
Real-time in the Solaris™ 8 Operating Environment



Solaris™ 8 Operating Environment Realtime

- Inherent part of system
- Advertised support of 300 μ s response time
- Powerful support for MP systems

Solaris Realtime Basics

- Fixed priority real-time processes
- User process priority manipulation
- High resolution timers
- Preemptive scheduling
- Deterministic and guaranteed dispatch latency
- Process priority inheritance
- Memory Management

Solaris Realtime Basics (Cont)

- Processor Sets
- Arbitrary granularity interval scheduling

Detailed Information

- Scheduling
- Interrupts
- Processor control
- Networking
- Timers
- Posix 1003.1b
- Posix 1003.1c

Traditional UNIX® Scheduling

- UNIX® scheduling originally designed to maximize system throughput in a multi-user time-sharing environment
- Priorities of resource intensive jobs lowered dynamically
- Antithesis of real-time

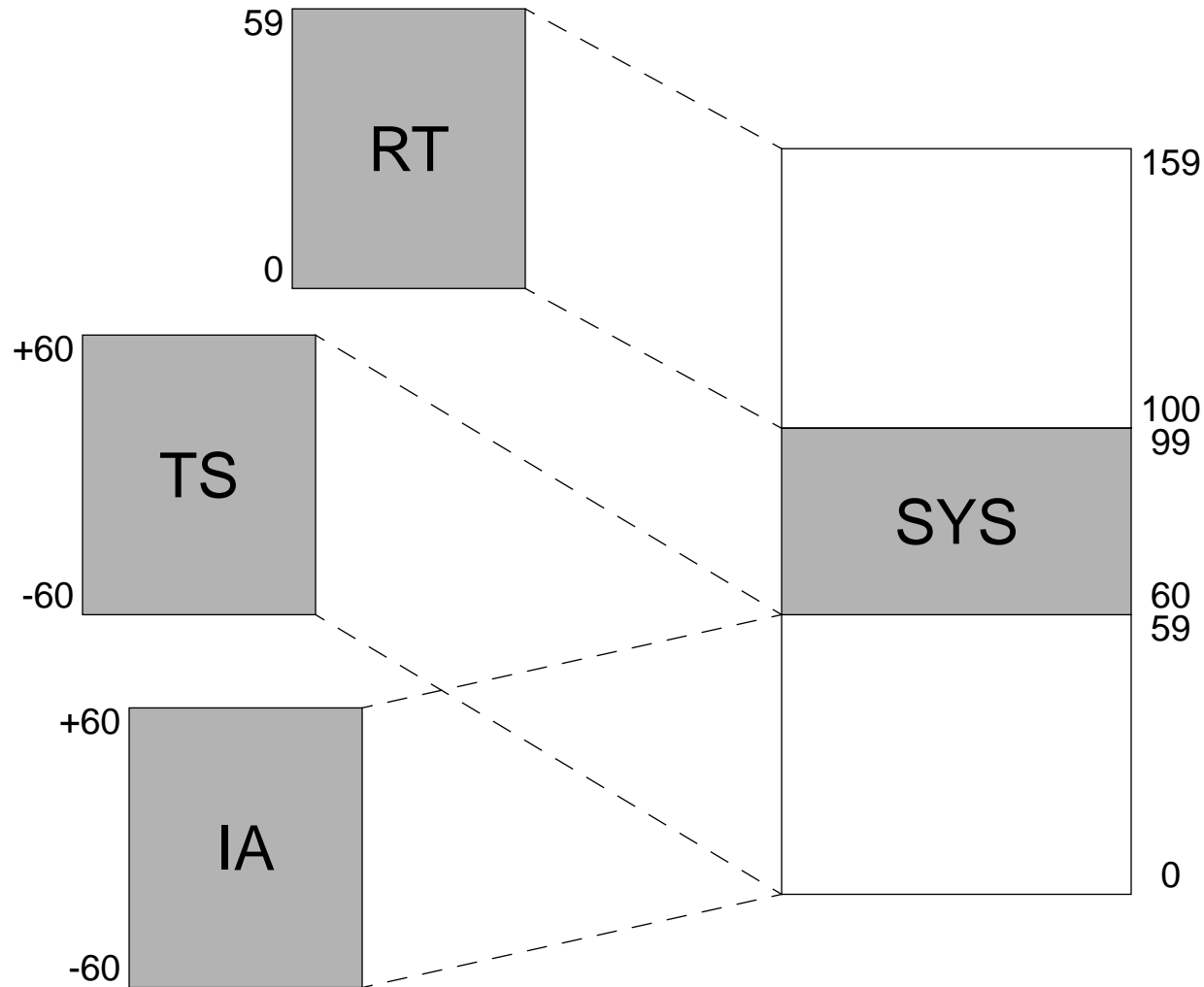
SVR4 Scheduling

- Scheduler redesigned in SVR4 to allow multiple *scheduling classes*
- Each priority within a scheduling class maps to a *global priority*
- Scheduling decisions are made based on global priority
- Traditional means of affecting priority (e.g. `nice(1)`) affect priority *within* a scheduling class
- Scheduling class and class-specific parameters are changed with `pricntl(1)` and `pricntl(2)`

Solaris Scheduling Classes

- Time-sharing (TS) provides traditional UNIX process scheduling, assuring fairness and maximizing throughput
- Interactive (IA) provides responsiveness for user interaction by favoring processes with keyboard focus
- System (SYS) provides fixed-priority scheduling for kernel threads (e.g. pageout, RPC service threads)
- Real-time (RT) provides fixed-priority preemptive scheduling for user processes

Default Global Priority Mappings



Real-time Scheduling Class

- Fixed priority with pure priority-preemption
- Different RT priorities have different time quanta (configurable via `dispadm(1M)`)
- Only super-user may put LWPs in the RT scheduling class (the LWPs themselves need not be super-user)
- LWPs in the RT scheduling class will never be swapped out (but they may incur pagefaults)
- The mapping of RT priority to global priority is configurable via `rt_dptbl(4)`

Priority Inversion

- If a high-priority thread blocks on a resource held by a low-priority thread, a runnable middle-priority thread can create a *priority inversion*
- For kernel synchronization primitives, *priority inheritance* eliminates priority inversions
- User-level priority inheritance was added in Solaris 8 (support for priority ceilings and priority inheritance for mutexes)

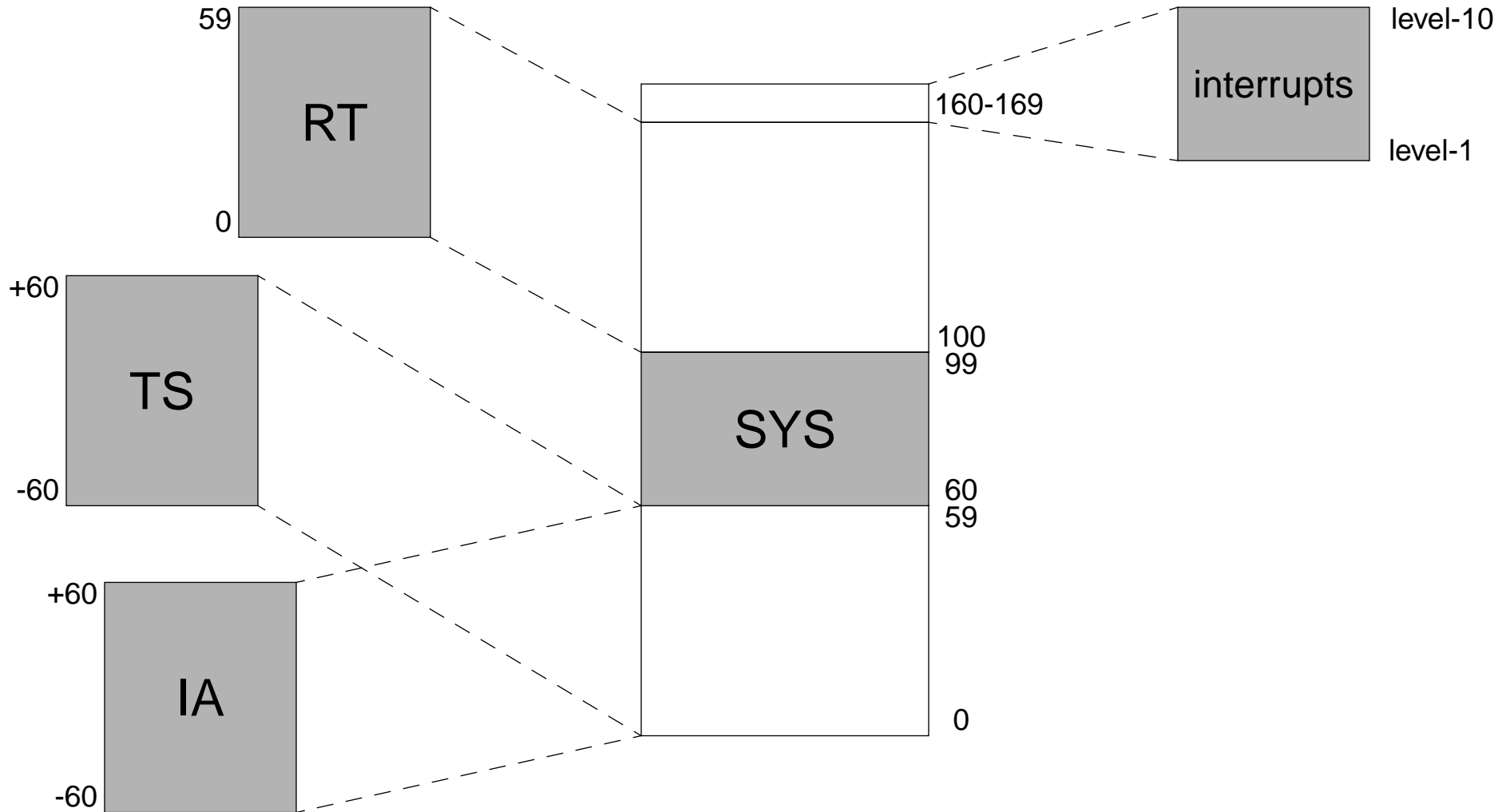
Interrupts

- Fifteen interrupt levels (level-1 through level-15)
- Interrupts at or below level-10 are handled as threads, and may thus block
- System clock executes at level-10
- Most devices interrupt below level-10
- Most device drivers can thus protect critical sections with synchronization primitives (i.e. without manipulating programmable interrupt level)

Interrupt Threads

- Interrupt threads are processor-bound
- When an interrupt thread at level- n blocks, the processor will run other threads, but will not lower programmable interrupt level below n
- Interrupt threads occupy the highest range of global priority, with level- n interrupt threads at a higher priority than level- $(n-1)$ threads

Interrupt Priority Mapping



Processor Control

- LWPs and processes may be bound to a processor with `processor_bind(2)` and `pbind(1M)`
- Binding does not preclude other, unbound, LWPs from running on the bound processor
- As of the Solaris™ 2.6 Operating Environment, processors may be grouped into a *processor set*

Processor Sets

- A processor set consists of some number of processors and some number of LWPs
- LWPs bound to the set will run only on the processors assigned to the set
- Processors assigned to the set will only run the LWPs bound to the set
- `pset_assign(2)` changes processor assignment;
`pset_bind(2)` changes LWP binding
- Sets can also be manipulated with `psrset(1M)`

Processor Set Caveats

- Only super-user may manipulate processor sets
- An LWP may only have one processor set binding
- An LWP may be bound to both a processor and a processor set only if the bound processor is assigned to the bound set
- Processor set binding inherited across `fork(2)`
- Interrupts may still be targeted at a processor assigned to a processor set

Interrupt Sheltering

- As of the Solaris™ 7 Operating Environment, a processor may be sheltered from unbound interrupts by putting it in the `P_NOINTR` state (see `p_online(2)`)
- The “-i” option to `psradm(1M)` puts the specified processor(s) in the `P_NOINTR` state
- The “-f” option to `psrset(1M)` puts all processors in the specified processor set in the `P_NOINTR` state
- At least one processor must be unsheltered (to service the system clock interrupt)

Interrupt Sheltering Caveats

- Bound interrupts (including cross calls) are unaffected by `P_NOINTR`
- On some platforms (e.g. SPARCServer™ 1000/2000), software has very little control over interrupt distribution

Address Space Control

- Page translations may be locked and unlocked with `mlock(3C)` and `munlock(3C)`
- All *currently* faulted translations may be locked by specifying the `MCL_CURRENT` flag to `mlockall(3C)`
- All *future* translations may be locked by specifying the `MCL_FUTURE` flag to `mlockall(3C)`
- No way to avoid the jitter induced by a minor fault; all translations should be faulted in and locked down *before* time-critical code is executed

Minimizing Jitter

- Run time-critical LWP in the RT scheduling class
- Bind LWP to a processor set containing `P_NOINTR` processors
- Lock all current and future translations
- Fault in all required text and data before executing time-critical code

Other Sources of Jitter

- By default, dynamic linking is done lazily
- Run-time binding for a dynamically linked function determined isn't determined until it's first called
- To avoid inducing jitter the first invocation of a dynamically linked function, set the environment variable `"LD_BIND_NOW"` to `"1"`

Real-time Networking

- All networking in SunOS 5.x is STREAMS-based
- STREAMS is *truly* the antithesis of real-time: at every opportunity, the framework forces the running thread to do everyone else's work
- STREAMS scheduling makes historical UNIX scheduling look hard real-time by comparison

Fixing STREAMS

- SunOS 5.8, improves the situation:
- High-priority threads (interrupt threads and, by default, threads in the RT scheduling class) will not be able to be hijacked by lower priority work

Real-Time - User level thread priority inheritance

- Realize support for `POSIX_THREAD_PRIO_INHERIT` and `POSIX_THREAD_PRIO_PROTECT` for mutexes of type `PTHREAD_PROCESS_SHARED` that are used by threads with system contention scope. It makes sense only for threads which are in the real-time scheduling class (i.e., whose scheduling policy is `SCHED_FIFO` or `SCHED_RR`).
- Adds an interface to support robustness for priority inheriting locks. If a process dies holding a robust mutex initialized with `_POSIX_THREAD_PRIO_INHERIT` the mutex is unlocked and the next application to acquire it will see `EOWNERDEAD`.
- This application can attempt to recover the state protected by the robust mutex and if it succeeds, calls `pthread_mutex_consistent_np` to clear the state of the mutex. At that point it unlocks the mutex.
- If it can't recover the state, it just unlocks the mutex. All attempts to lock the mutex after that will fail with the return code of `ENOTRECOVERABLE`. At that point the mutex must be destroyed and reinitialized to be used.

Timestamps

- Fast, nanosecond-resolution timestamps available via `gethrtime(3C)`
- `gethrtime(3C)` isn't correlated to the time-of-day, and is thus not subject to resetting or adjustment by way of `adjtime(2)` or `settimeofday(3C)`
- Applications requiring time-of-day timestamps should use `gettimeofday(3C)`
- `gethrtime(3C)` should *always* be used for any performance-related timestamps

Timers

- SunOS™ has always had BSD-style interval timers (`setitimer(2)`, `getitimer(2)`)
- As of SunOS™ 5.6, support for POSIX.1b timers (`timer_create(3RT)`, `timer_settime(3RT)`) was added
- POSIX timers based on clock type `CLOCK_REALTIME`, BSD timers, and `poll(2)` timeouts are dispatched by the system clock
- By default, system clock executes at 100 hertz (yielding a 10 millisecond timer granularity)

Configurable hz

- As of SunOS™ 5.6, the system clock rate is configurable
- Setting “`hires_tick`” to “1” in `/etc/system`, sets `hz` to 1,000 (yielding 1 millisecond timer granularity)
- A higher `hz` value (and thus a finer timer granularity) is possible by explicitly tuning `hz` in `/etc/system`
- Be careful; if `hz` is too high, the system will run nothing but the clock code

Shortcomings of Configurable hz

- Cranking the system clock has undesirable performance side-effects
- For some customers, a 1 millisecond timer resolution is *still* too coarse
- This is a common complaint when the application requires an interval timer rate which does not divide the system clock rate (e.g. 60 hertz)

CLOCK_HIGHRES

- In the Solaris™ 8 Operating Environment, we have added a new POSIX clock type (in addition to **CLOCK_REALTIME**): **CLOCK_HIGHRES**
- **CLOCK_HIGHRES** timers don't rely on the system clock for dispatch; they deal directly with the hardware timer source
- Timer resolution is as fine-grained as the platform can provide (up to 1 nanosecond resolution)
- **clock_getres(3RT)** can be used to determine the platform's **CLOCK_HIGHRES** resolution

CLOCK_HIGHRES Details

- **CLOCK_HIGHRES** timers use a *per-processor* timer source
- The actual hardware timer source used is transparent to the application, and “may change” over the lifetime of the timer
- In practice, the processor binding of a **CLOCK_HIGHRES** timer shadows that of its creating LWP
- If a processor is assigned to a processor set, *only* **CLOCK_HIGHRES** timers for LWPs bound to the set will use the processor’s timer source

On UltraSPARC™...

- **CLOCK_HIGHRES** is driven with the per-processor **TICK_COMPARE** register
- **CLOCK_HIGHRES** resolution on UltraSPARC™ is thus the reciprocal of the processor clock rate
- For example, a 250 MHz UltraSPARC™ has a **CLOCK_HIGHRES** granularity of 4 nanoseconds

On Other Platforms...

- Hardware timer source is interval-based
- Interval used for both `CLOCK_HIGHRES` timers and the system clock is “hz”
- Setting “`hires_tick`” is still important for real-time on non-UltraSPARC platforms

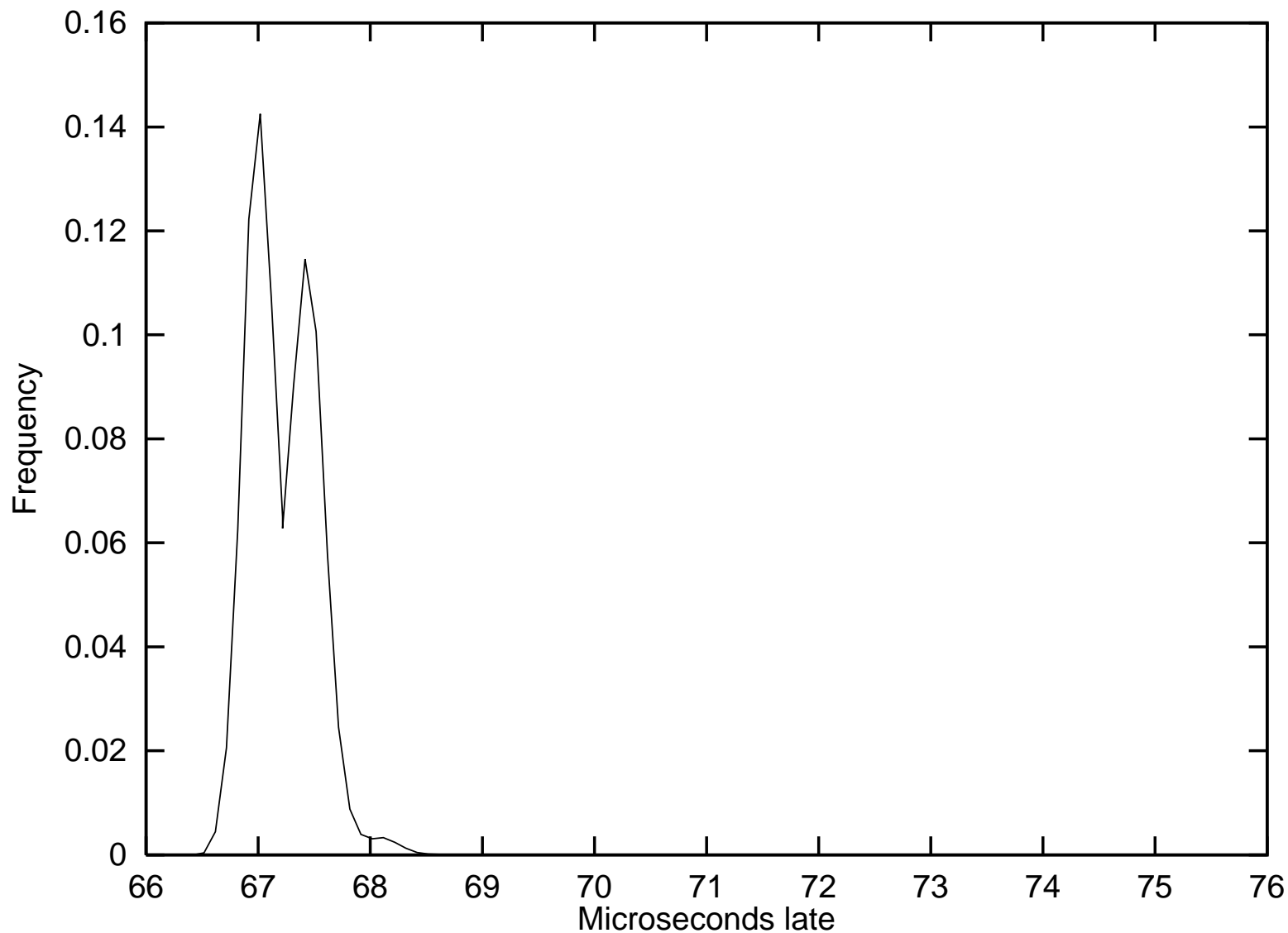
CLOCK_HIGHRES Caveat

- On platforms providing extremely fine-grained **CLOCK_HIGHRES** resolution (i.e. UltraSPARC), timers with small intervals will effect high interrupt rates
- Sufficiently small intervals will wedge the processor
- **CLOCK_HIGHRES** timers may only be created by root
- ...unless the tunable “**clock_highres_nonroot**” is set to a non-zero value in **/etc/system**

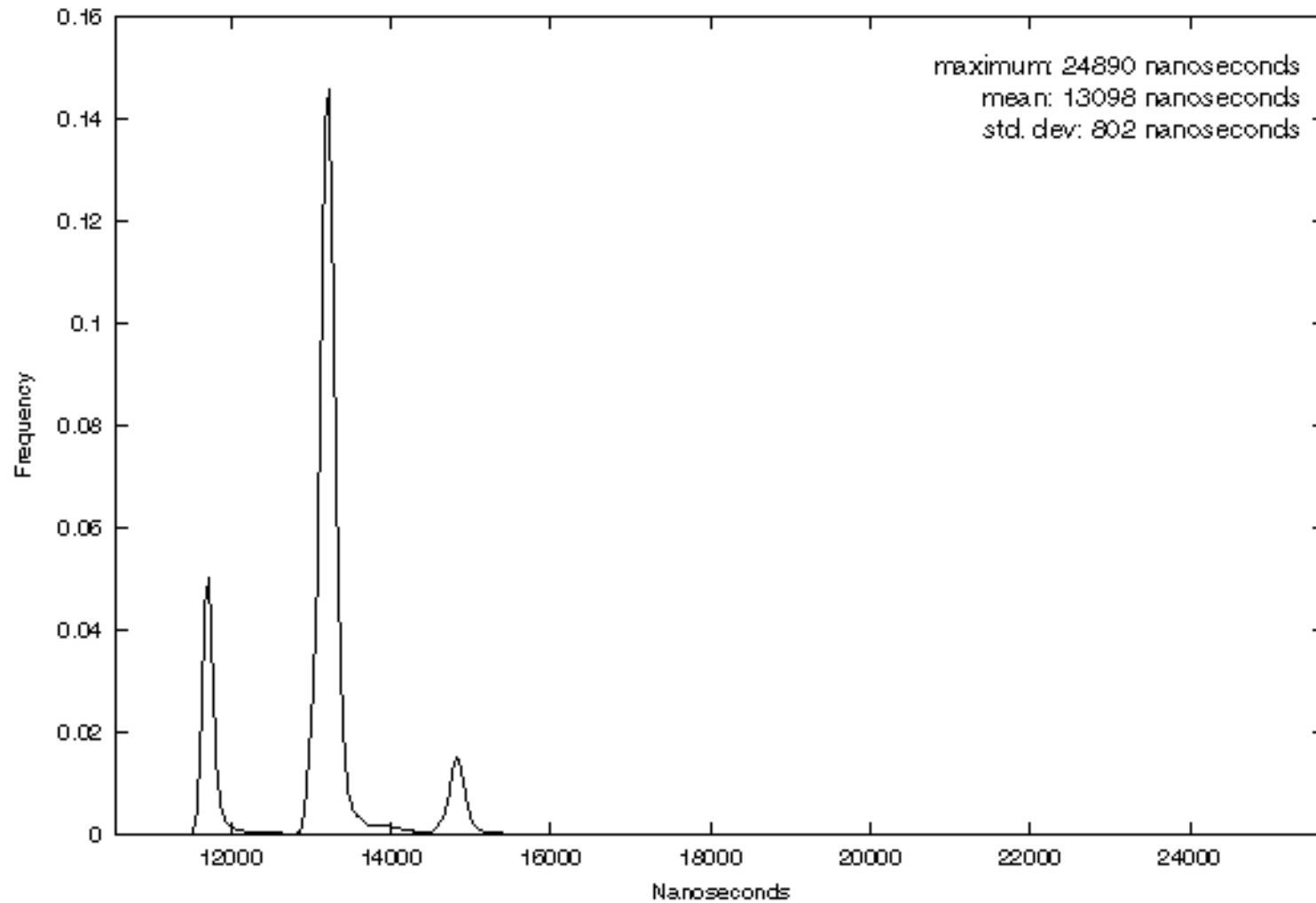
CLOCK_HIGHRES and P_NOINTR

- **CLOCK_HIGHRES** timers for bound LWPs are considered to be “bound interrupts”, and are therefore unaffected by **P_NOINTR**
- Extremely low interval timer jitter for **CLOCK_HIGHRES** timers bound to a processor set containing only **P_NOINTR** processors

CLOCK_HIGHRES Jitter



Total response time Solaris 5.8 (Generic) - (2x400 Ultra-250)



POSIX 1003.1b Real-time Extensions

- The definition of real-time used in defining the scope of the POSIX 1003.1b standard is:

“Real-time in operating systems: the ability of the operating system to provide a required level of service in a bounded response time.”

- The key elements defining the scope are:
 1. Defining a sufficient set of functionality to cover a significant part of the real-time application program domain, and
 2. Defining sufficient performance constraints and performance related functions to allow a realtime application to achieve deterministic response from the system.

POSIX 1003.1b Real-time Extensions in Solaris

POSIX 1003.1b Real-time Extensions feature	POSIX feature in Solaris 2.6	Solaris specific feature available	Notes
Asynchronous I/O	YES	YES	aioread/write(3), aiocancel(3), ...
Mapped Files	YES		mmap(2), munmap(2)
Memory Locking	YES		mlock(3C), munlock(3C), memcntl(2)
Range of Memory Locking	YES		mlockall(3C), munlockall(3C)
Memory Protection	YES		mprotect(2)
Message Passing	YES	YES	IPC Message Queues
Prioritized I/O	NO		I/O priority based on process priority
Priority Scheduling	YES		above plus a "nice" value
Real-time Signals	YES		SIGRTMIN=37, SIGRTMAX=44
Semaphores	YES	YES	IPC Semaphores
File Synchronization	YES		fsync(3C), fdatasync(3R)
Shared Memory Objects	YES	YES	IPC Shared Memory
Synchronized I/O	YES		msync(3C), fcntl(2), ...
Timers	YES	YES	setitimer(2), getitimer(2)

POSIX 1003.1c Thread Extensions

- Provides interfaces and functionality to support multiple flows of control, called *threads*, within a process.
- The key elements defining the scope are:
 1. Defining a sufficient set of functionality to support multiple threads of control within a process
 2. Defining a sufficient set of functionality to support the realtime application domain
 3. Defining sufficient performance constraints and performance related functions to allow a realtime application to achieve deterministic response from the system
- An implementation can claim to conform to P1003.1c if:

it provides the library bindings for the interfaces and actually supports functionality sufficient to allow definition of the `_POSIX_THREADS` symbol. Additional functionality (e.g., `_POSIX_THREAD_PRIORITY_SCHEDULING`) is optional.

POSIX 1003.1c Thread Extensions in Solaris

POSIX 1003.1c Thread Extensions feature	Solaris 2.6	Solaris 7	Solaris 8
_POSIX_THREADS	YES	YES	YES
_POSIX_THREAD_ATTR_STACKSIZE	YES	YES	YES
_POSIX_THREAD_ATTR_STACKADDR	YES	YES	YES
_POSIX_THREAD_PRIORITY_SCHEDULING	NO	YES	YES
_POSIX_THREAD_PRIO_INHERIT	NO	NO	YES
_POSIX_THREAD_PRIO_PROTECT	NO	NO	YES
_POSIX_THREAD_PROCESS_SHARED	YES	YES	YES
_POSIX_THREAD_SAFE_FUNCTIONS	YES	YES	YES

Conclusions

- SunOS™ 5.x has become increasingly appropriate for real-time applications
- Focus is on MP real-time
- The Solaris™ 8 Operating Environment adds some compelling real-time features not found in other operating systems (real-time or otherwise)

Sun, Sun Microsystems, the Sun Logo, Solaris, Ultra, and SunOS are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All SPARC trademarks are used under license and are registered trademarks of SPARC International, Inc. in the United States and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc. Unix is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd.