Chapter 4: Threads

Chapter 4: Threads

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

Process

- Resource ownership - process includes a virtual address space to hold the process image
- Scheduling/execution- follows an execution path that may be interleaved with other processes
- **These two characteristics** are treated independently by the operating system

Separate two ideas:

- Process: Ownership of memory, files, other resources
- Thread: Unit of execution we use to dispatch

Process/Thread

Process--Resource ownership

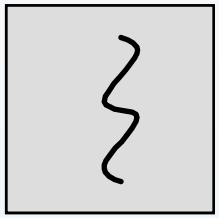
- Have a virtual address space which holds the process image
- Protected access to processors, other processes, files, and I/O resources

Thread--Scheduling/Execution

- An execution state (running, ready, etc.)
- Saved thread context when not running
- Some per-thread static storage for local variables
- Access to the memory and resources of its process

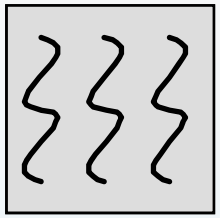
Multithreading

DOS



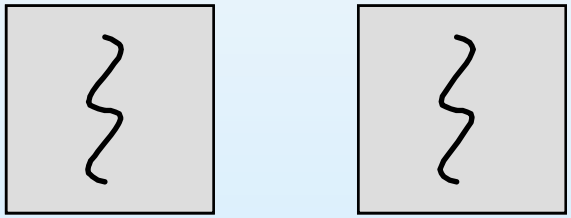
one process
one thread

Java



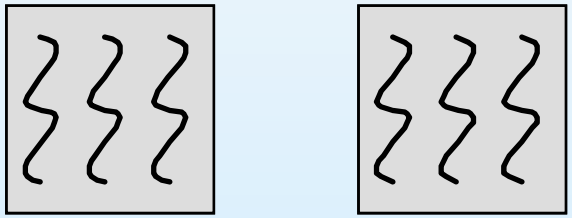
one process
multiple threads

UNIX



multiple processes
one thread per process

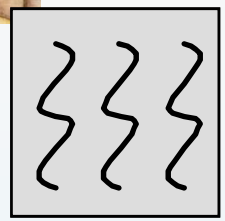
Windows
Linux
OS/2



multiple processes
multiple threads per process

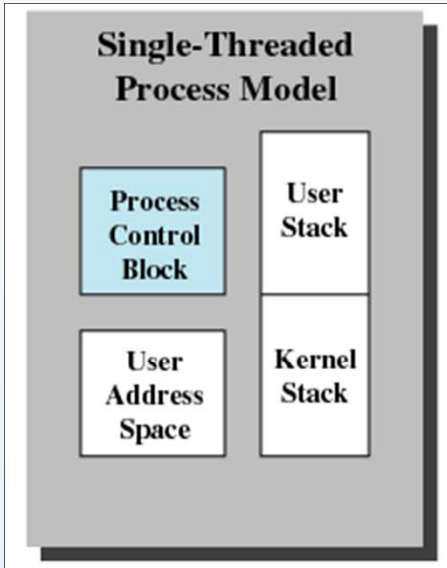
Operating system supports multiple threads of execution within **a single process**

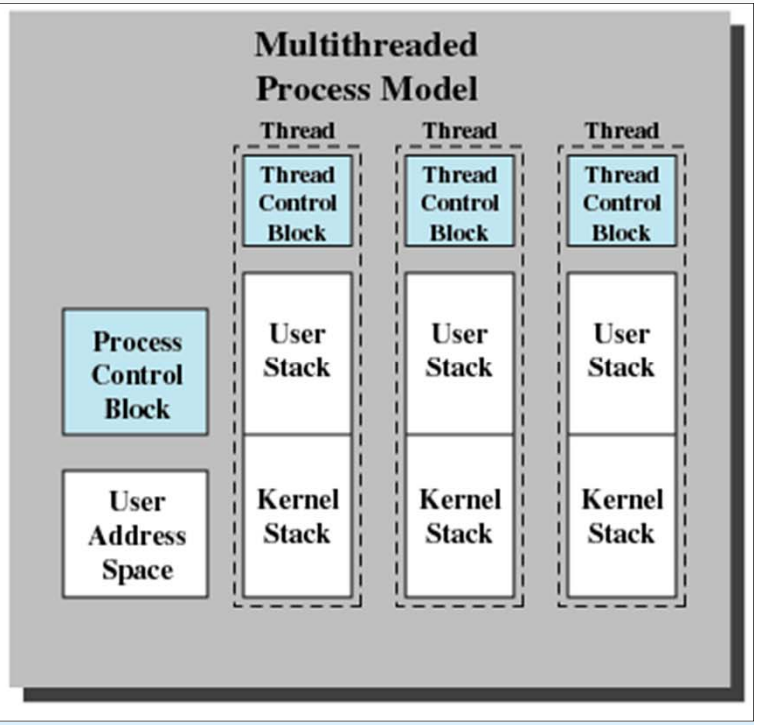
Multithreading

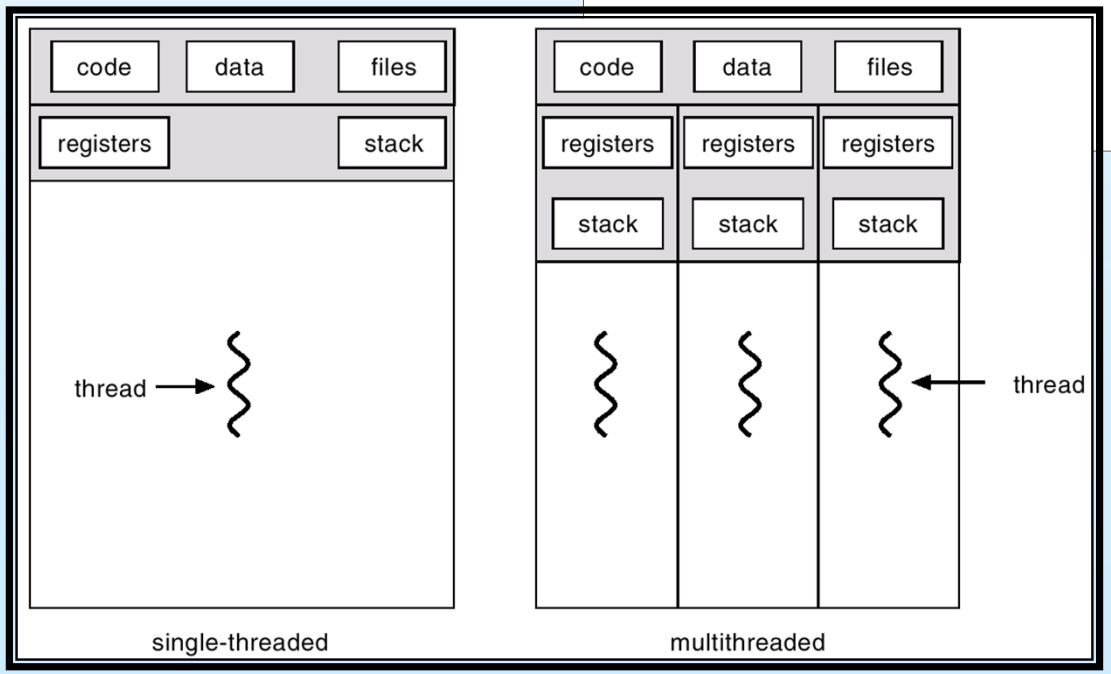


Operating system supports **multiple threads** of execution within **a single process**

Single and Multithreaded Processes



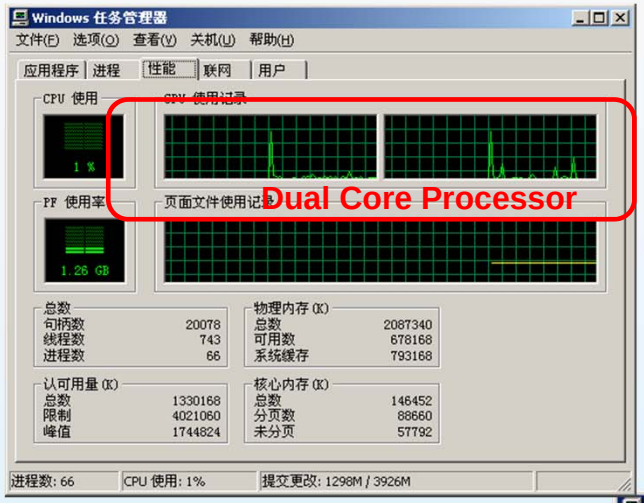




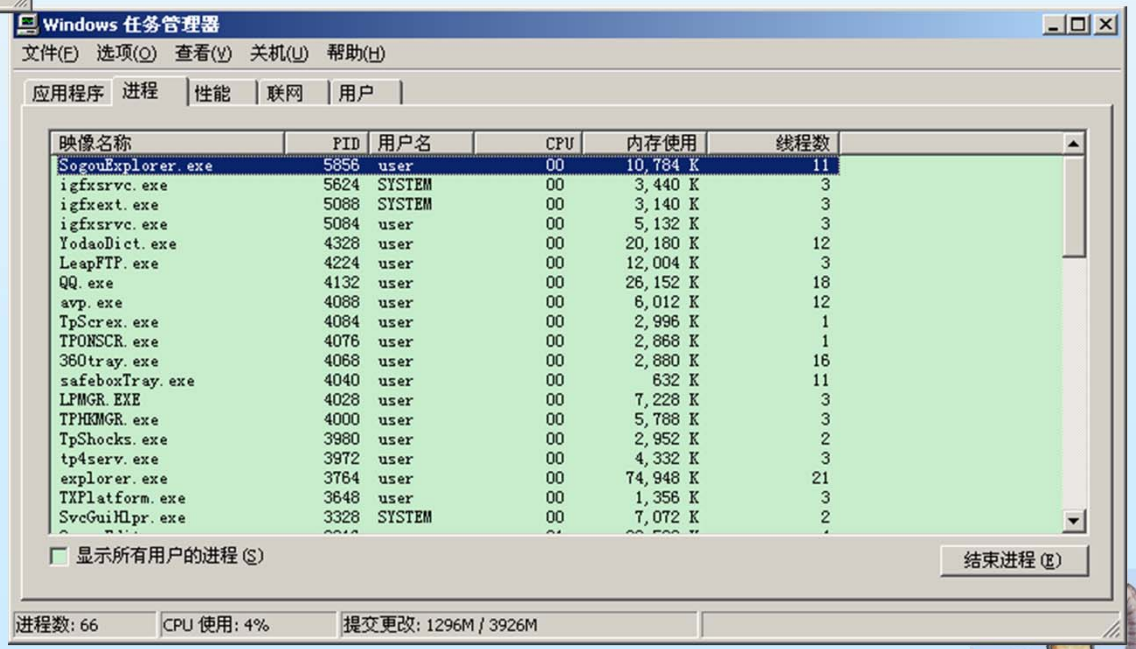
Benefits

- Responsiveness
- Resource Sharing
- Economy
- Utilization of MP Architectures

Examples



Thread sorting demo



User and Kernel Threads

- User threads - Thread management done by user-level threads library.
- Kernel threads - Threads directly supported by the kernel.

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

Multithreading Models

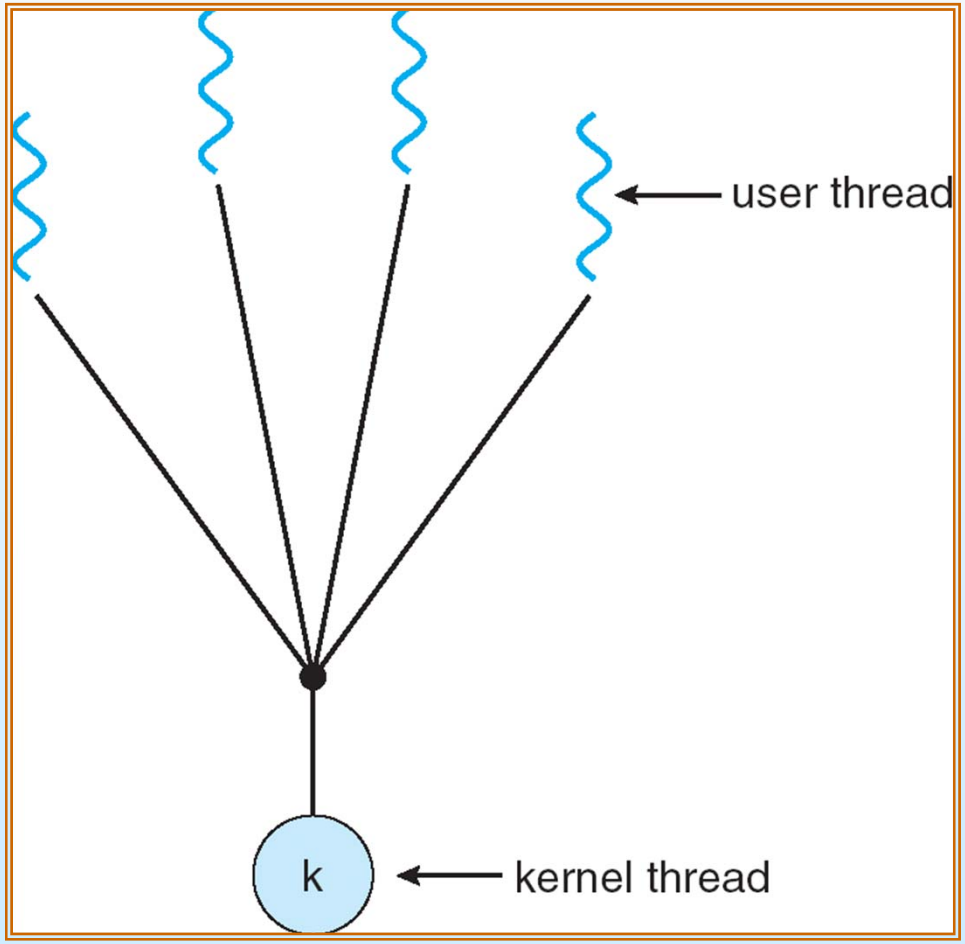
Mapping user threads to kernel threads:

- Many-to-One
- One-to-One
- Many-to-Many

Many-to-One

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

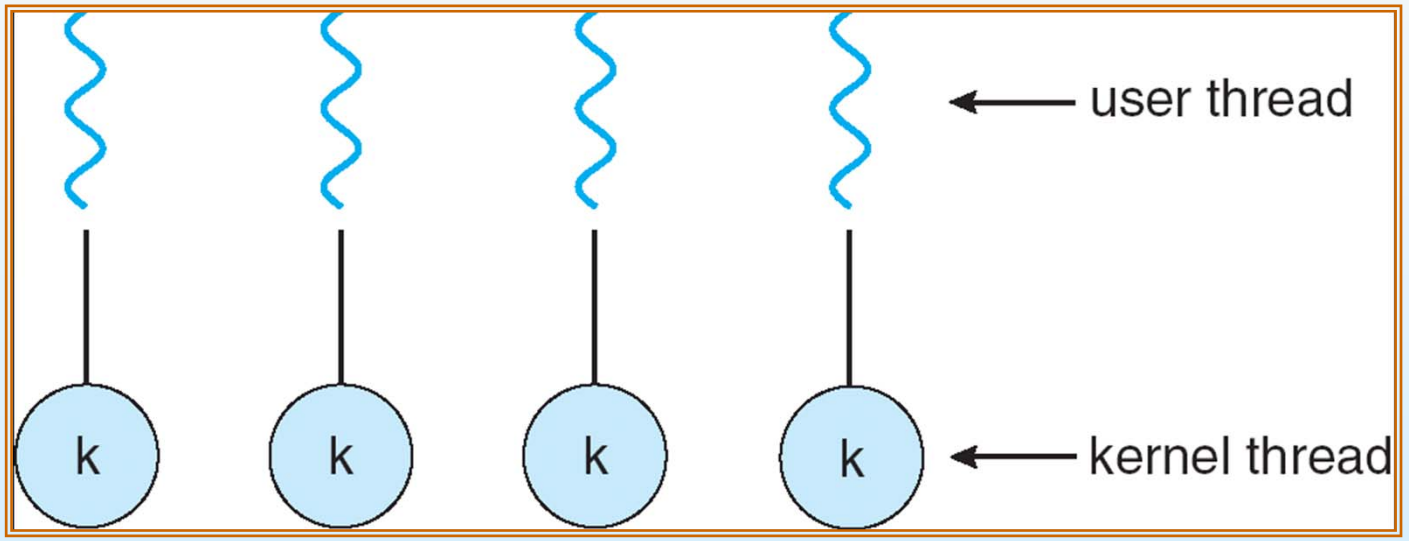
Many-to-One Model



One-to-One

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

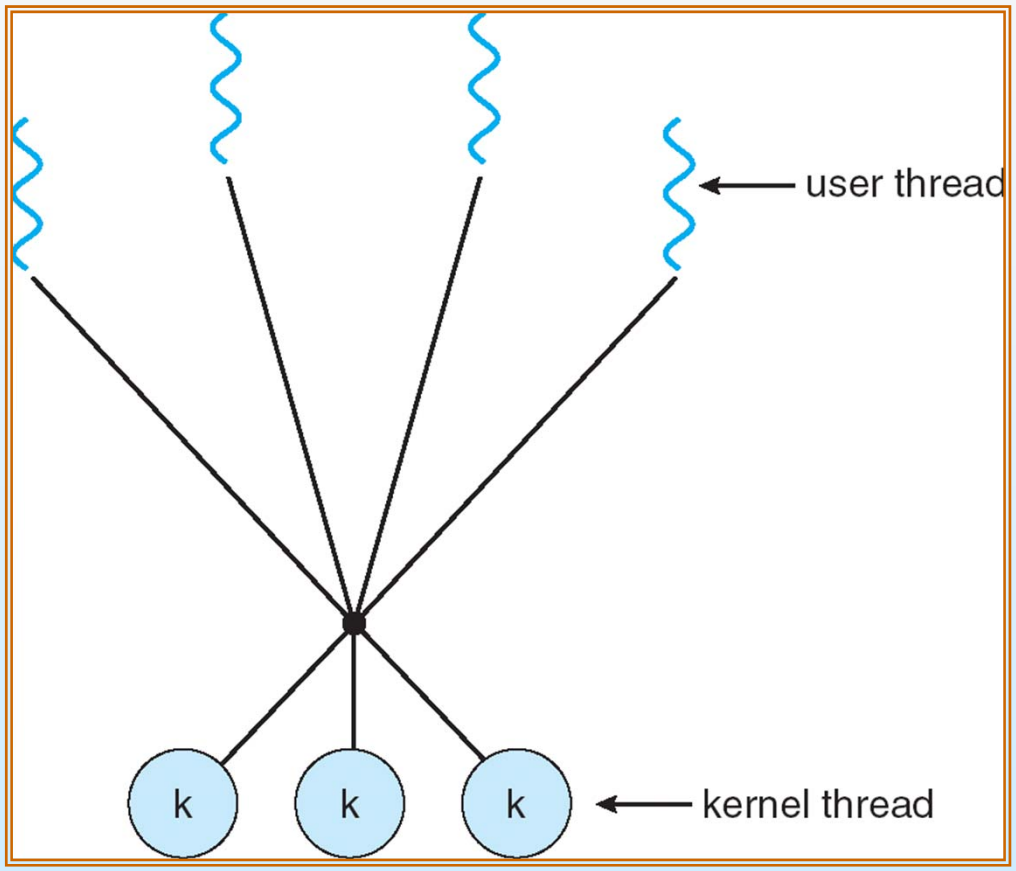
One-to-one Model



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

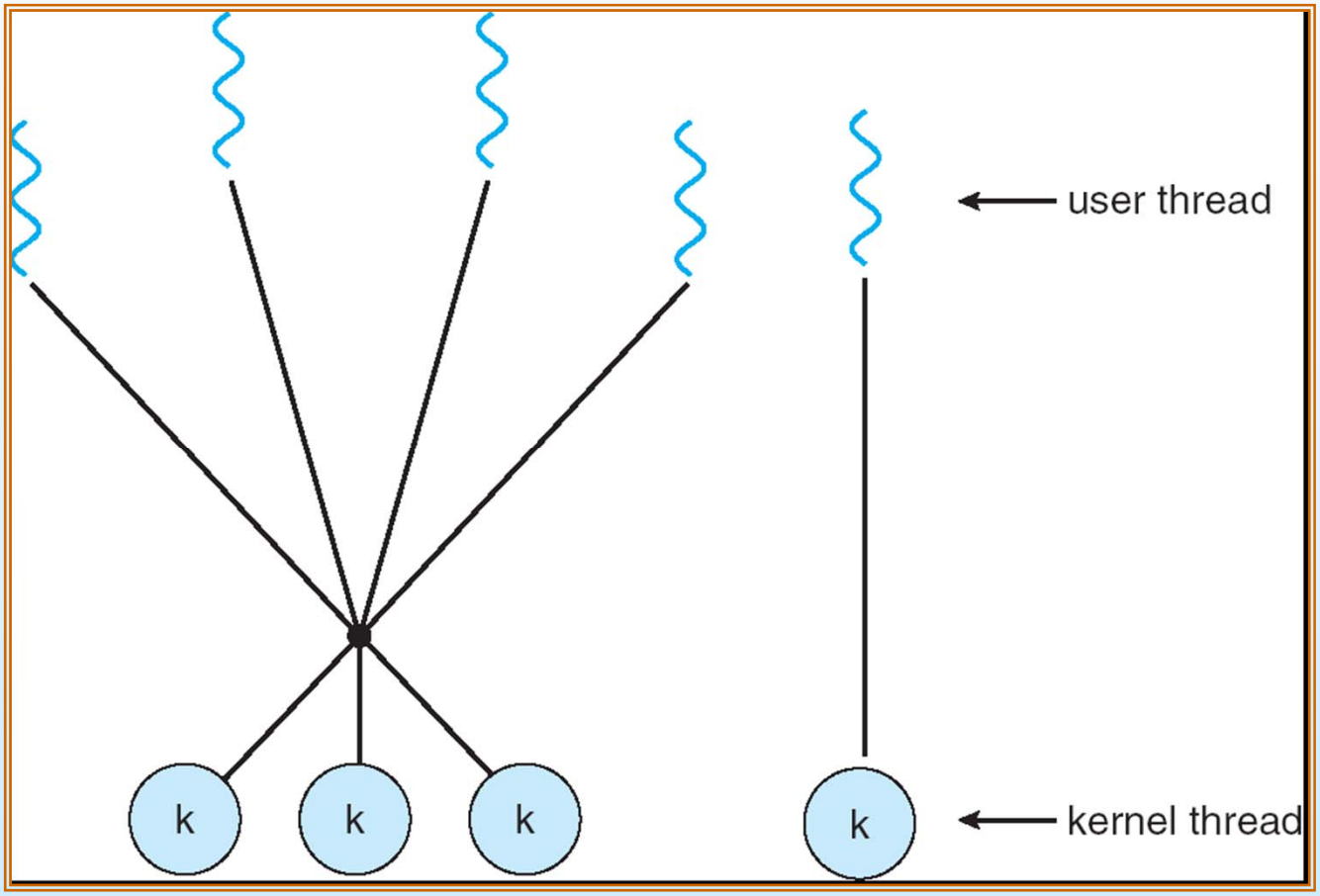
Many-to-Many Model



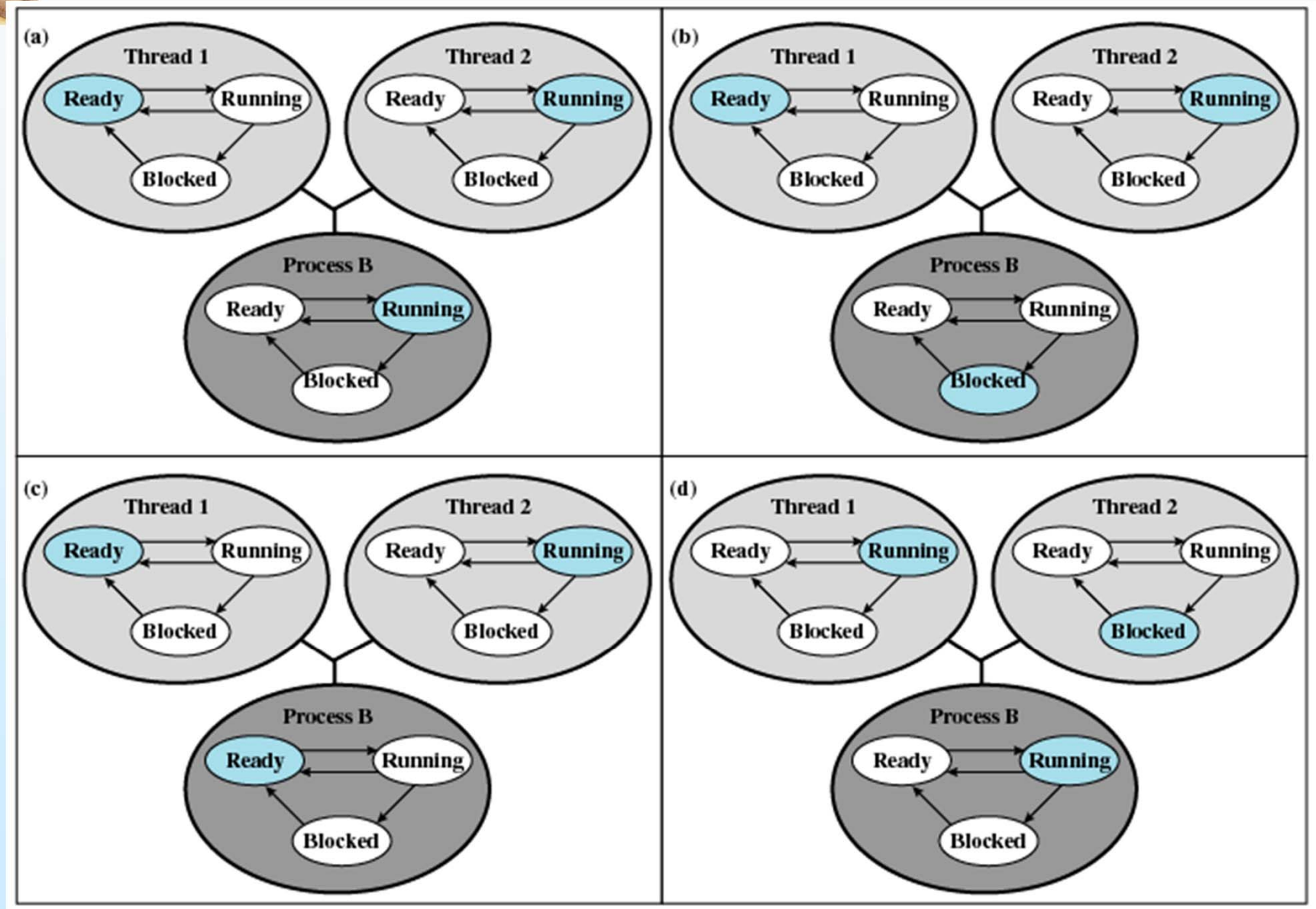
Two-level Model

- 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

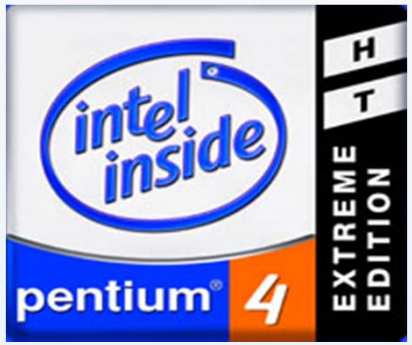
Two-level Model



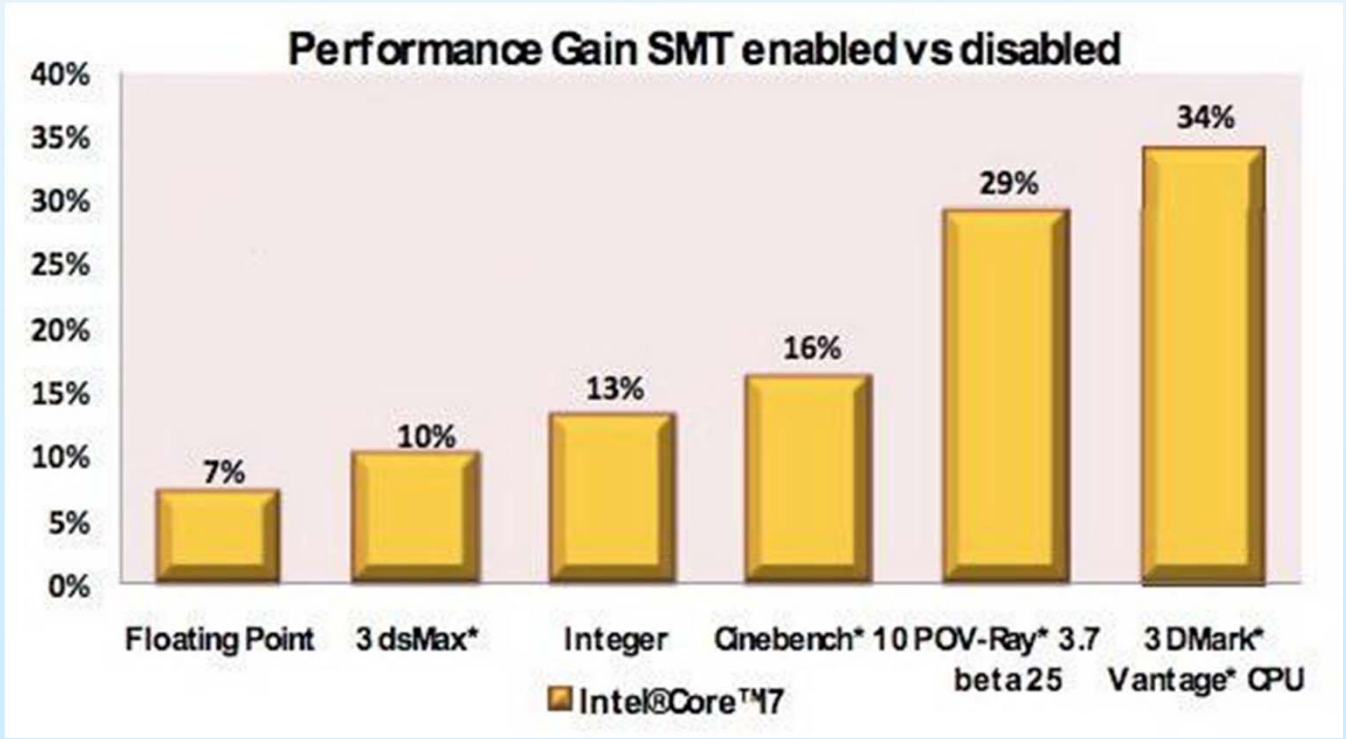
User Threads



Hyper-Threading Technology



1. Parallelism via better utilization of existing resources
2. Hyper-threading is an Intel-proprietary technology used to improve parallelization of computations (doing multiple tasks at once) performed on PC microprocessors.



Threading Issues

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

Java Threads

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

```
public interface Runnable
{
    public abstract void run();
}
```

Java Threads - Example Program

```
class MutableInteger
{
   private int value;
   public int getValue() {
     return value;
   }
   public void setValue(int value) {
     this.value = value;
   }
}

class Summation implements Runnable
{
   private int upper;
   private MutableInteger sumValue;
   public Summation(int upper, MutableInteger sumValue) {
     this.upper = upper;
     this.sumValue = sumValue;
   }
   public void run() {
     int sum = 0;
     for (int i = 0; i <= upper; i++)
        sum += i;
     sumValue.setValue(sum);
   }
}
```

Java Threads - Example Program

```
public class Driver
{
  public static void main(String[] args) {
    if (args.length > 0) {
      if (Integer.parseInt(args[0]) < 0)
        System.err.println(args[0] + " must be >= 0.");
      else {
        // create the object to be shared
        MutableInteger sum = new MutableInteger();
        int upper = Integer.parseInt(args[0]);
        Thread thrd = new Thread(new Summation(upper, sum));
        thrd.start();
        try {
          thrd.join();
          System.out.println
                  ("The sum of "+upper+" is "+sum.getValue());
        } catch (InterruptedException ie) { }
      }
    }
    else
      System.err.println("Usage: Summation <integer value>");
    }
}
```

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)

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)

End of Chapter 4