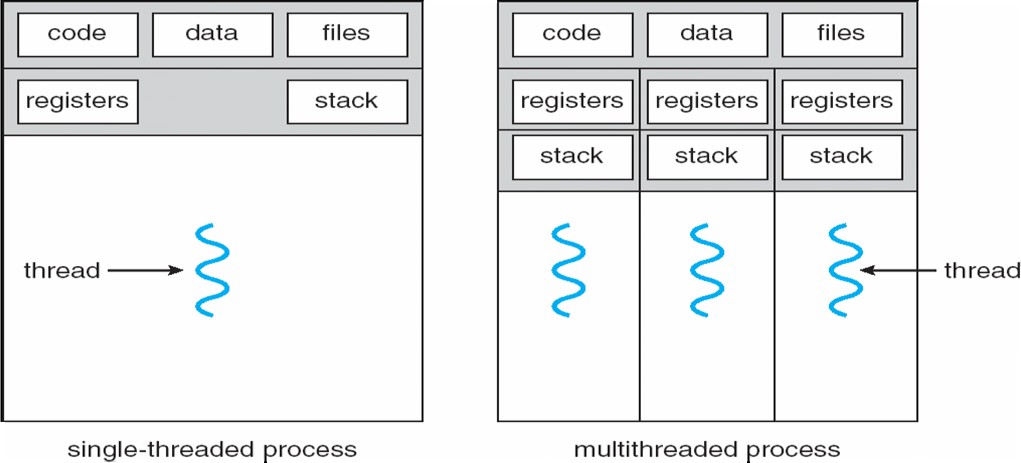
##### Unit 1 lecture 6

Topics to Be Covered:

* **Threads**
* Types of Thread

**Threads**

* To introduce the notion of a thread — a fundamental unit of CPU utilization that forms the basis of multithreaded computer systems
* To discuss the APIs for the Pthreads, Win32, and Java thread libraries
* To examine issues related to multithreaded programming

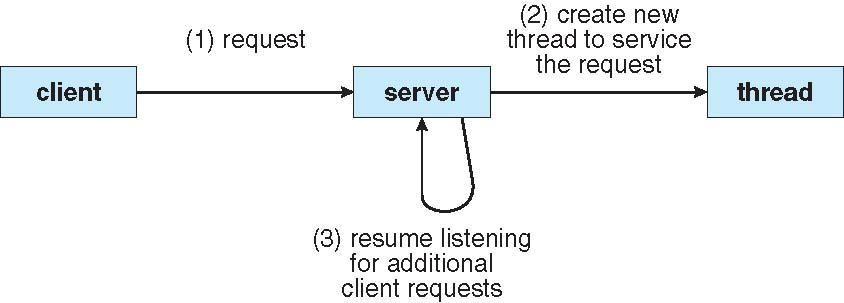
** Single and Multithreaded Processes**

**Benefits**

* **Responsiveness**
* **Resource sharing**
* **Economy**
* **Scalability**

**Multithreaded Server**

**Architecture**

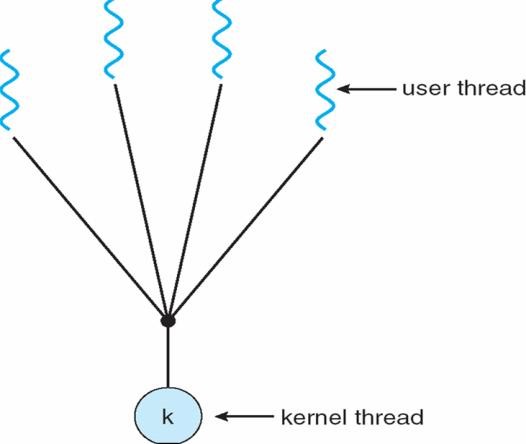


**Types of Thread**

**User Threads**

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

**Kernel Threads**

* Supported by the Kernel

Multithreading Models

* **Many –To – one**
* **One-To – One**
* **Many-To-Many**

##### Many-to-One

Many user-level threads mapped to single kernel thread

Ex Solaris Green Thread

##### One-to-One

Each user-level thread maps to kernel thread Examples Windows NT/XP/2000

##### Many-to-Many Model

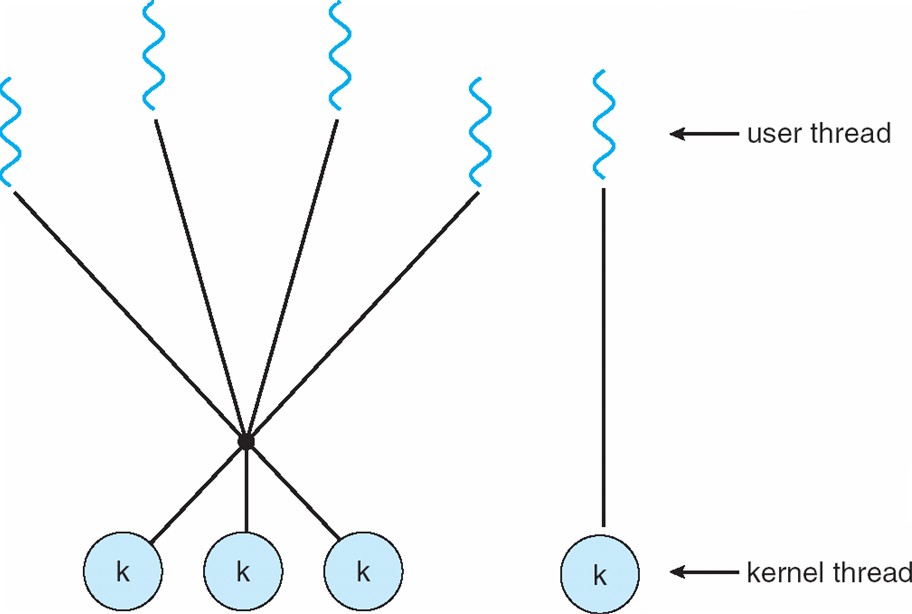
##### Allows many user level threads to be mapped to many kernel thread

##### 

##### Allows the operating system to create a sufficient number of kernel threads

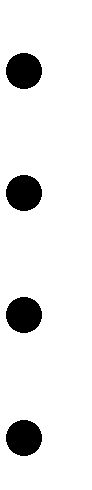
##### Solaris prior to version 9Windows NT/2000 with the *ThreadFiber* package

##### Two-level Model

Similar to M:M, except that it allows a user thread to be **bound** to kernel thread Examples

* IRIX
* HP-UX
* Solaris 8 and earlier

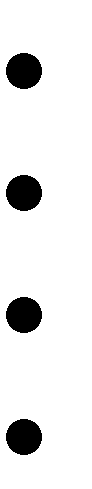
##### Thread Libraries

Thread library provides programmer with API for creating and managing threads Two primary ways of implementing

Library entirely in user space

Kernel-level library supported by the OS

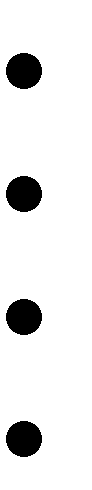
##### Pthreads

May be provided either as user-level or kernel-level

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)

##### Java Threads

Java threads are managed by the JVM

Typically implemented using the threads model provided by underlying OS Java threads may be created by:lExtending Thread class

Implementing the Runnable interface

##### Thread Cancellation

* Terminating thread before it has finished
* Two general approaches:

1. **Asynchronous cancellation** terminates the target thread immediately
2. **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
* 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 threa 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)

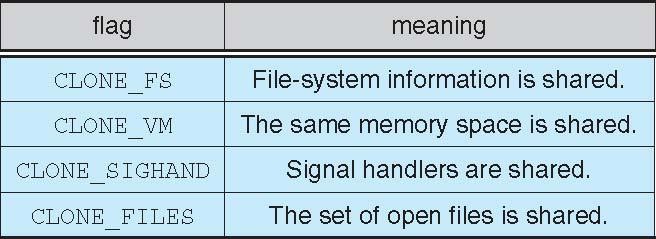
##### Scheduler Activations

* 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

Implements the one-to-one mapping, kernel-level

* 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)

##### CPU Scheduling

* To introduce CPU scheduling, which is the basis for multiprogrammed operating systems
* To describe various CPU-scheduling algorithms
* To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system
* Maximum CPU utilization obtained with multiprogramming CPU–I/O Burst Cycle – Process execution consists of a *cycle* of CPU execution and I/O wait
* **CPU burst** distribution

##### Histogram of CPU-burst Times

