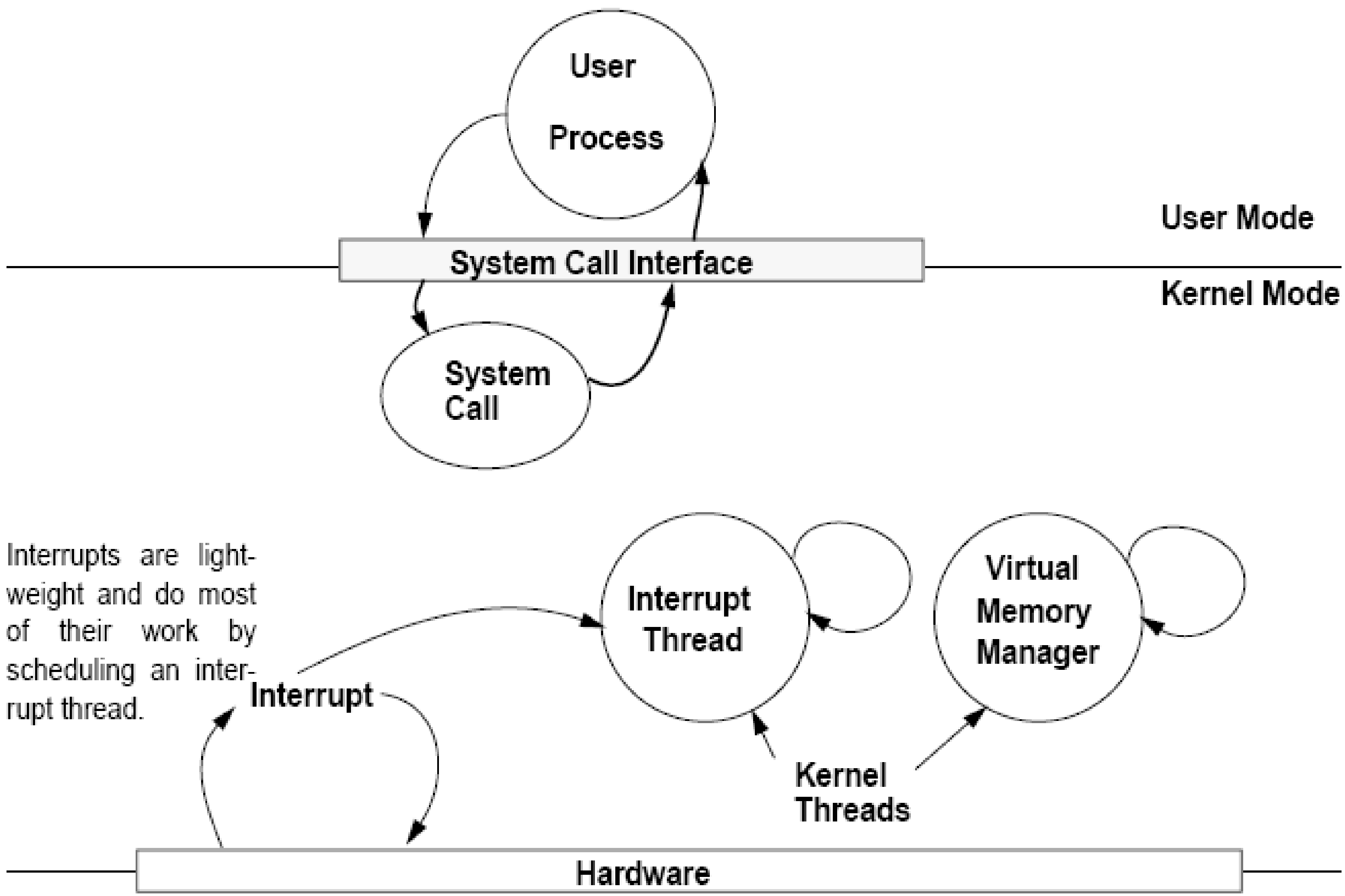
Execution Context

- Process Context
  - acts on behalf of the user process
  - access to the process's user area (*uarea*), and process structures for resource accounting
- Interrupt Context
  - Interrupt threads execute in an interrupt context
  - have their own stack and can access only kernel data structures
- Kernel Context
  - Kernel management threads run in the kernel context
  - share the kernel's environment with each other
  - typically cannot access process-related data
  - E.g. Page scanner

Threads in Kernel and Interrupt Context

- Interrupt Handlers
  - Kernel threads handle all but high-priority interrupts.
- Kernel Management Threads
  - kernel has own threads to carry out system management tasks
  - kernel management threads execute in the *kernel's* execution context
  - scheduled in the system (SYS) scheduling class at a higher priority than most other threads on the system.

Process, Interrupt, and Kernel Threads

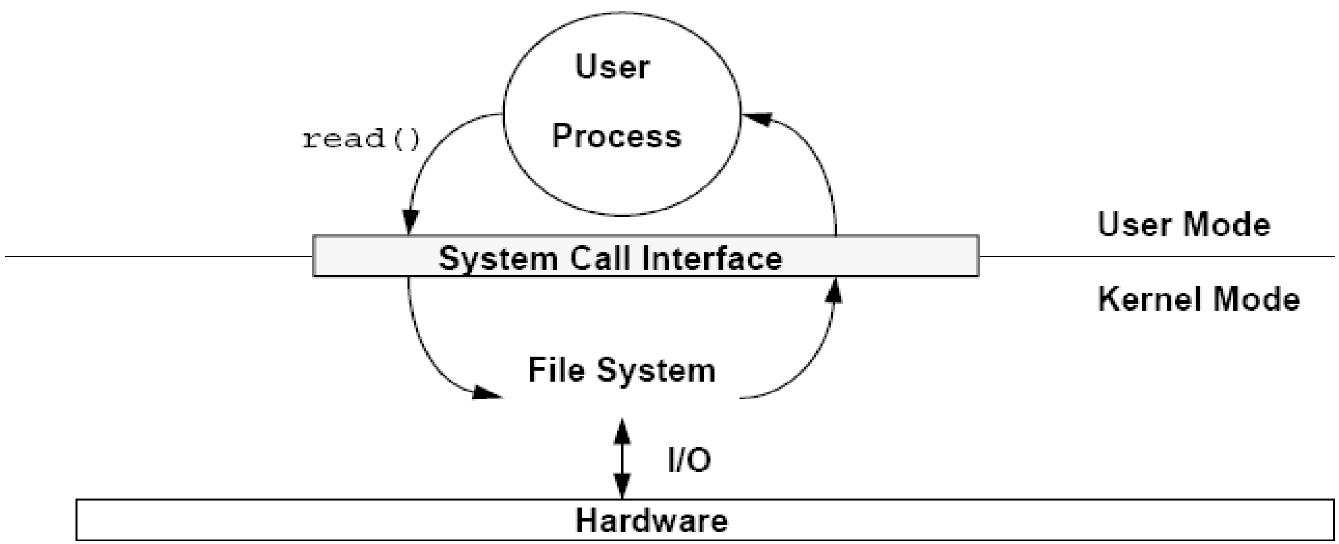


Outline

- Kernel Services
- system call
- Trap
- Interrupt
- kernel callout
- system clock

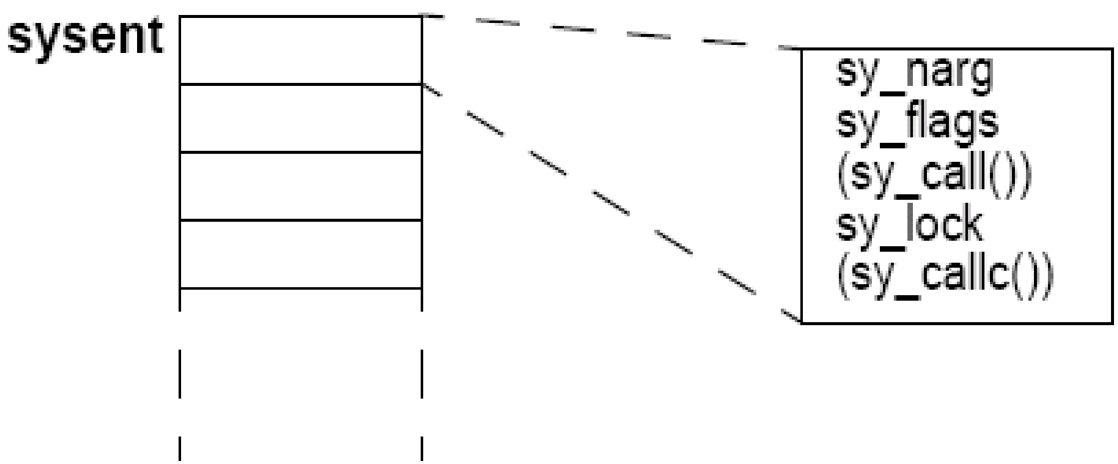
Enter the kernel mode by System Calls

- User processes/applications access kernel services through the system call facility
- Modes of execution (kernel & user) provide protection
- invocation of a system call causes the processor to change from user mode to kernel mode



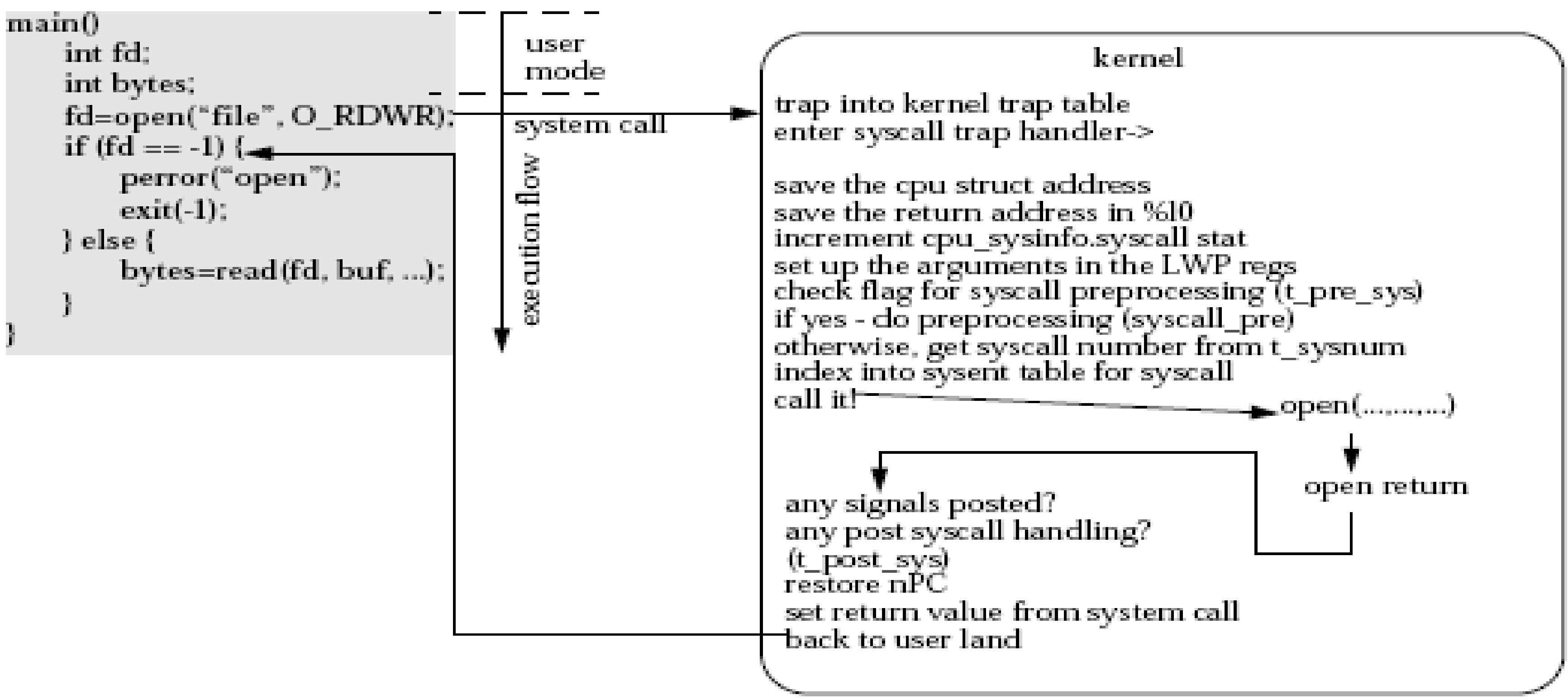
Regular System Calls

- kernel sysent table
  - contains an entry for every system call supported on the system
  - an array populated with sysent structures



Execution of System Calls

- results in the software issuing a trap instruction
- is executed on behalf of the calling thread



Fast Trap System Calls

- Solaris kernel's feature
- user processes can
  - jump into protected kernel mode
  - do minimal processing and thenreturn
  - without the overhead of saving all the state that a regular system call does
- only be used when the processing required in the kernel does not significantly interfere with registers and stacks.

Outline

- Kernel Services
- system call
- Trap
- Interrupt
- kernel callout
- system clock

UltraSPARC I & II Traps

- SPARC processor architecture uses traps as a unified mechanism to handle
  - system calls
  - processor exceptions
  - interrupts
- A SPARC trap is a procedure call as a result of
  - synchronous processor exception,
  - an asynchronous processor exception
  - a software-initiated trap instruction
  - a device interrupt

Processing of Traps

- hardware do
  - Save certain processor state
  - enters privileged mode
  - executing code in the corresponding trap table slot
- And go on
  - Execute trap handler for the type of trap
  - Once interrupt handler has finished, control is returned to the interrupted thread

UltraSPARC I & II Trap Types(1)

- **Processor resets**
  - Power-on reset, machine resets, software-initiated resets
- **Memory management exceptions**
  - MMU page faults, page protection violations, memory errors, misaligned accesses, etc.
- **Instruction exceptions**
  - Attempts to execute privileged instructions from nonprivileged mode, illegal instructions, etc.

UltraSPARC I & II Trap Types(2)

- **Floating-point exceptions**
  - Floating-point exceptions, floating-point mode instruction attempted when floating point unit disabled, etc.
- **SPARC register management**
  - Traps for SPARC register window spilling, filling, or cleaning.
- **Software-initiated traps**
  - Traps initiated by the SPARC trap instruction (Tcc); primarily used for system call entry in Solaris.

UltraSPARC I & II Trap Priority Levels

- Each UltraSPARC I & II trap has an associated priority level
- Highest-priority trap is taken first
  - 0 is the highest priority
- Interrupt traps are subject to trap priority precedence
  - compared against the *processor interrupt level* (PIL)

UltraSPARC I & II Trap Levels

- Nested traps
  - a trap can be received while another trap is being handled
- Nested traps have five levels
  - From trap level 0 (normal execution, no trap)
  - To trap level 4 (an error handling state and should not be reached during normal processing)

UltraSPARC I & II Trap Table Layout（1）

- UltraSPARC I & II trap table is halved
  - the lower half contains trap handlers for traps taken at trap level 0
  - the upper half contains handlers for traps taken when the trap level is 1 or greater
- Each half of the trap table is further divided into two sections
  - 256 hardware traps in the lower section
  - 256 software traps in the upper section (for the SPARC Tcc software trap instructions)

UltraSPARC I & II Trap Table Layout （ 2 ）

	Trap Table Contents	Trap Types
Trap Level = 0	Hardware Traps	000...07F
	Spill/Fill Traps	080...0FF
	Software Traps	100...17F
	Reserved	180...1FF
Trap Level > 0	Hardware Traps	000...07F
	Spill/Fill Traps	080...0FF
	Software Traps	100...17F
	Reserved	180...1FF

Software Traps

- Software traps are initiated by the SPARC trap instruction, Tcc.
- used primarily for system calls in the Solaris kernel
- three software traps for system calls
  - native system calls
  - 32-bit system calls (when 32-bit applications are run on a 64-bit kernel)
  - SunOS 4.x binary compatibility system calls
- several ultra-fast system calls implemented as their own trap

UltraSPARC Software Traps

Trap Definition	Trap Type Value	Priority
Trap instruction (SunOS 4.x syscalls)	100	16
Trap instruction (user breakpoints)	101	16
Trap instruction (divide by zero)	102	16
Trap instruction (flush windows)	103	16
Trap instruction (clean windows)	104	16
Trap instruction (do unaligned references)	106	16
Trap instruction (32-bit system call)	108	16
Trap instruction (set trap0)	109	16
Trap instructions (user traps)	110 – 123	16
Trap instructions (get_hrtime)	124	16
Trap instructions (get_hrvtime)	125	16
Trap instructions (self_xcall)	126	16
Trap instructions (get_hrestime)	127	16
Trap instructions (trace)	130-137	16
Trap instructions (64-bit system call)	140	16

A Utility for Trap Analysis

- Trapstat
  - dynamically monitors trap activity
  - analyze the traps taken on each processor installed in the system

```
# trapstat 3
vct  name                |    cpu0    cpu1
---------------------+--------------
 24  cleanwin            |    3636    4285
 41  level-1             |      99       1
 45  level-5             |       1       0
 46  level-6             |      60       0
 47  level-7             |      23       0
 4a  level-10            |     100       0
 4d  level-13            |      31      67
 4e  level-14            |     100       0
 60  int-vec             |     161      90
 64  itlb-miss           |    5329   11128
 68  dtlb-miss           |   39130   82077
 6c  dtlb-prot           |       3       2
 84  spill-1-normal      |    1210     992
 8c  spill-3-normal      |     136     286
 98  spill-6-normal      |    5752   20286
 a4  spill-1-other       |     476    1116
 ac  spill-3-other       |    4782    9010
 c4  fill-1-normal       |    1218     752
 cc  fill-3-normal       |    3725    7972
 d8  fill-6-normal       |    5576   20273
103  flush-wins          |      31       0
108  syscall-32          |    2809    3813
124  getts               |    1009    2523
127  gethrtime           |    1004     477
---------------------+--------------
ttl                      |   76401  165150
```

Outline

- Kernel Services
- system call
- Trap
- Interrupt
- kernel callout
- system clock

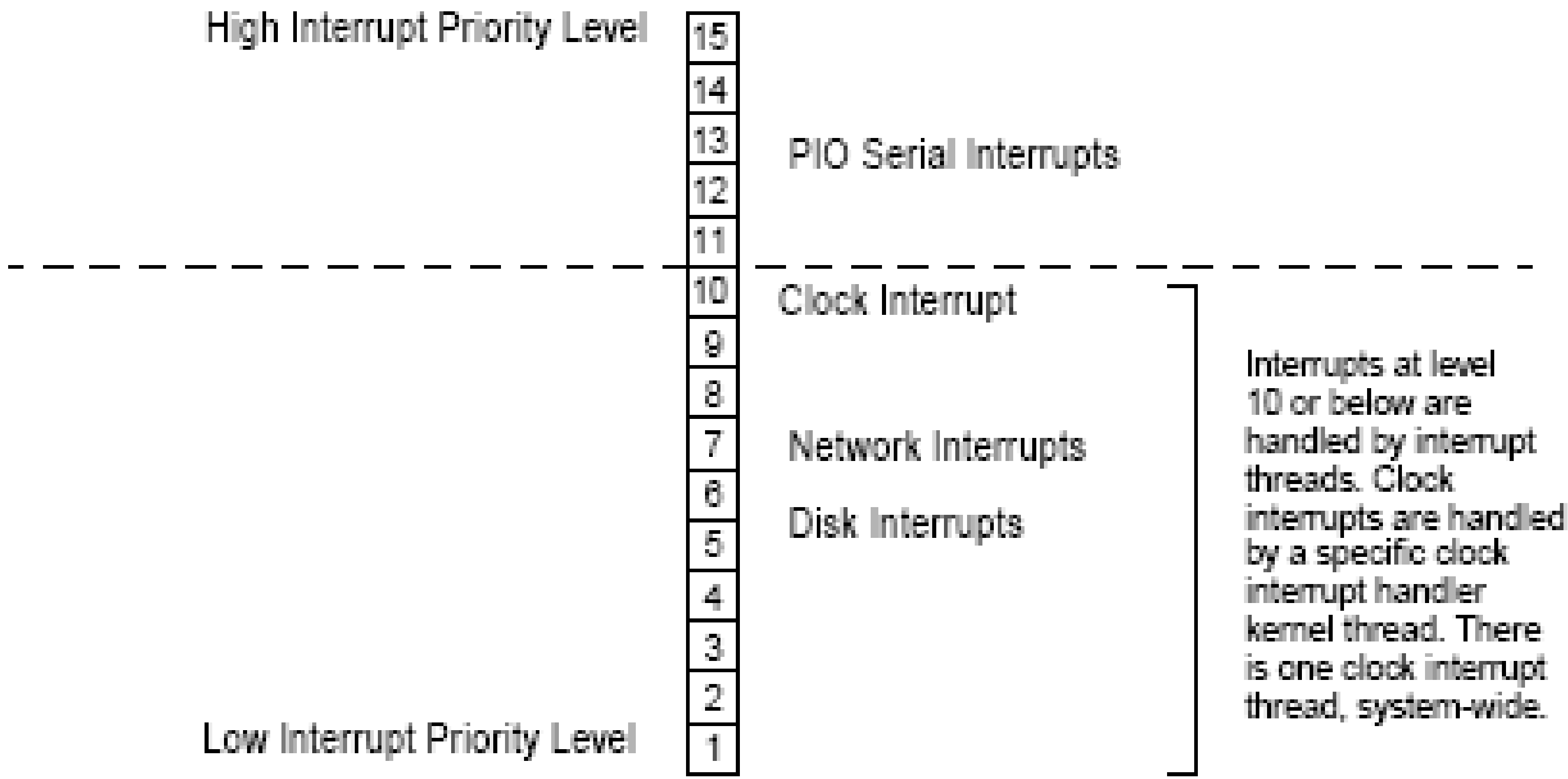
Interrupts

- An asynchronous event, not associated with the currently executing instruction
- Like traps
  - interrupts result in a vectored transfer of control to a specific routine
    - a device interrupt handler (part of the device driver).
  - interrupts are hardware architecture specific
- Interrupts can be “hardware” or “software”
  - “Hard”ware interrupts generated by I/O devices
  - Soft interrupts are established via a call to the kernel add_softintr() function

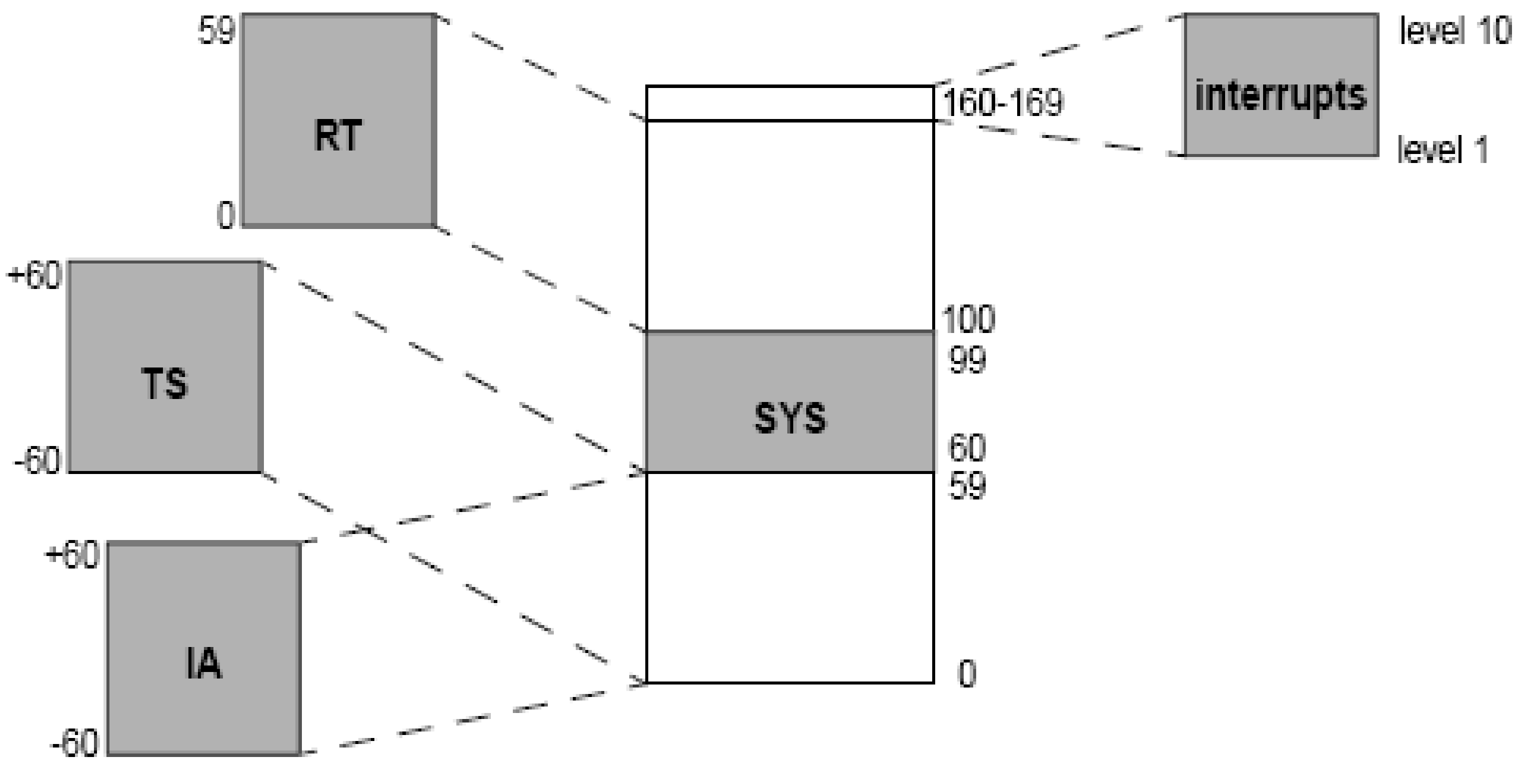
Interrupt priority

- based on interrupt level
  - higher levels have higher priority
- 15 (1-15) interrupt levels defined
  - Levels 1-9 are serviced by an interrupt thread linked to the processor that took the interrupt
  - Level 10 is the clock, and is handled by a dedicated clock_intr_thread
  - Levels 11-15 are handled in the context of the thread that was executing
    - these are considered high priority interrupts
- Dispatcher locks are held at level 11

Interrupt priority



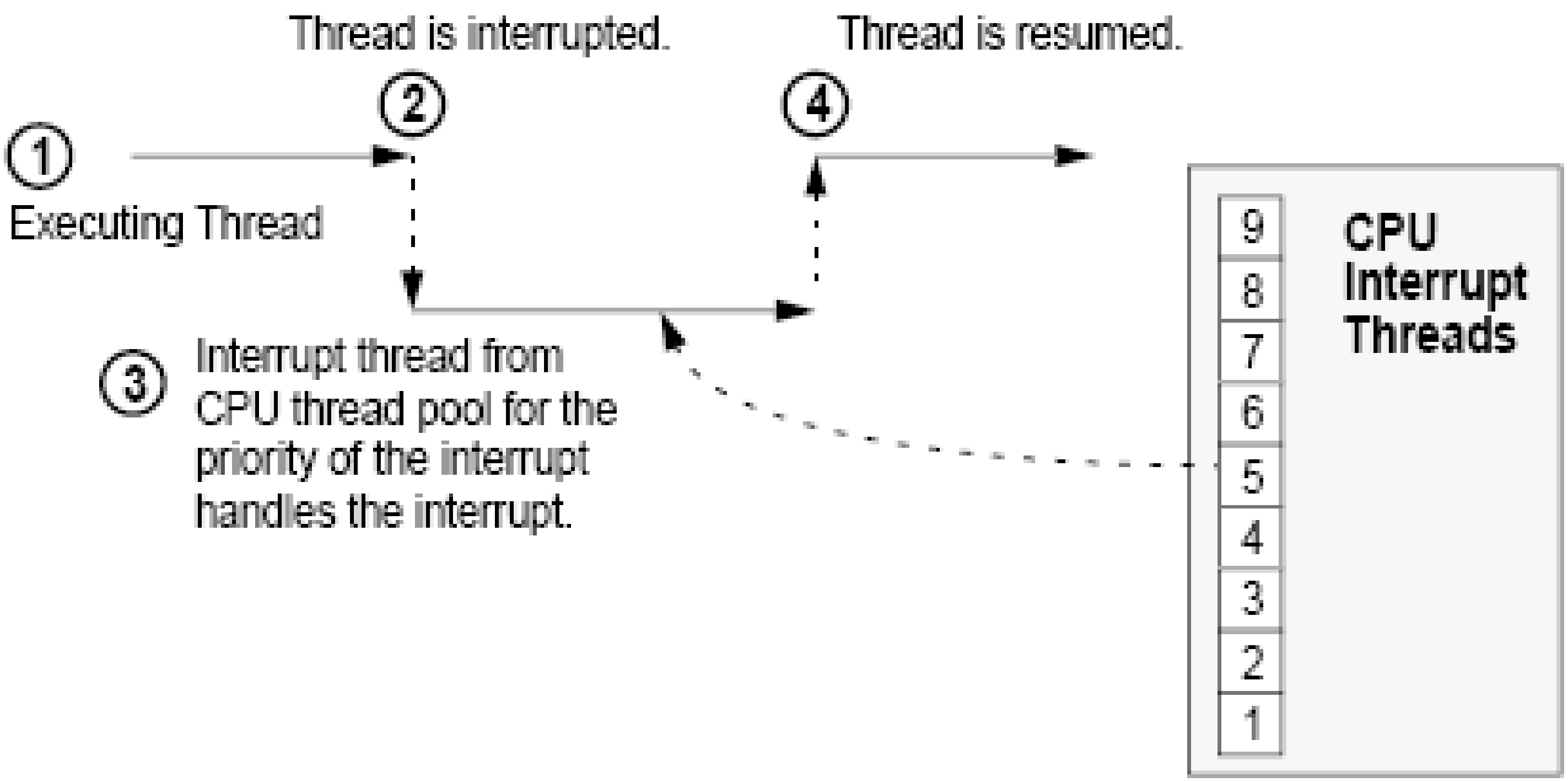
Interrupt Thread Priorities



interrupt threads

- When a CPU takes an interrupt, the currently running thread is “pinned” (not context switched out), some context is “borrowed”, and the interrupt thread runs
- If the interrupt thread completes
  - Simply unpin the pinned thread, and let it resume
- If the interrupt thread blocks
  - Must be upgraded to a “complete” thread, so it can block
    - This is the ithr column in mpstat
- Allow the pinned thread to resume

Handling Interrupts with Threads



opensolaris

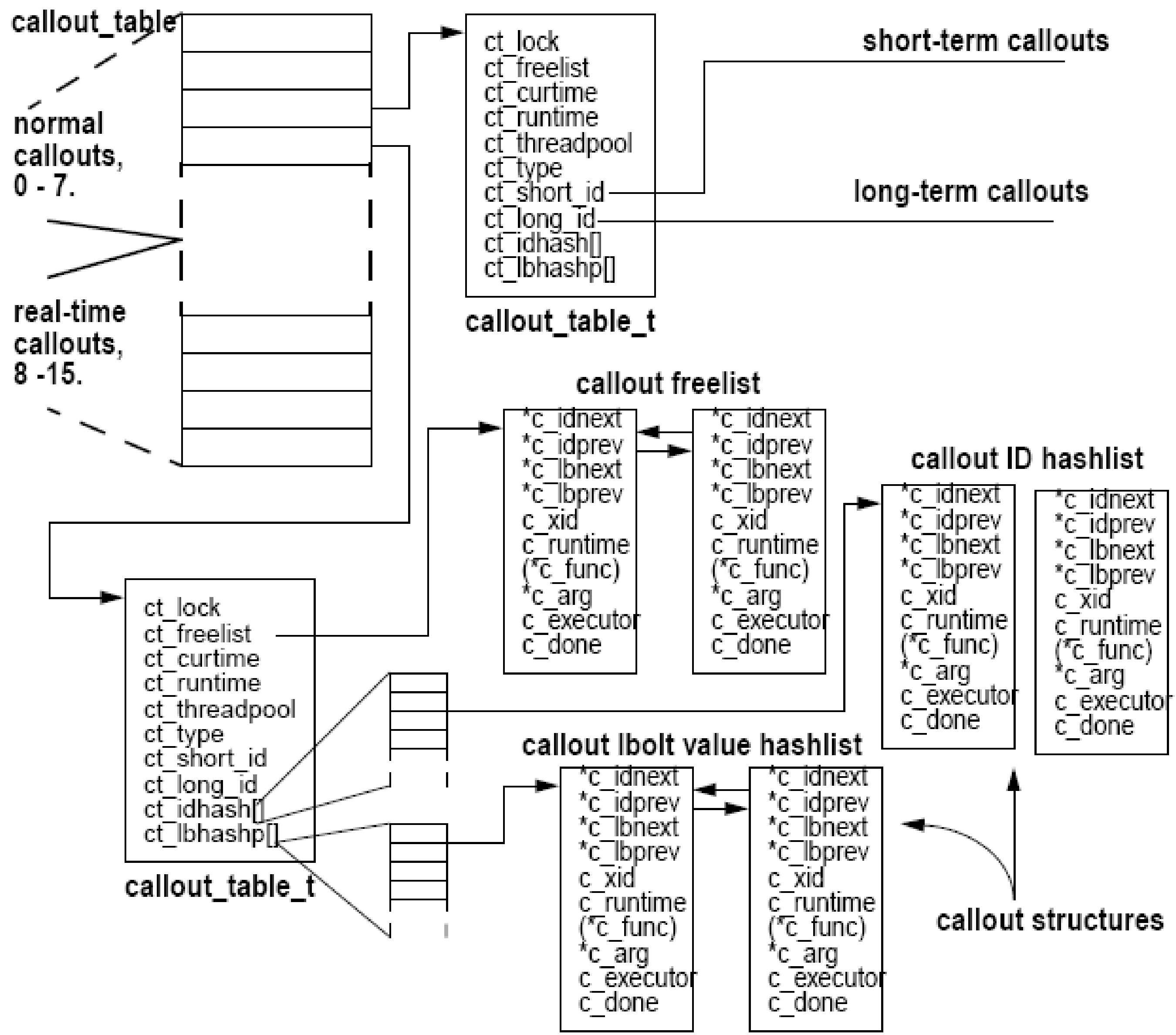
Outline

- Kernel Services
- system call
- Trap
- Interrupt
- kernel callout
- system clock

Kernel Callout

- general-purpose, time-based event scheduling
- kernel routines can place functions on the callout table through the timeout(9F) interface.
- With each clock interrupt, the tick value is tested and the function is executed when the time interval expires

Solaris 2.6 and 7 Callout Tables



Outline

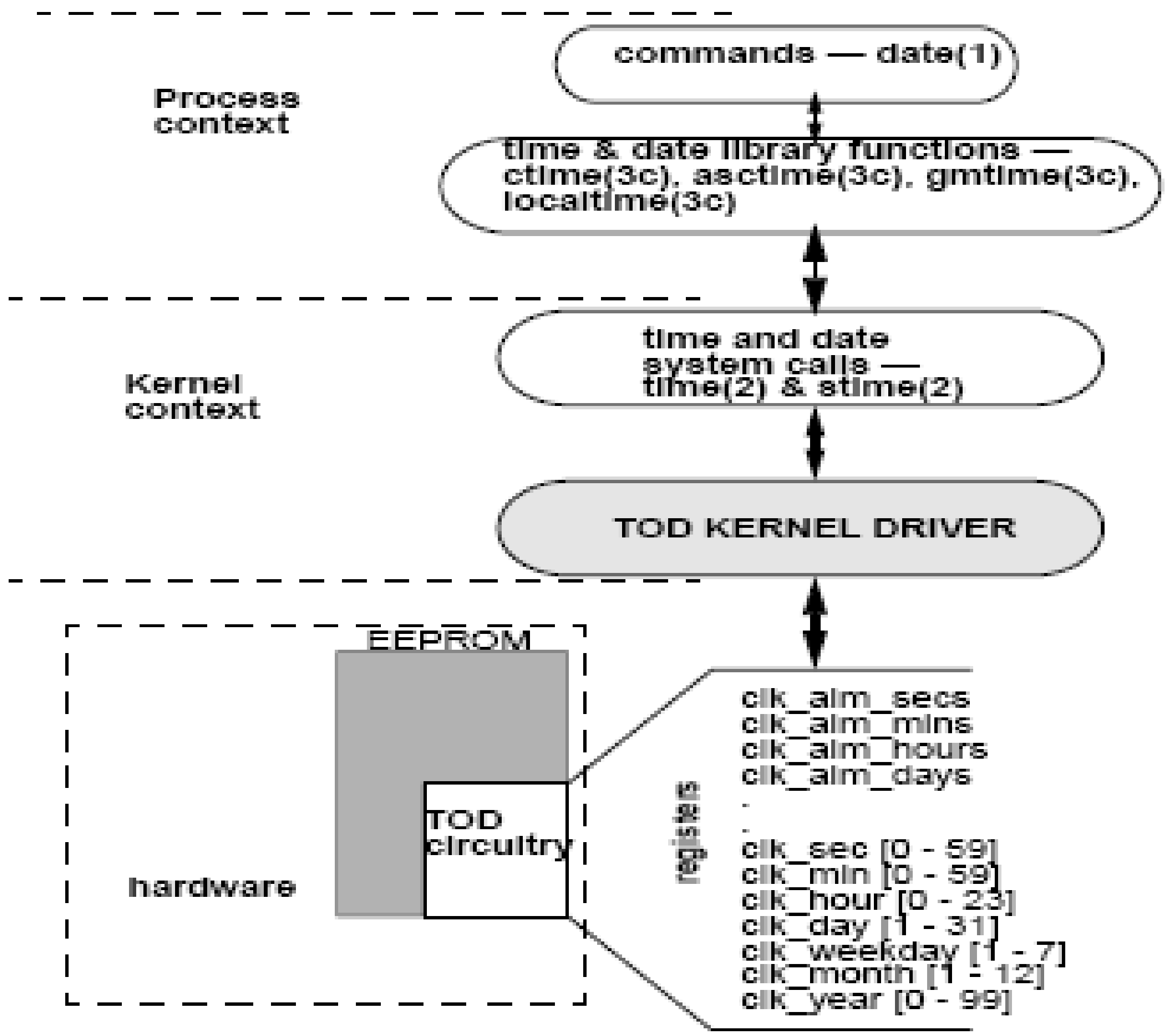
- Kernel Services
- system call
- Trap
- Interrupt
- kernel callout
- system clock

opensolaris

System Clocks

- All Sun systems implement a Time-Of-Day (TOD) clock chip that keeps time
- TOD clock circuitry is part of the system EEPROM
- TOD device driver implemented to read/write TOD -accessable as a device
- Clock interrupts generated 100 times a second - every 10 milliseconds
- Clock interrupt handler performs generic housekeeping functions

System Clocks



opensolaris

Clock interrupt handler

- Calculate free anon space
- Calculate freemem
- Calculate waitio
- Calculate usr, sys & idle for each cpu
- Do dispatcher tick processing
- Increment lbolt
- Check the callout queue
- Update vminfo stats
- Calculate runq and swapq sizes
- Run fsflush if it’s time
- Wake up the memory scheduler if necessary

High-Resolution Timer

- nanosecond-level timing functions
- internal gethrestime() (get high-resolution time) function
- System call API
  - setitimer(2)
    - support for real-time interval timers
  - gethrtime(3C)
    - provides programs with nanosecond-level granularity for timing
  - gethrtime(3C)
    - read the TICK register and return a normalized (converted to nanoseconds) value

Reference

- Solaris Internals-Core Kernel Components, Jim Mauro, Richard McDougall, Sun Microsystems Press, 2000
- SOLARIS Kernel Performance, Observability & Debugging, Richard McDougall, James Mauro,USENIX’05 ,2005
- Solaris Internals and Performance Management, Richard McDougall,2002

End

- Last.first@Sun.COM