北京邮电大学软件学院
School Of software Engineering Of BUPT

Operating Systems

Lecture 3 Threads

Jinpengchen
Email: jpchen@bupt.edu.cn

Catalog Description

- Overview
- Multithreading Models
- Threading Issues

Why Processes?

- 自从60年代提出进程概念以来，在操作系统中一直都是以进程作为独立运行的基本单位，直到80年代中期，人们又提出了更小的能独立运行的基本单位—线程。

Why Processes?

- [案例]编写一个MP3播放软件，核心功能模块有三个：
  - 从MP3音频文件中读取数据；
  - 对数据进行解压缩；
  - 把解压缩后的音频数据播放出来。

Why Processes?

- 单进程的实现方法：

```
        Main()
        {
        While(true)
I/O     {read();
CPU     decompress();
        Play();
        }
        }
        read(){...};
        decompress(){...};
        Play(){...};
```

问题：

- ✓ 播放出来的声音是否连贯？
- ✓ 各个函数之间不是并发执行，影响资源的使用效率？

Youtube的状态栏

Why Processes?

- 多进程的实现方法：

程序1

```
Main()
{
While(true)
{read();
}
}
read(){…};
```

程序2

```
Main()
{
While(true)
{decompress();
}
}
decompress(){…};
```

程序3

```
Main()
{
While(true)
{Play();
}
}
Play(){…};
```

Why Processes?

- 怎么解决这些问题：

  需要提出一种新的实体，满足以下特性：
  实体之间可以并发地执行；
  实体之间共享相同的地址空间；

  这种实体就是：线程（Thread）

Thread concept overview

- What is the thread?
  - Thread:

    ✓ A sequential execution stream within a process

    ✓ a thread of execution

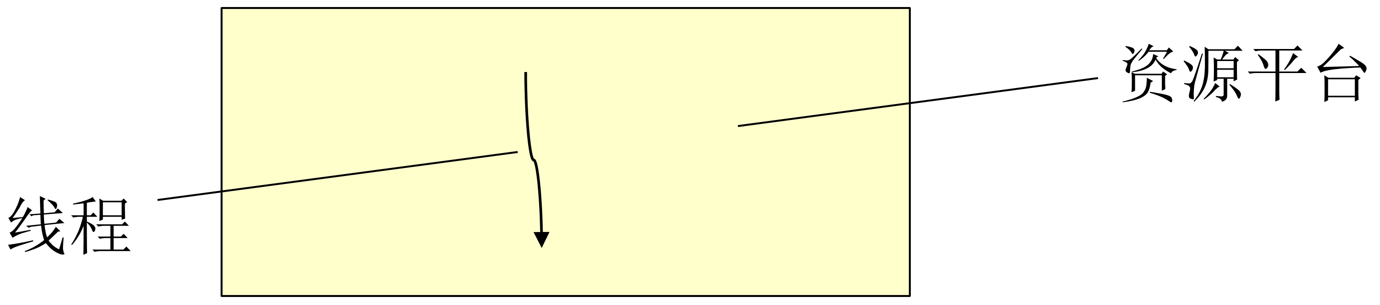
    ✓ 进程当中的一条执行流程

Thread concept overview

- A thread is a basic unit of CPU utilization;
  - it comprises a thread ID, a program counter, a register set, and a stack.
  - it shares with other threads belonging to the same process, the code section, the data section, and other OS resources, such as open files, signals, etc
- A traditional process has a single thread of control: heavyweight process.

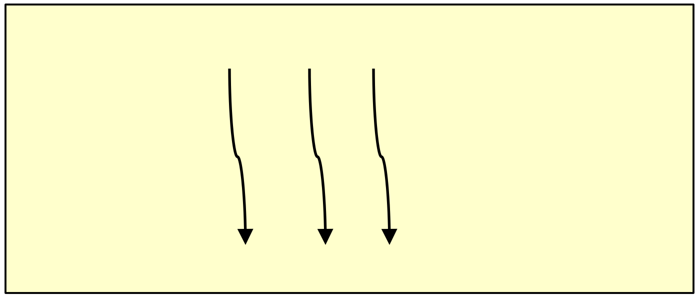
Thread concept overview

- 从两个方面来理解进程
  - 从资源组合的角度—进程把一组相关的资源组合起来，构成一个资源平台（环境），包括地址空间（代码段、数据段）、打开的文件等各种资源
  - 从运行的角度：代码在这个平台上的一条执行流程（线程）



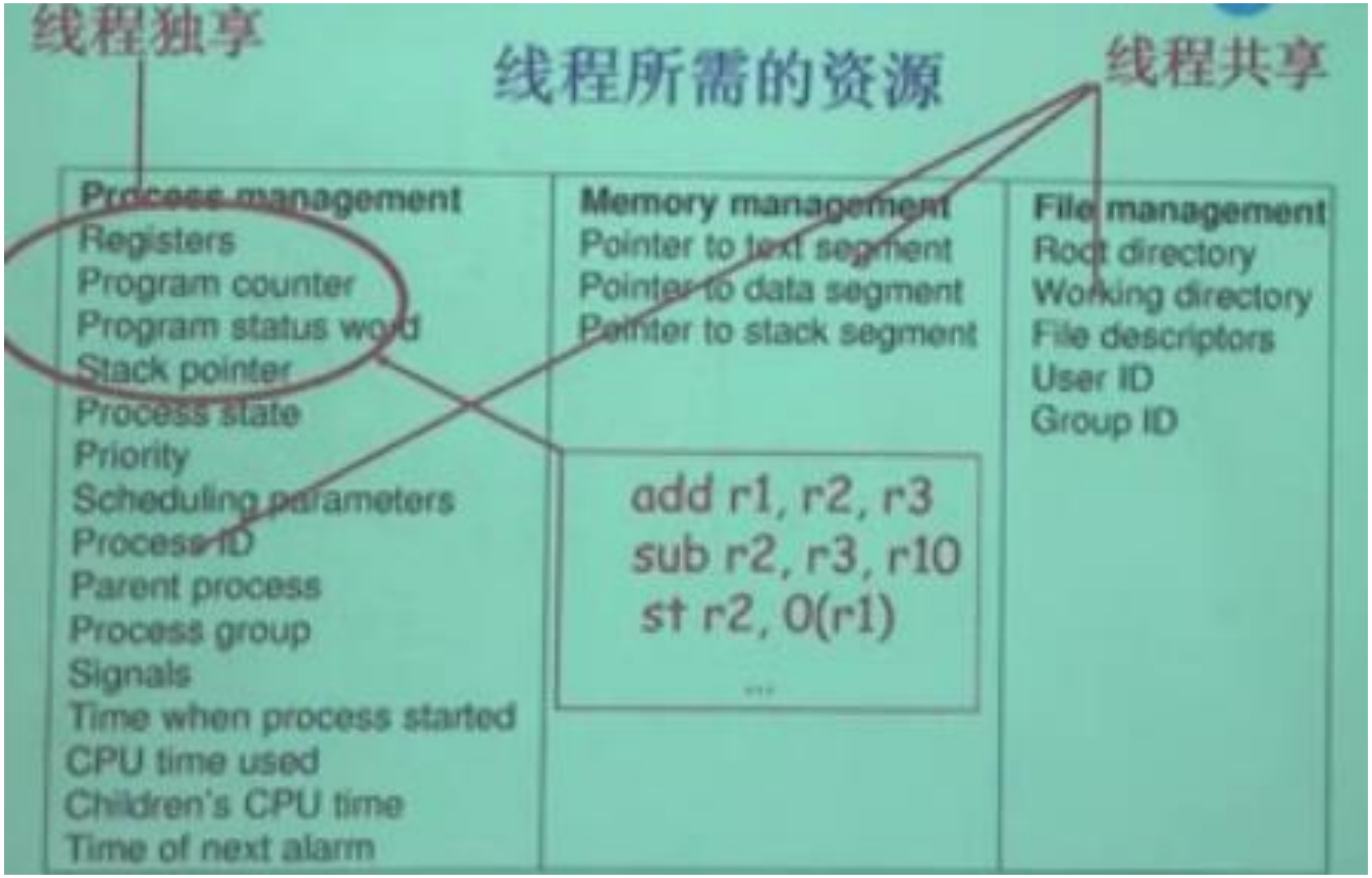
Thread concept overview

- 进程 = 线程 + 资源平台
  - 优点：
  - ✓ 一个进程中可以同时存在多个线程
  - ✓ 各个线程之间可以并发地执行
  - ✓ 各个线程之间可以共享地址空间



Thread concept overview

- Single and Multithreaded Processes

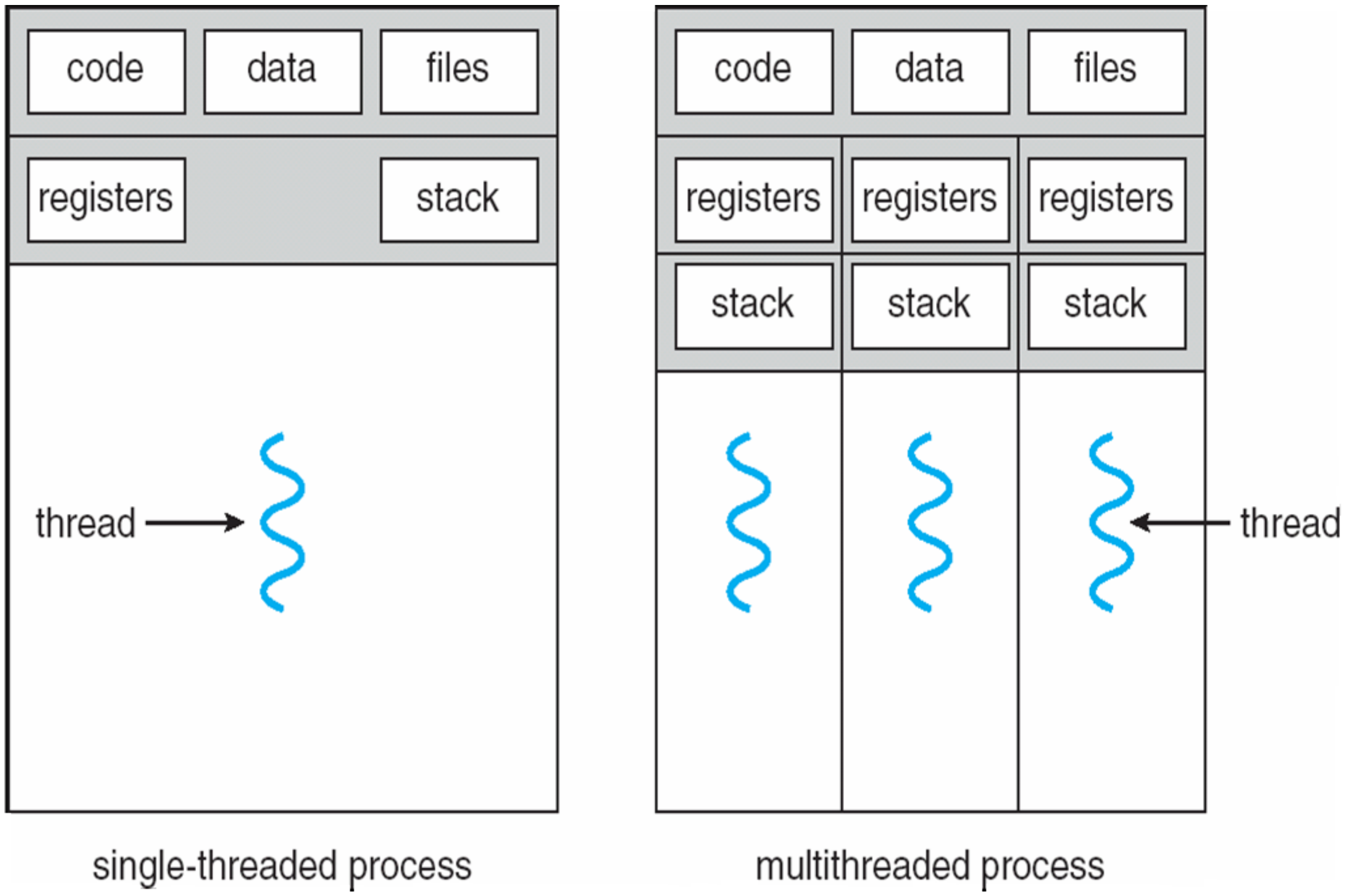

线程独享
线程所需的资源
线程共享

Process management	Memory management	File management
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

```
add r1, r2, r3
sub r2, r3, r10
st r2, 0(r1)
...
```

Thread concept overview

- Single and Multithreaded Processes



Thread concept overview

- 线程与进程的比较
  - 进程是资源分配单位,线程是CPU分配单位
  - 进程拥有一个完整的资源平台，而线程值独享必不可少的资源，如寄存器和栈
  - 线程同样具有就绪、阻塞和执行三种状态，同样具有状态之间的转换关系
  - 线程能减少并发执行的时间和空间开销
  - 线程=轻量级进程（lightweight process）

Thread concept overview

- Motivation
  - On modern desktop PC, many APPs are multithreaded.
  - a separate process with several threads
  - Example 1: A web browser
    - ✓ one for displaying images or text;
    - ✓ another for retrieving data from network
  - Example 2: A word processor
    - ✓ one for displaying graphics;
    - ✓ another for responding to keystrokes from the user;
    - ✓ and a third for performing spelling & grammar checking in the background

Thread concept overview

- Motivation
  - In certain situations, a single application may be required to perform several similar tasks. Example: a web server
  - Allow a server to service several concurrent requests. Example: an RPC server and Java's RMI systems
  - The OS itself needs to perform some specific tasks in kernel, such as managing devices or interrupt handling.
    - ✓ PARTICULAR, many OS systems are now multithreaded.
    - ✓ Example: Solaris, Linux

Thread concept overview

- Benefits
  - Responsiveness（响应度高）
    - ✓ Example: an interactive application such as web browser, while one thread loading an image, another thread allowing user interaction
  - Resource Sharing
    - ✓ address space, memory, and other resources
  - Economy
    - ✓ Solaris:
    - ✓ creating a process is about 30 times slower than creating a thread;
    - ✓ context switching is about 5 times slower
  - Utilization of MP Architectures
    - ✓ parallelism and concurrency ↑

Catalog Description

- Overview
- Multithreading Models
- Threading Issues

Two methods to support threads

- User threads VS. Kernel threads
  - User threads
    - ✓ Thread management done by user-level threads library without kernel support

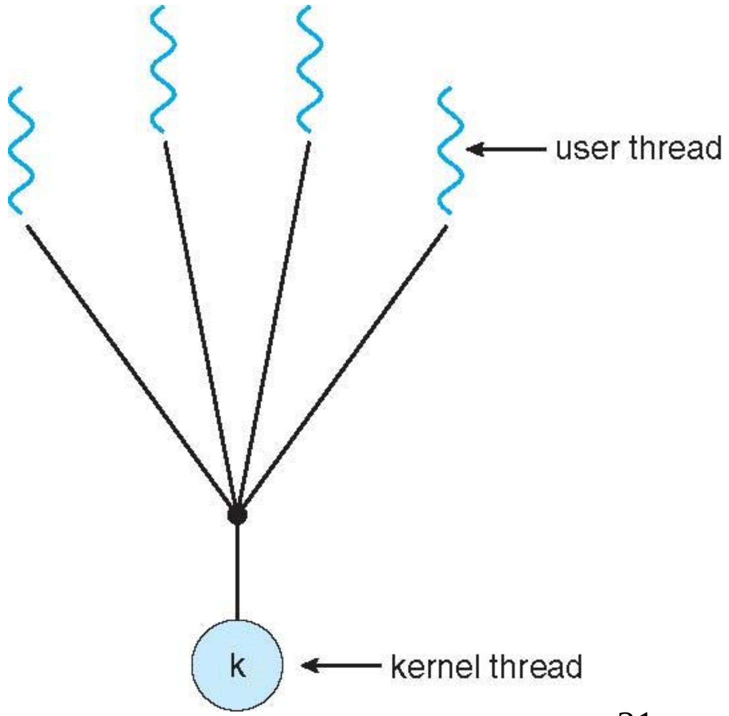
      --Kernel may be multithreaded or not.
  - Three primary thread libraries:
    - ✓ POSIX Pthreads
    - ✓ Win32 threads
    - ✓ Java threads

Two methods to support threads

- User threads VS. Kernel threads
  - Kernel threads
    - ✓ Supported by the Kernel, usually may be slower than user thread
  - Examples:
    - ✓ Windows XP/2000
    - ✓ Solaris
    - ✓ Linux
    - ✓ Tru64 UNIX (formerly Digital UNIX)
    - ✓ Mac OS X

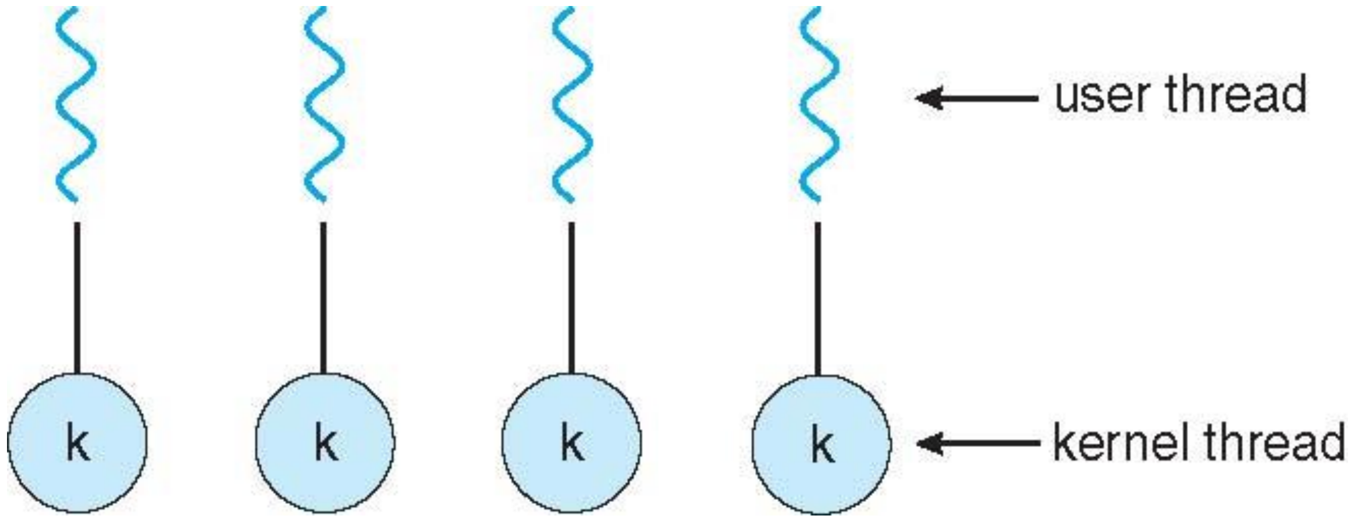
Multithreading Models

- The relationship between user threads and kernel threads
  - Many-to-One [n:1]
  - One-to-One [1:1]
  - Many-to-Many [n:m]
- Many-to-One [n:1]
  - Many user-level threads Mapped to single kernel thread
  - Examples:
    - ✓ Solaris Green Threads
    - ✓ GNU Portable Threads



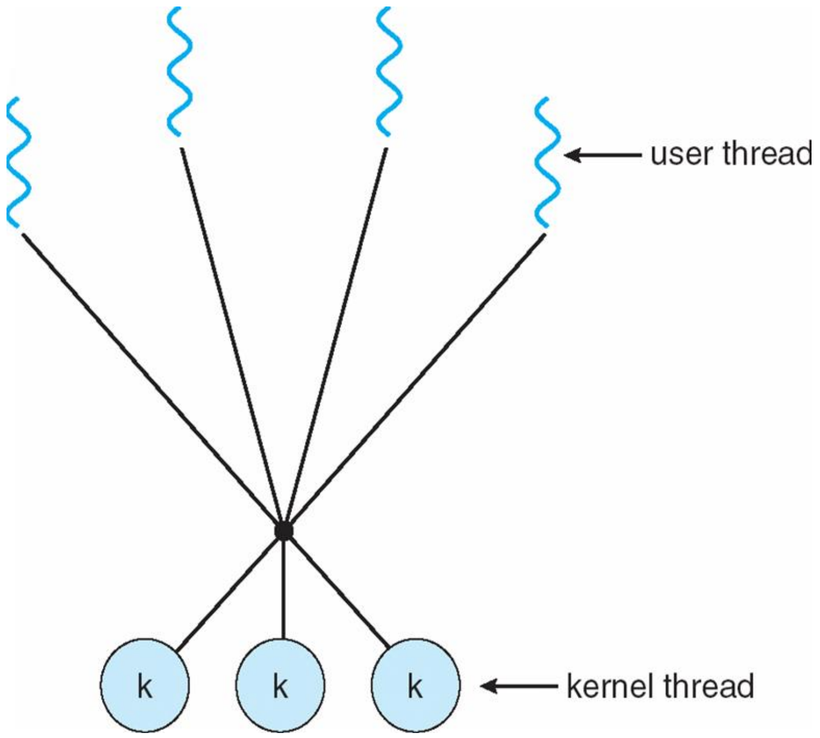
Multithreading Models

- One-to-One [1:1]
  - Each user-level thread maps to a kernel thread
  - Examples:
    - ✓ Windows NT/XP/2000
    - ✓ Linux
    - ✓ Solaris 9 and later



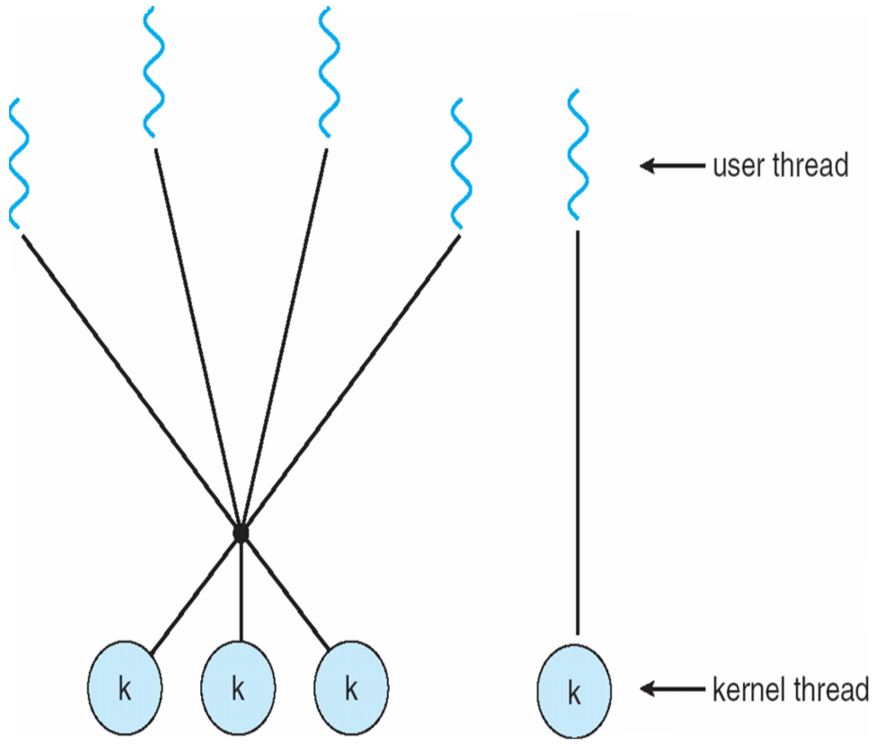
Multithreading Models

- Many-to-Many [n:m]
  - Allows many user level threads to be mapped to many kernel threads
  - Allows the operating system to create a sufficient number of kernel threads
  - Examples:
    - ✓ Solaris prior to version 9
    - ✓ Windows NT/2000 with the ThreadFiber package



Multithreading Models

- Two-level Model, a popular variation on many-to-many model
  - Similar to n:m, except that it allows a user thread to be bound to a kernel thread
  - Examples:
    - ✓ IRIX
    - ✓ HP-UX
    - ✓ Tru64 UNIX
    - ✓ Solaris 8 and earlier



Catalog Description

- Overview
- Multithreading Models
- Threading Issues

Threading Issues

- Semantics of fork() and exec() system calls
  - Does fork() duplicate only the calling thread or all threads?
  - Some UNIX system have chosen to have two versions
  - Which one version to use? Depend on the APP.
- Thread cancellation
  - Terminating a thread before it has finished
  - Two general approaches:
    - ✓ Asynchronous (异步) cancellation terminates the target thread immediately
    - ✓ Deferred (延时) cancellation allows the target thread to periodically check if it should be cancelled

Threading Issues

- Signal Handling
  - Signals are used in UNIX systems to notify a process that a particular event has occurred:
    - ✓ Synchronous: illegal memory access, division by 0
    - ✓ Asynchronous: Ctrl+C
  - All signals follow the same pattern:
    - ✓ Signal is generated by particular event
    - ✓ Signal is delivered to a process
    - ✓ Signal is handled
  - Signal handler may be handled by
    - ✓ a default signal handler, or
    - ✓ a user-defined signal handler

Threading Issues

- Signal Handling
  - When multithread, where should a signal be delivered?
    - ✓ 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

Threading Issues

- 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)

Threading Issues

- Scheduler Activations
  - Both n:m and Two-level models require communication to maintain the appropriate number of kernel threads allocated to the application
  - Scheduler activations provide up calls – a communication mechanism from the kernel to the thread library
  - This communication allows an application to maintain the correct number kernel threads