Kernel Thread Implementation

- based on paper “Beyond Multiprocessing… Multithreading the SunOS Kernel” by Eykholt et al.

- paper is report on actual kernel threads implementation, not many new ideas, but good to know technical details.

SunOS 5.0 kernel

SunOS 5.0 kernel is the main operating system component for Solaris 2.0

- kernel consists of kernel threads – lightweight, inexpensive switching, fully preemtible
- process model changed from heavyweight to lightweight processes (LWPs)
- LWPs map one-to-one to kernel threads
- kernel threads that are only used internally don’t need to be associated with LWPs (system threads)
- entire kernel is multithreaded, even interrupts

Data structures

data for processor, process, LWP and kernel thread

- per-process structure
  - list of kernel threads associated with process
  - pointer to process address space
  - user credentials
  - list of signal handlers
- per-LWP structure (swappable):
  - user-level registers
  - system call arguments
  - signal handling masks
  - resource usage information
  - profiling pointers
  - pointer to kernel thread and process structure

Data structures

- kernel thread structure
  - kernel registers
  - scheduling class
  - dispatch queue links
  - pointer to the stack
  - pointers to LWP, process, and CPU structure
- CPU structure:
  - pointer to currently executing thread
  - pointer to idle thread
  - current dispatching and interrupt handling info
  - for the most part architecture independent
- for efficient access to structures, hold pointer to current thread in global register
  - enables access to struct fields with single instruction

Kernel Scheduling

- Support for multiple scheduling classes
  - system, timesharing, real-time (fixed-priority)
  - dispatcher chooses thread with greatest global priority to run on a CPU
- Preemption – kernel is fully preemtible
  - if a higher priority thread becomes runnable, it is scheduled *as soon as possible* in the place of a lower priority thread
  - important for real-time and interrupt threads
    - e.g. user code run by real-time thread can have sufficiently high priority to preempt other threads, i.e. preemption in user-level libraries…

Synchronization

- Standard abstractions are supported: mutexes (non-recursive), condition variables, semaphores, readers/writer locks
- mutexes and writer locks support priority inheritance
- specific primitives can be used to interrupt blocking with a signal
- mutex default blocking policy
  - spin while owner of the lock is running on a different processor
  - sleep if the owner ceases to run
- sync variables occupy minimal space
  - contain a two byte index, used to find a structure that contains the header for the queue of suspended objects and priority inheritance information

Interrupts as threads

- mutexes when spinning need to raise interrupt level above that of interrupt handlers that use mutexes
  - otherwise?
  - this is expensive, and makes interrupt coding inconvenient
- Solution: interrupts (up to a certain level) are threaded as asynchronously created and dispatched high-priority threads
  - mutexes get interrupted, but the result is the creation of an interrupt thread
  - interrupt threads can sleep and can synchronize
  - interrupts disabled in some places (putting a thread to sleep, if it needs to be awaken, during dispatching)

Implementing interrupts as threads

- for efficiency there are preallocated, partially initialized interrupt threads
- initially (after interrupt occurs), interrupt threads are not separate from interrupted thread, which is pinned, and cannot move to another CPU
- if the interrupt thread blocks, then it becomes a full-fledged thread, separate from interrupted thread, and can be descheduled.
- CPU structure keeps track of interrupt level of blocked interrupt thread, so lower level interrupts are ignored

Implementing interrupts as threads

- Overall cost
  - overhead of 40 SPARC instructions per interrupt
  - saving of 12 instructions per mutex (no changes in interrupt level, etc.)
  - space per pre-allocated interrupt thread (~8K per thread per CPU, 9 interrupt levels on SPARC, plus clock)
- One clock interrupt thread per system – it is not disabled, just queued, in case it is delayed by higher level interrupts