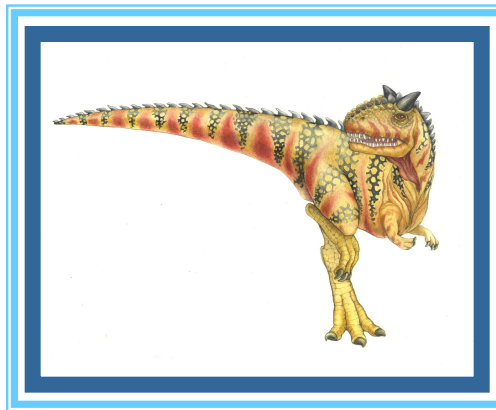


Chapter 4: Threads





Chapter 4: Threads

- Overview
- Multicore Programming
- Multithreading Models
- Thread Libraries





Objectives

- 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, Windows, and Java thread libraries





Motivation

- Most modern applications are multithreaded
- Threads run within application
- Multiple tasks with the application can be implemented by separate threads
 - Update display
 - Fetch data
 - Spell checking
 - Answer a network request
- Process creation is heavy-weight while thread creation is light-weight
- Can simplify code, increase efficiency
- Kernels are generally multithreaded





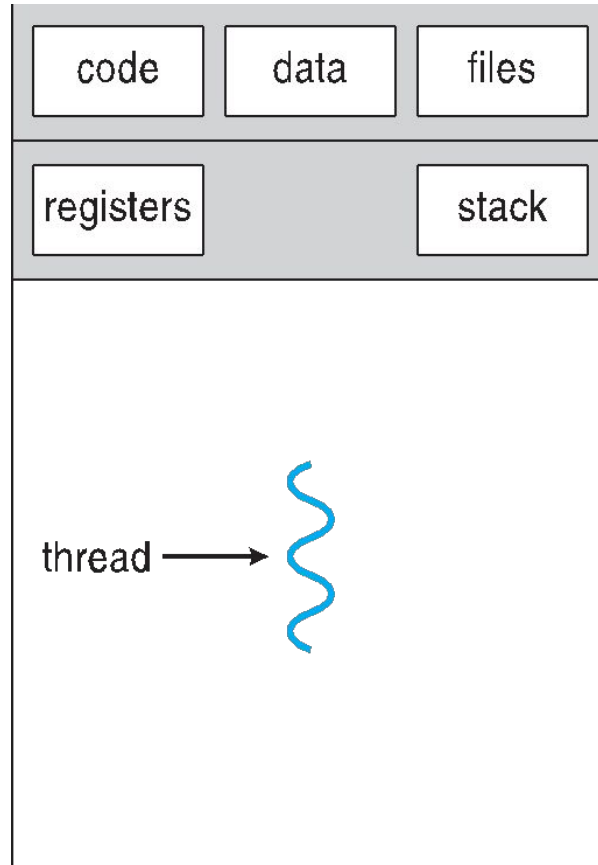
Threads

- ❑ A thread is a single sequential flow of control within a program.
Threads have same properties as of the process so they are called as light weight processes. Each thread has different states. Each thread has
 - A program counter
 - A register set
 - A stack space
- ❑ Threads are not independent of each other as they share the code, data, OS resources etc.

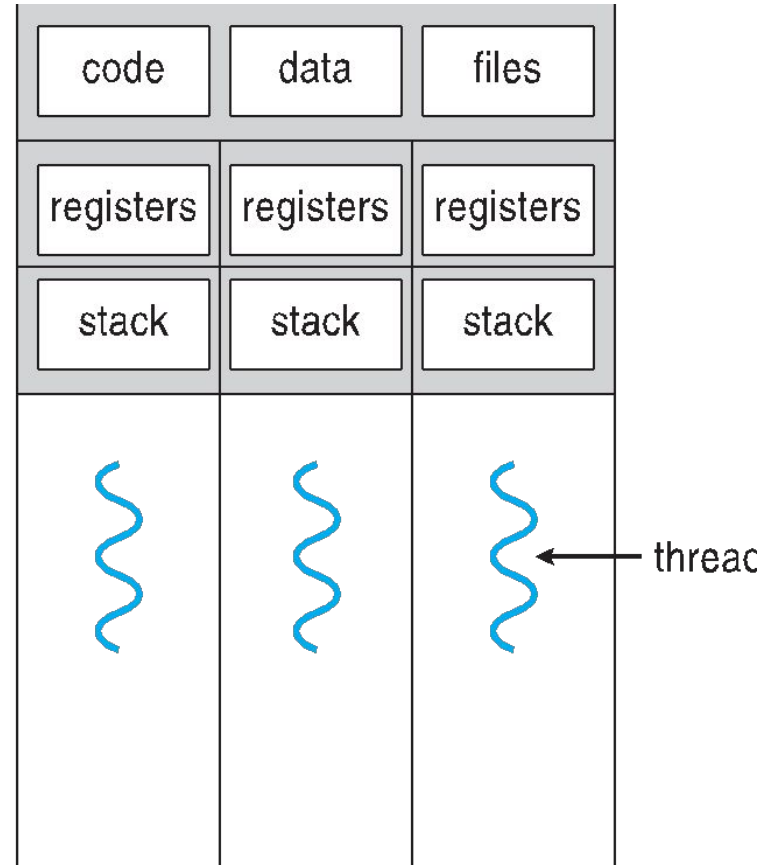




Single and Multithreaded Processes



single-threaded process

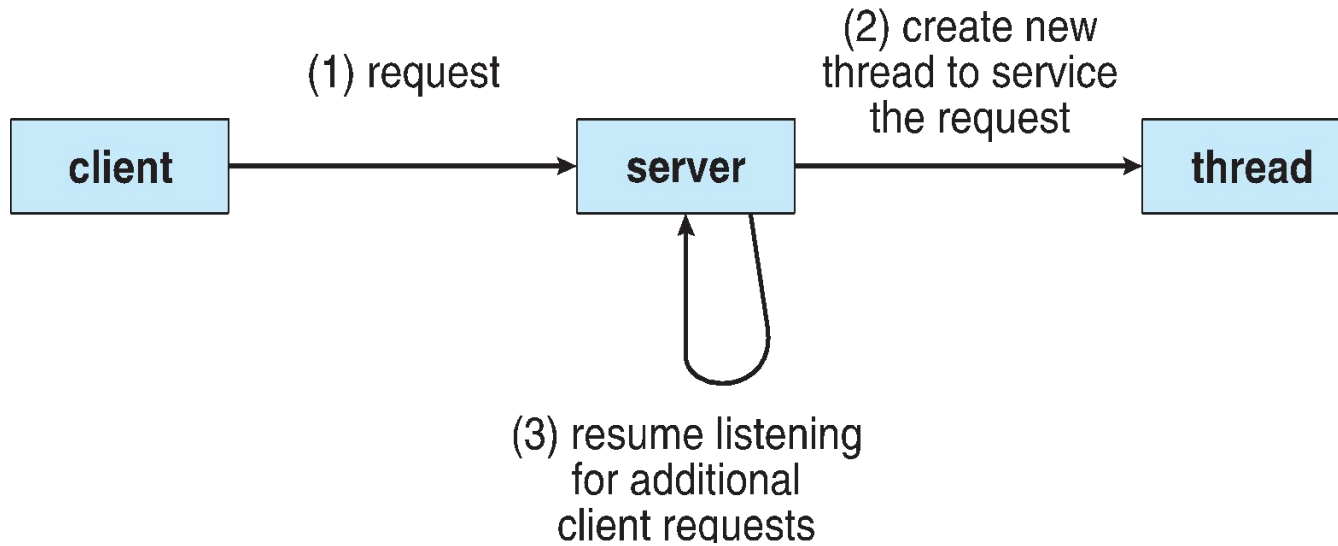


multithreaded process





Multithreaded Server Architecture





Benefits

- **Responsiveness** – may allow continued execution if part of process is blocked, especially important for user interfaces
- **Resource Sharing** – threads share resources of process, easier than shared memory or message passing
- **Economy** – cheaper than process creation, thread switching lower overhead than context switching
- **Scalability** – process can take advantage of multiprocessor architectures





Process vs threads

| Comparison Basis | Process | Thread |
|-------------------|---|---|
| Definition | A process is a program under execution i.e an active program. | A thread is a lightweight process that can be managed independently by a scheduler. |
| Context Switching | Processes require more time for context switching as they are more heavy. | Threads require less time for context switching as they are lighter than processes. |
| Memory Sharing | Processes are totally independent and don't share memory. | A thread may share some memory with its peer threads. |
| Communication | Communication between processes requires more time than between threads. | Communication between threads requires less time than between processes . |
| Blocked | If a process gets blocked, remaining processes can continue execution. | If a user level thread gets blocked, all of its peer threads also get blocked. |





Process vs threads

| | | |
|-----------------------|---|---|
| Resource Consumption | Processes require more resources than threads. | Threads generally need less resources than processes. |
| Dependency | Individual processes are independent of each other. | Threads are parts of a process and so are dependent. |
| Data and Code sharing | Processes have independent data and code segments. | A thread shares the data segment, code segment, files etc. with its peer threads. |
| Treatment by OS | All the different processes are treated separately by the operating system. | All user level peer threads are treated as a single task by the operating system. |





Multicore Programming

- **Multicore** or **multiprocessor** systems putting pressure on programmers, challenges include:
 - **Dividing activities**
 - **Balance**
 - **Data splitting**
 - **Data dependency**
 - **Testing and debugging**
- **Parallelism** implies a system can perform more than one task simultaneously
- **Concurrency** supports more than one task making progress
 - Single processor / core, scheduler providing concurrency



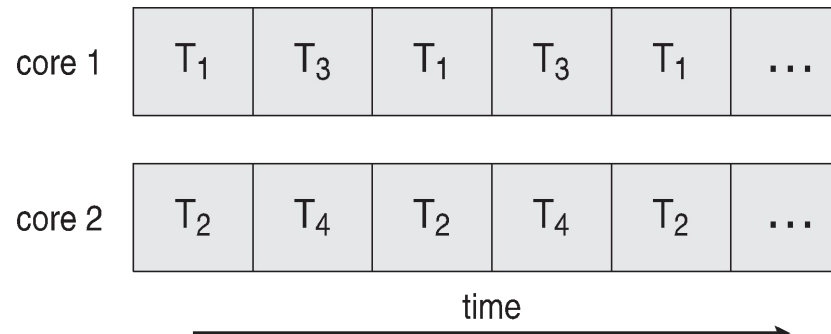


Concurrency vs. Parallelism

- **Concurrent execution on single-core system:**



- **Parallelism on a multi-core system:**





Multicore Programming (Cont.)

- Types of parallelism
 - **Data parallelism** – distributes subsets of the same data across multiple cores, same operation on each
 - **Task parallelism** – distributing threads across cores, each thread performing unique operation
- As # of threads grows, so does architectural support for threading
 - CPUs have cores as well as **hardware threads**
 - Consider Oracle SPARC T4 with 8 cores, and 8 hardware threads per core





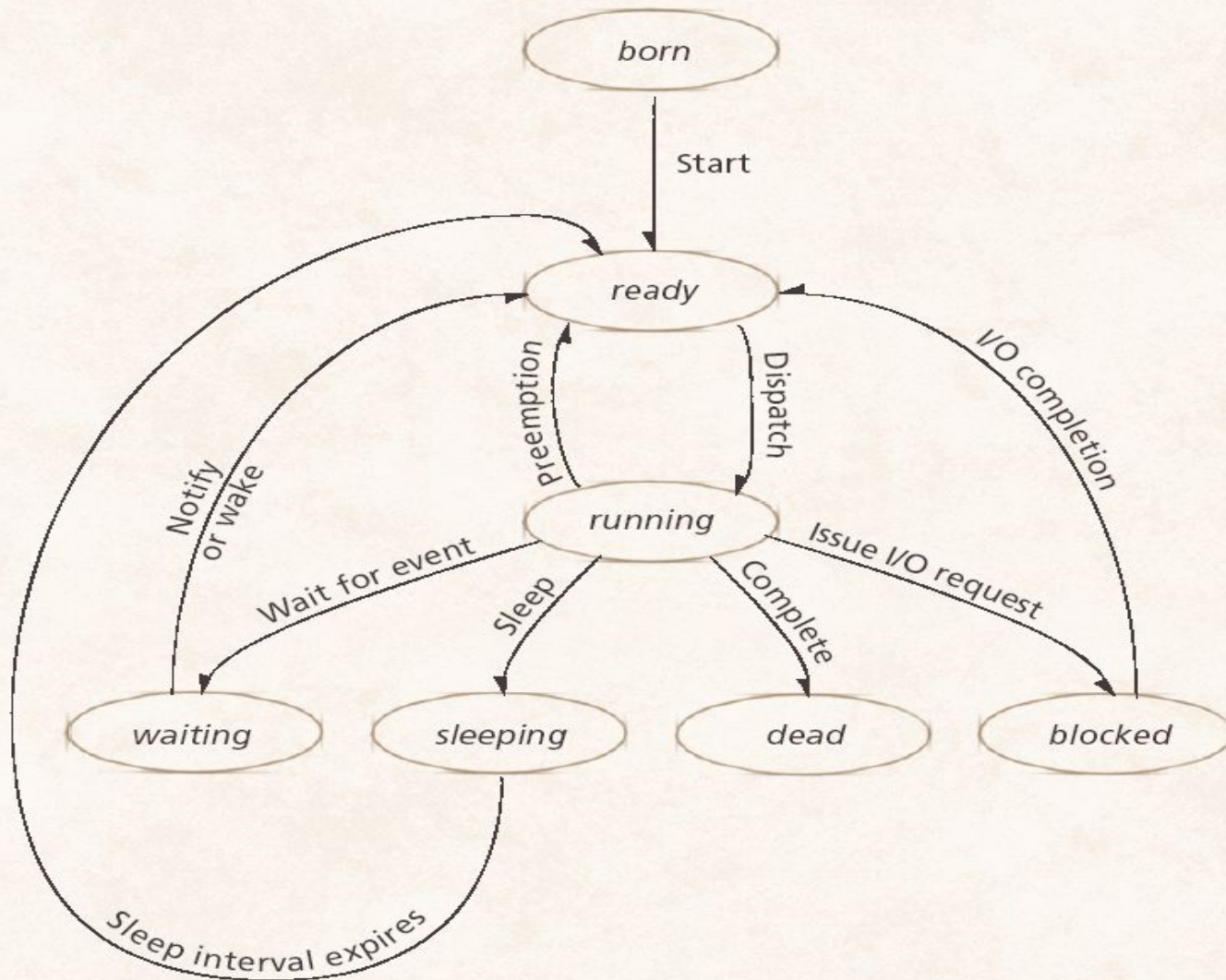
Thread States : Life Cycle of a Thread

- *Born state*
- *Ready state (runnable state)*
- *Running state*
- *Dead state*
- *Blocked state*
- *Waiting state*
- *Sleeping state*
 - 4 Sleep interval specifies for how long a thread will sleep





Thread States : Life Cycle of a Thread





User Threads and Kernel Threads

- **User threads** - management done by user-level threads library
- Three primary thread libraries:
 - POSIX **Pthreads**
 - Windows threads
 - Java threads
- **Kernel threads** - Supported by the Kernel
- Examples – virtually all general purpose operating systems, including:
 - Windows
 - Solaris
 - Linux
 - Mac OS X





User level vs kernel level threads

| USER LEVEL THREAD | KERNEL LEVEL THREAD |
|--|---|
| User thread are implemented by users. | kernel threads are implemented by OS. |
| OS doesn't recognized user level threads. | Kernel threads are recognized by OS. |
| Implementation of User threads is easy. | Implementation of Kernel thread is complicated. |
| Context switch time is less. | Context switch time is more. |
| Context switch requires no hardware support. | Hardware support is needed. |
| If one user level thread perform blocking operation then entire process will be blocked. | If one kernel thread perform blocking operation then another thread can continue execution. |
| User level threads are designed as dependent threads. | Kernel level threads are designed as independent threads. |
| User-level threads are generally fast to create and manage | The kernel-level threads are slow and inefficient. For instance, threads operations are hundreds of times slower than that of user-level threads. |
| User level thread is generic and can run on any operating system. | Kernel level thread is specific to the operating system. |
| Example : Java thread, POSIX threads. | Example : Window Solaris. |





Multithreading Models

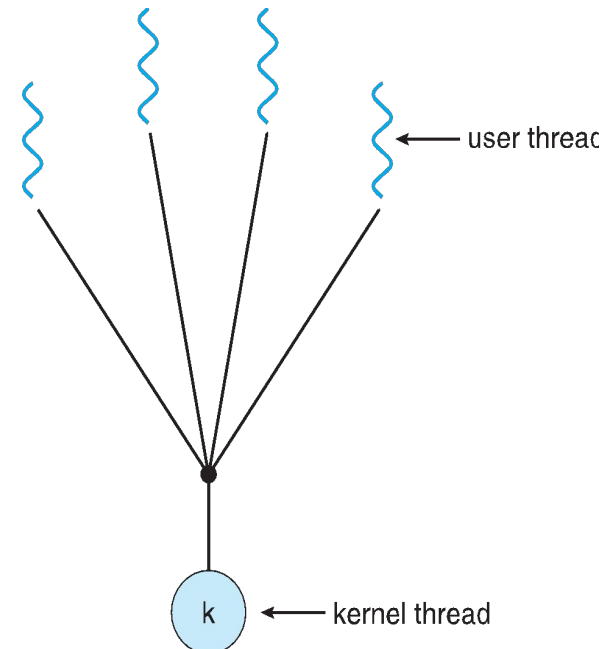
- Many-to-One
- One-to-One
- Many-to-Many





Many-to-One

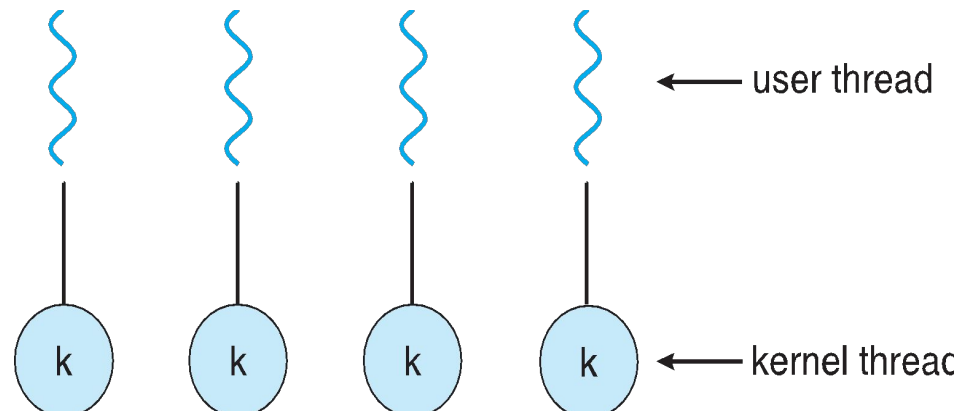
- Many user-level threads mapped to single kernel thread
- One thread blocking causes all to block
- Multiple threads may not run in parallel on multicore system because only one may be in kernel at a time
- Few systems currently use this model
- Examples:
 - **Solaris Green Threads**
 - **GNU Portable Threads**





One-to-One

- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead
- Examples
 - Windows
 - 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 with the *ThreadFiber* package

