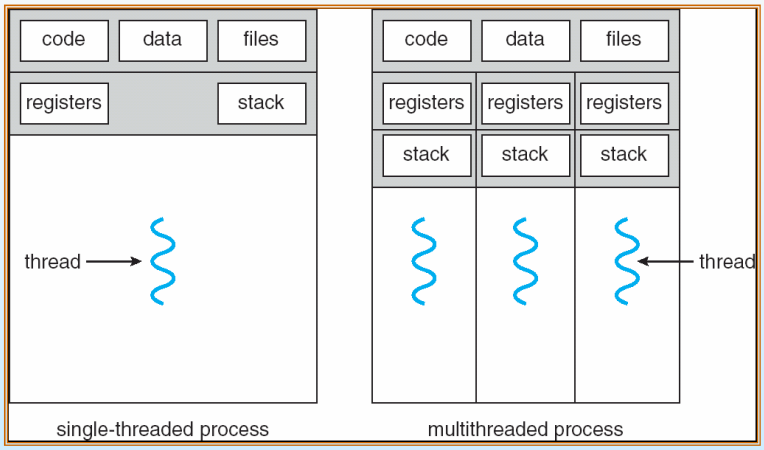
Chapter 4: Threads

Chapter 4: Threads

- Overview
- Multithreading Models
- Threading Issues
- Pthreads
- Windows XP Threads
- Linux Threads
- Java Threads

Single and Multithreaded Processes



Heavyweight process

Benefits

- Single application may perform several similar task: web server
- Responsiveness:
  - program may continue to run even if part of it is blocked
- Resource Sharing
  - Share the memory/resource of the process. Multiple threads within the same address space
- Economy
  - Allocate memory/resource is costly. Solaris: 20 times slower to create a process than a thread.Contexte switching is 5 times slower.
- Utilization of MP Architectures

Thread libraries

- Provide an API for creating & managing threads
- User space
  - No kernel support
  - **Invoking a function results in a local function calli n user space and not a system call**
- **Kernel space**
  - **Library supported directly by the kernel**
  - **Code & data structure are present in the kernel**
  - **API → system calll**

User Threads

- Thread management done by user-level threads library
  - Provide an API for creating & managing threads
- Three primary thread libraries:
  - POSIX Pthreads (kernel/user)
  - Win32 threads
  - Java threads (user)

Kernel Threads

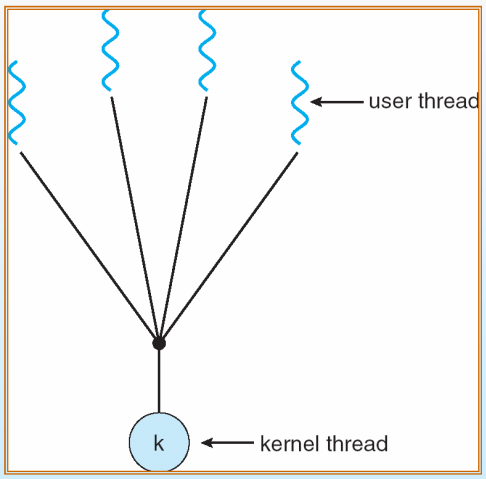
- Supported by the Kernel
- Examples
  - Windows XP/2000
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

Multithreading Models

- Ultimately there must exist a relationship between user threads and kernel thread
- Many-to-One
- One-to-One
- Many-to-Many

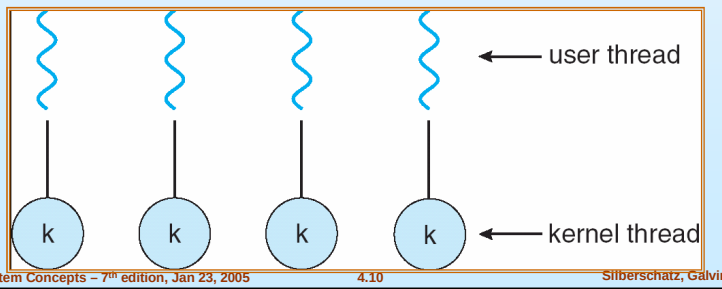
Many-to-One

- Many user-level threads mapped to single kernel thread
- Thread management done in user space →efficient
- The whole process will block if a thread makes a blocking system call
- Only one thread can access the kernel at a time → unable to run on multiprocessors
- Examples:
  - Solaris Green Threads
  - GNU Portable Threads



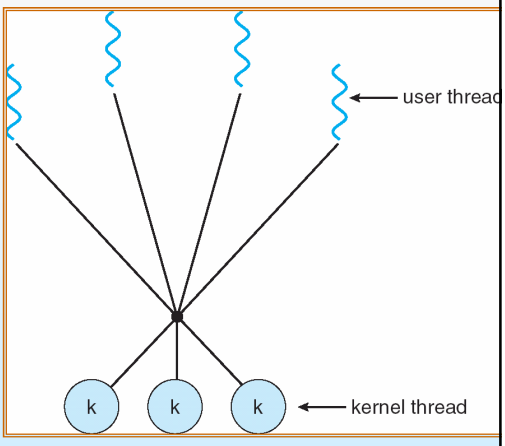
One-to-One

- Each user-level thread maps to kernel thread
- Examples
  - Windows NT/XP/2000
  - Linux
  - Solaris 9 and later
- More concurrency
- Parallel multiprocessors
- Creating user thread == creating a kernel process



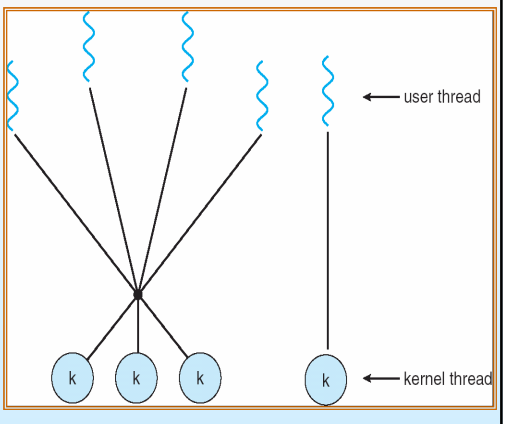
Many-to-Many Model

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



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



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()

- Semantic of **fork()** and **exec()** change in a multithread program
- Does **fork()** duplicate only the calling thread or all threads?
  - Two version of fork
    - Duplicates all threads
    - Duplicates only the calling thread
- Exec() will replace all the process – including all threads

Thread Cancellation

- Terminating a thread before it has finished
  - Data Base searching / stop button
- Two general approaches:
  - **Asynchronous cancellation** terminates the **target thread** immediately
  - **Deferred cancellation** allows the target thread to periodically check if it should be cancelled
- **Problem when resources have been allocated to a thread**

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
- Synchronous signal
  - Illegal memory access / division by 0
  - → delivered to the same process that performed the operation that cause the signal
- Asynchronous signal
  - Generated by an external event (Control-C)
- Every signal may be handled by
  - Default signal handler
  - User defined signal handler

Signal Handling

- 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
- Synchronous signal → delivered to the thread
- Asynchronous signal ? (not so clear)
  - Control-C should be delivered to all threads
- Thread should specify which signal it will accept / which it will block
- Signal needs to be handled only once → to the first thread that do not block it
- *Kill(aid_t aid, int signal)*
- *Pthread_kill(pthread_t tid, int signal)*

Thread Pools

- Create a number of threads in a pool where they await work
  - Web server
- 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

- **Threads belonging to a process share the data of the process**
- 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

- Communication issue between the kernel and the user space
- 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)

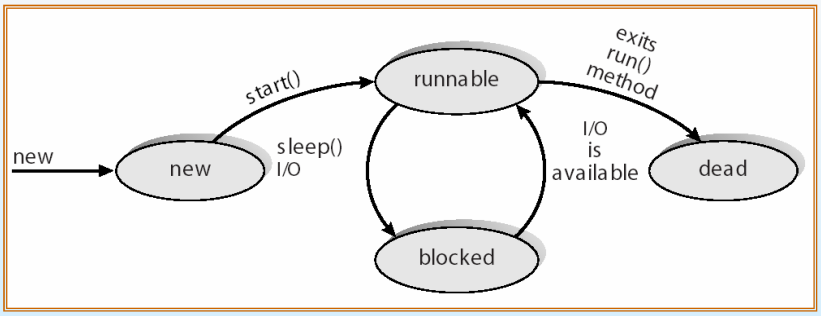
Linux Threads

- 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



End of Chapter 4