PRINCIPLES OF OPERATING SYSTEMS

LECTURE- 7

Principles of Operating Systems

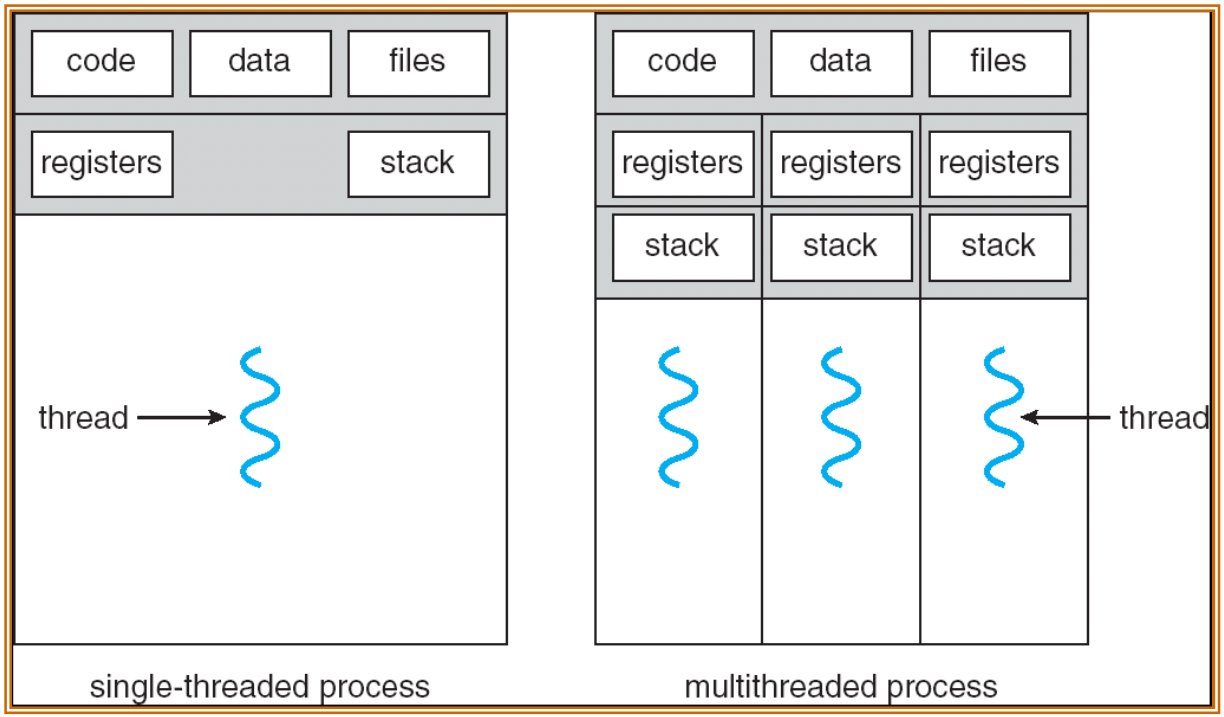
USER LEVEL AND KERNEL LEVEL THREADS

Threads

- Processes do not share resources well
  - high context switching overhead
- Idea: Separate concurrency from protection

- Multithreading: *a single program made up of a number of different concurrent activities*
- A thread (or lightweight process)
  - basic unit of CPU utilization; it consists of:
    - program counter, register set and stack space
  - A thread shares the following with peer threads:
    - code section, data section and OS resources (open files, signals)
    - No protection between threads
  - Collectively called a task.
- Heavyweight process is a task with one thread.

Single and Multithreaded Processes



- Threads encapsulate concurrency: “Active” component
- Address spaces encapsulate protection: “Passive” part
  - Keeps buggy program from trashing the system

Benefits

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

Threads(Cont.)

- In a multiple threaded task, while one server thread is blocked and waiting, a second thread in the same task can run.
  - Cooperation of multiple threads in the same job confers higher throughput and improved performance.
  - Applications that require sharing a common buffer (i.e. producer-consumer) benefit from thread utilization.
- Threads provide a mechanism that allows sequential processes to make blocking system calls while also achieving parallelism.

Thread State

- State shared by all threads in process/addr space
  - Contents of memory (global variables, heap)
  - I/O state (file system, network connections, etc)
- State "private" to each thread
  - Kept in TCB ≡ Thread Control Block
  - CPU registers (including, program counter)
  - Execution stack
    - Parameters, Temporary variables
    - return PCs are kept while called procedures are executing

Threads (cont.)

- Thread context switch still requires a register set switch, but no memory management related work!!
- Thread states -
  - *ready, blocked, running, terminated*
- Threads share CPU and only one thread can run at a time.
- No protection among threads.

Examples: Multithreaded programs

- Embedded systems
  - Elevators, Planes, Medical systems, Wristwatches
  - Single Program, concurrent operations
- Most modern OS kernels
  - Internally concurrent because have to deal with concurrent requests by multiple users
  - But no protection needed within kernel
- Database Servers
  - Access to shared data by many concurrent users
  - Also background utility processing must be done

More Examples: Multithreaded programs

- Network Servers
  - Concurrent requests from network
  - Again, single program, multiple concurrent operations
  - File server, Web server, and airline reservation systems
- Parallel Programming (More than one physical CPU)
  - Split program into multiple threads for parallelism
  - This is called Multiprocessing

# threads Per AS: / # of addr spaces:	One	Many
One	MS/DOS, early Macintosh	Traditional UNIX
Many	Embedded systems (Geoworks, VxWorks, JavaOS,etc) JavaOS, Pilot(PC)	Mach, OS/2, Linux Windows 9x??? Win NT to XP, Solaris, HP-UX, OS X

Real operating systems have either

- One or many address spaces
- One or many threads per address space

Types of Threads

- Kernel-supported threads (Mach and OS/2)
- User-level threads
- Hybrid approach implements both user-level and kernel-supported threads (Solaris 2).

Kernel Threads

- Supported by the Kernel
  - Native threads supported directly by the kernel
  - Every thread can run or block independently
  - One process may have several threads waiting on different things
- Downside of kernel threads: a bit expensive
  - Need to make a crossing into kernel mode to schedule
- Examples
  - Windows XP/2000, Solaris, Linux,Tru64 UNIX, Mac OS X, Mach, OS/2

User Threads

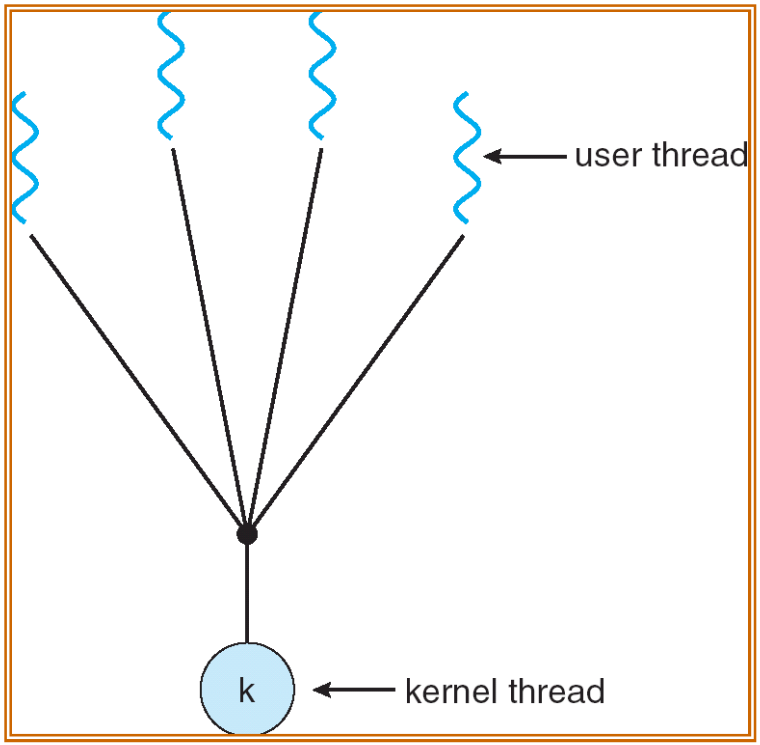
- Supported above the kernel, via a set of library calls at the user level.
  - Thread management done by user-level threads library
    - User program provides scheduler and thread package
  - May have several user threads per kernel thread
  - User threads may be scheduled non-premptively relative to each other (only switch on yield())
  - Advantages
    - Cheap, Fast
      - Threads do not need to call OS and cause interrupts to kernel
  - Disadv: If kernel is single threaded, system call from any thread can block the entire task.
- Example thread libraries:
  - POSIX Pthreads, Win32 threads, Java threads

Multithreading Models

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

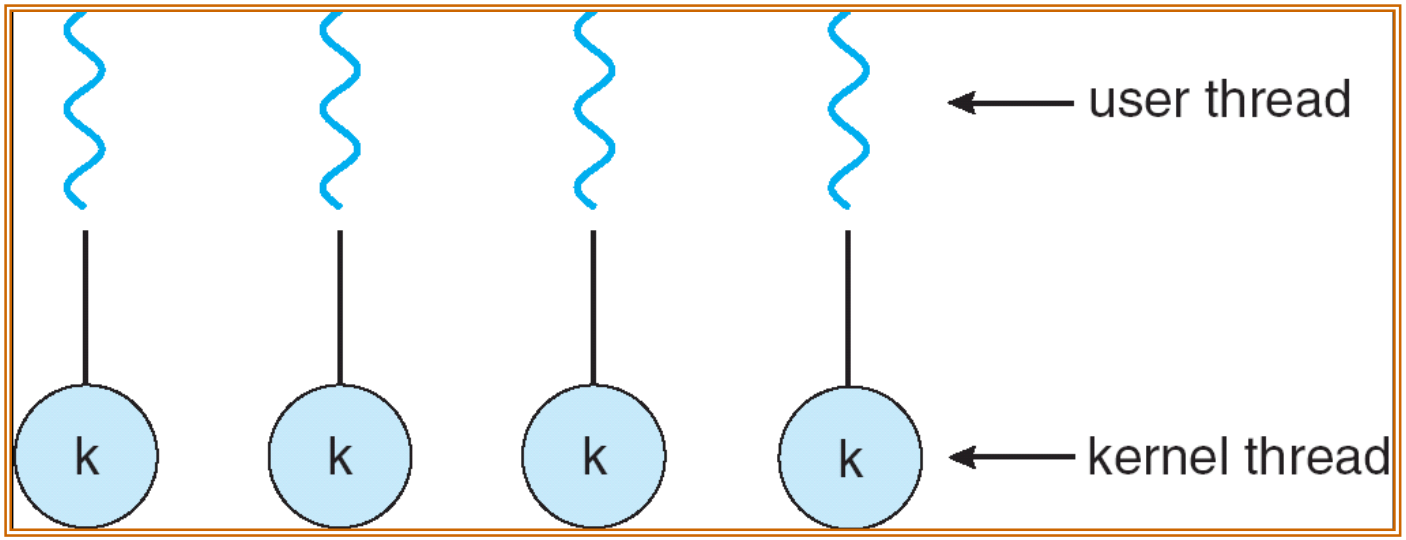
Many-to-One

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



One-to-One

- Each user-level thread maps to kernel thread



Examples

- Windows NT/XP/2000; Linux; Solaris 9 and later

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