- Project 1 questions?

- More on Process Management & Context Switching…
  - traditional PCB
  - multi-level (before Solaris 9): PCB, LWP, kernel and user threads (handout)
  - Linux task structure

Traditional PCB

- Process ID
- user credentials
- runtime info
  - priority, signal mask, registers, kernel stack, resource limitations…
- signal handlers table
- memory map
- file descriptors

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

- Linux threads and processes represented via task struct
  - collection of pointers to corresponding content
  - threads from same process share FS, MM, signal handlers, open files… (see Silberschatz 4.5.2)
    - kernel code takes care of locking if necessary for shared access
  - all tasks linked into lists, but also accessable via hash on PID
  - if architecture permits (PPC, UltraSparc) `current` pointer in register

- User level threads
  - PC, SP
  - registers
  - stack
  - tread private/specific data
    - e.g., for `errno`
- Memory footprint/access critical for performance.

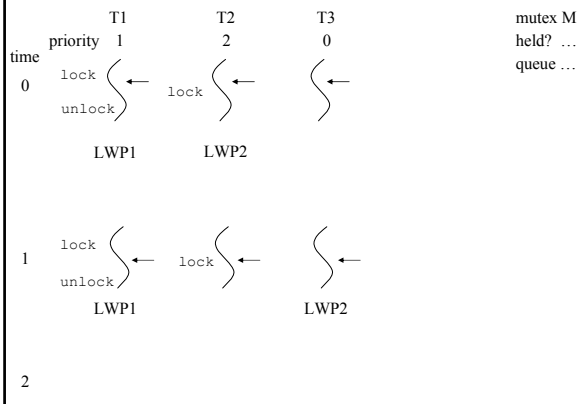
Process Contention Scope Context Switching

There are four ways to cause a running thread to context switch:

- Synchronization
  - the most common way: the thread goes to sleep on a mutex or condition variable
- Pre-emption
  - a running thread does something that causes a high-priority thread to become runnable
  - hard to implement entirely in user-space (in SMPs) except in a one-to-one model
- Yielding
  - a thread may explicitly yield to another thread of the same priority
- Time-slicing
  - threads of the same priority may be context switched periodically

PCS Contexts Switching

- Time slicing and pre-emption need kernel collaboration (except for pre-emption on uniprocessors)
  - at the very least a signal needs to be sent and/or handled
- *Question:* What happens when the library context switches threads?



- Context switch is necessary and nowadays efficient
- Key factor is warm cache
  - architecture support may help
  - “processor affinity” or “cache affinity”