CAS CS 552
Intro to Operating Systems

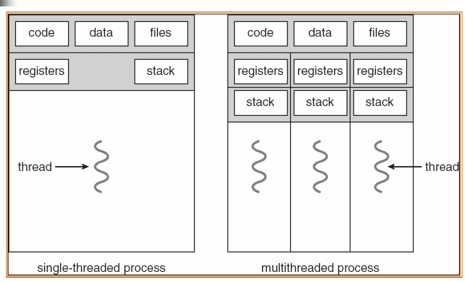
Richard West

Threads

Threads

- All processes have at least one "thread of control" (i.e., execution path through their address space)
- A multi-threaded process has multiple threads of control within its address space
- Each thread has its own program counter value and stack

Single and Multithreaded Processes



Thread Resources

- Threads share the code segment of the process address space with other threads executing in the same address space
  - Also share global variables (in data segment), open files, possibly signals…

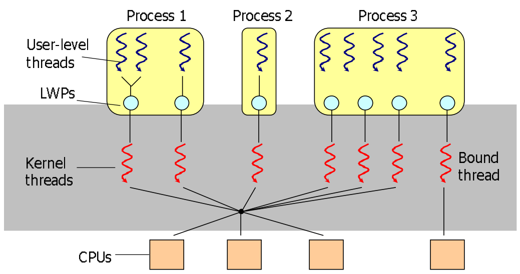
Thread Context Switching

- Cheaper than process switching – Why?
  - What state needs to be saved/restored?
  - Unlike process switching, no heavyweight memory management operations needed to change address spaces
    - Possibly involving bringing a process into main memory after previously being swapped to disk
    - Possibly involving a TLB flush/reload!

User-Level Threads

- e.g., POSIX Pthreads with "process scope"
- User-level threads are somewhat independent of the OS
  - Context-switch overheads are very low since no trap to the kernel is necessary
  - Switching between threads is usually cooperative, with one thread explicitly yielding to another via a call to a library routine
  - Problems with user-level threads?
    - OS scheduling, blocking issues

Example Solaris 2.x Two-Level Thread Model



Solaris 2.x Example (Continued)

- User-level threads are MUXed onto LWPs
- LWPs each have a corresponding "kernel thread"
- OS schedules kernel threads for execution on CPUs
- User-level threads may be scheduled/switched amongst LWPs by a thread library without kernel intervention
- A blocked kernel thread (e.g., awaiting I/O completion) will yield CPU to another kernel thread
  - Blocking a kernel thread, blocks the LWP and attached user-level thread also
  - The kernel may schedule another kernel thread associated with another LWP in the same process, if one exists

Addition Information

- The following slides are taken, more or less, from Silberschatz et al and are provided as background information…

User Threads

- Thread management done by user-level threads library
- Three primary thread libraries:
  - POSIX Pthreads
  - Win32 threads
  - Java threads

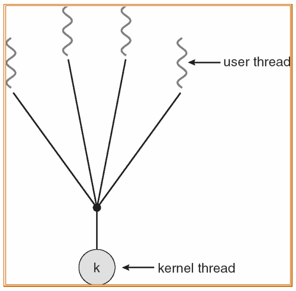
Kernel Threads

- Supported by the Kernel
- Examples
  - Windows XP/2000
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

Many-to-One

- Many user-level threads mapped to single kernel thread
- Examples:
  - Solaris Green Threads
  - GNU Portable Threads

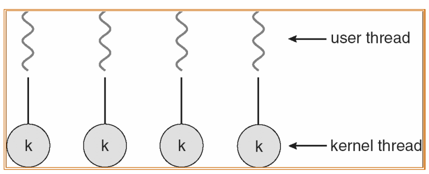
Many-to-One Model



One-to-One

- Each user-level thread maps to kernel thread
- Examples
  - Windows NT/XP/2000
  - Linux
  - Solaris 9 and later

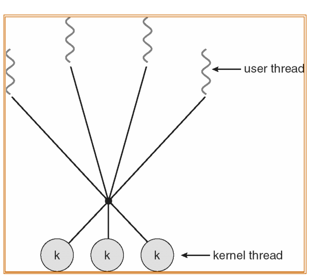
One-to-one Model



Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Solaris prior to version 9
- Windows NT/2000 with the *ThreadFiber* package

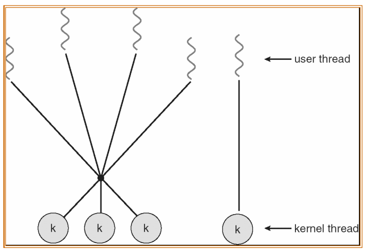
Many-to-Many Model



Two-level Model

- Similar to M:M, except that it allows a user thread to be **bound** to kernel thread
- Examples
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 and earlier

Two-level Model



Threading Issues

- Semantics of **fork()** and **exec()** system calls
- Thread cancellation
- Signal handling
- Thread pools
- Thread specific data
- Scheduler activations

Semantics of fork() and exec()

- Does **fork()** duplicate only the calling thread or all threads?

Thread Cancellation

- Terminating a thread before it has finished
- Two general approaches:
  - **Asynchronous cancellation** terminates the target thread immediately
  - **Deferred cancellation** allows the target thread to periodically check if it should be cancelled

Signal Handling

- Signals are used in UNIX systems to notify a process that a particular event has occurred
- A **signal handler** is used to process signals
  1. Signal is generated by particular event
  2. Signal is delivered to a process
  3. Signal is handled
- Options:
  - Deliver the signal to the thread to which the signal applies
  - Deliver the signal to every thread in the process
  - Deliver the signal to certain threads in the process
  - Assign a specific thread to receive all signals for the process

Thread Pools

- Create a number of threads in a pool where they await work
- Advantages:
  - Usually slightly faster to service a request with an existing thread than create a new thread
  - Allows the number of threads in the application(s) to be bound to the size of the pool

Thread Specific Data

- Allows each thread to have its own copy of data
- Useful when you do not have control over the thread creation process (i.e., when using a thread pool)

Scheduler Activations

- Both M:M and Two-level models require communication to maintain the appropriate number of kernel threads allocated to the application
- Scheduler activations provide **upcalls** - a communication mechanism from the kernel to the thread library
- This communication allows an application to maintain the correct number kernel threads

Pthreads

- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems (Solaris, Linux, Mac OS X)

Windows XP Threads

- Implements the one-to-one mapping
- Each thread contains
  - A thread id
  - Register set
  - Separate user and kernel stacks
  - Private data storage area
- The register set, stacks, and private storage area are known as the **context** of the threads
- The primary data structures of a thread include:
  - ETHREAD (executive thread block)
  - KTHREAD (kernel thread block)
  - TEB (thread environment block)

Linux Threads

- Linux refers to them as *tasks* rather than *threads*
- Thread creation is done through **clone()** system call
- **clone()** allows a child task to share the address space of the parent task (process)

Java Threads

- Java threads are managed by the JVM
- Java threads may be created by:
  - Extending Thread class
  - Implementing the Runnable interface

Java Thread States