Processes and Threads

- Processes have two characteristics:
  - **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

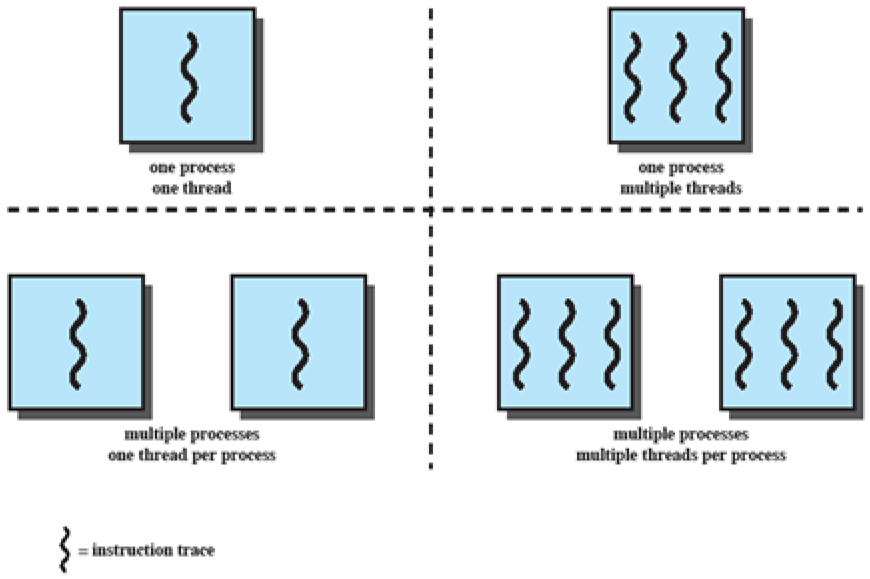
Threads...

Processes and Threads

- To distinguish above two characteristics..
- The unit of dispatching is referred to as a ***thread*** or lightweight process
- The unit of resource ownership is referred to as a process or ***task***

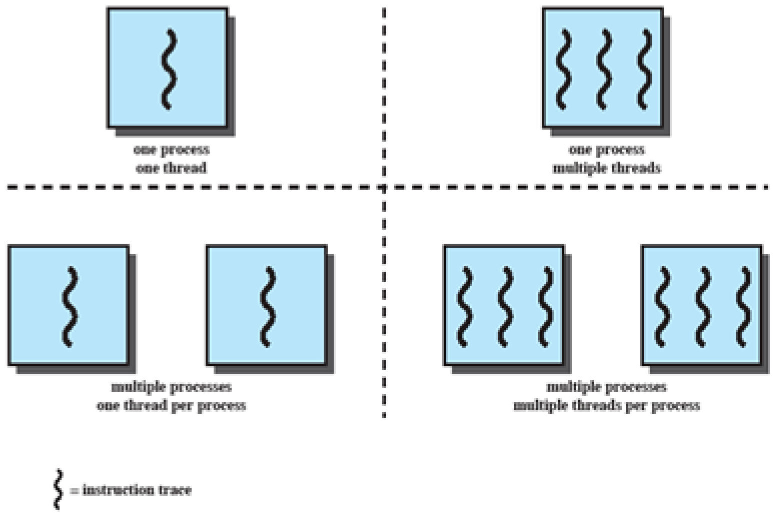
Multithreading

- The ability of an OS to support multiple, concurrent paths of execution within a single process.



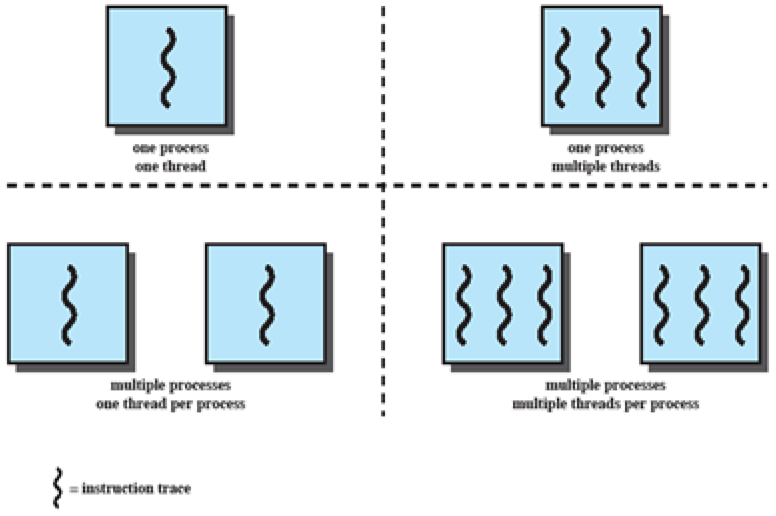
Single Thread Approaches

- MS-DOS supports a single user process and a single thread.
- Some UNIX, support multiple user processes but only support one thread per process



Multithreading

- Java run-time environment is a single process with multiple threads
- Multiple processes ***and*** threads are found in Windows, Solaris, and many modern versions of UNIX



Processes

- In a multithreaded environment, a process is defined as the unit of resource allocation and a unit of protection. Process is associated with...
- A virtual address space which holds the process image
- Protected access to
  - Processors,
  - Other processes,
  - Files,
  - I/O resources

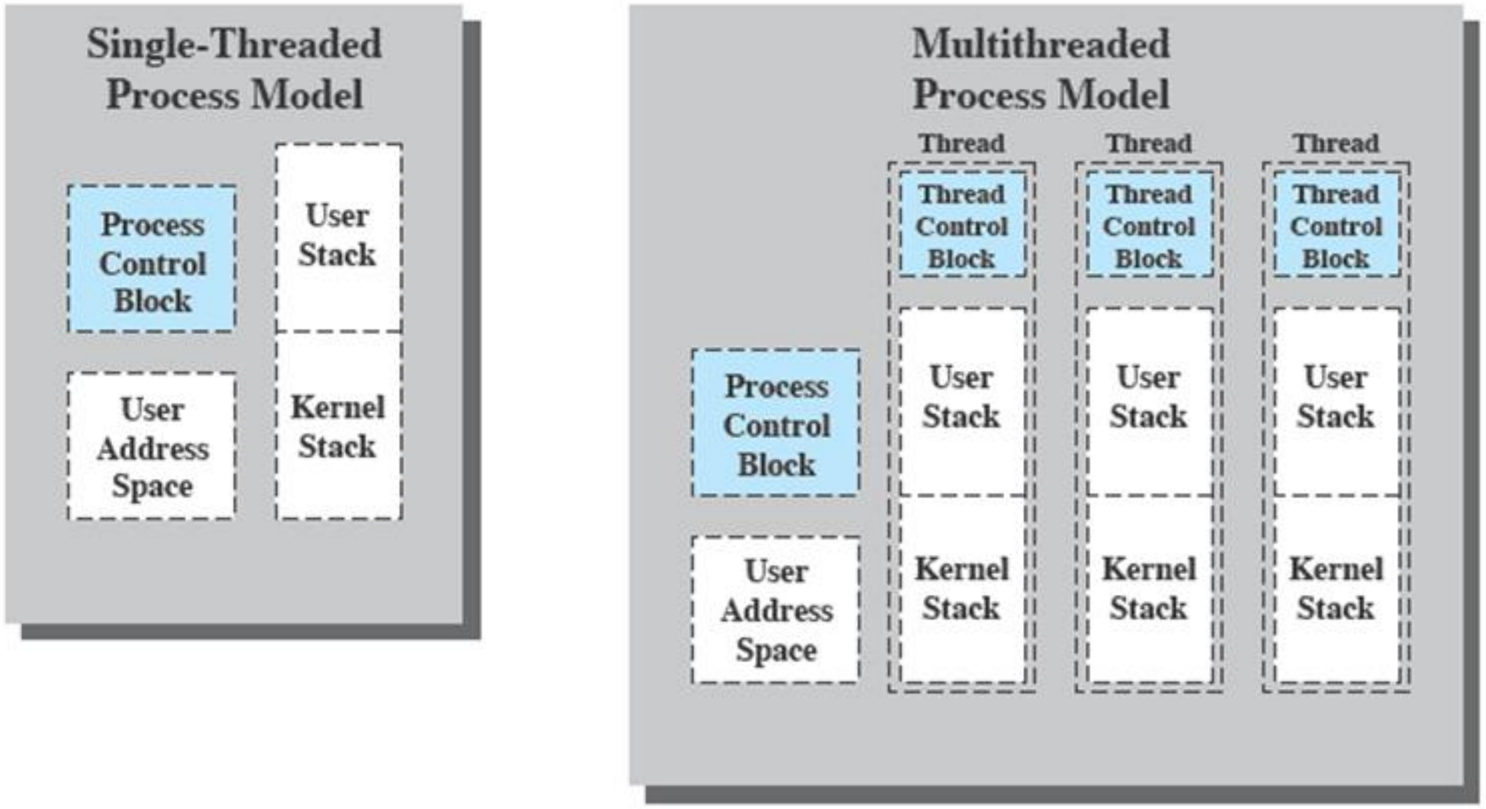
One or More Threads in Process

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

One view…

- *One way to view a thread is as an independent program counter operating **<u>within</u>** a process.*

Threads vs. processes



Distinction between threads and processes from the point of view of process management.

- In a single-threaded process model, the representation of a process includes
  - its process control block
  - user address space,
  - user and kernel stacks to manage the call/return behaviour of the execution of the process.

While the process is running, it controls the processor registers. The contents of these registers are saved when the process is not running.

In a multithreaded environment,

- there is still a single process control block and user address space associated with the process,
- **but** separate stacks for each thread,
- as well as a separate control block for each thread containing register values, priority, and other thread-related state information.

Thus, all of the threads of a process share the state and resources of that process.

- They reside in the same address space and have access to the same data.
- When one thread alters an item of data in memory, other threads see the results if and when they access that item.
- If one thread opens a file with read privileges, other threads in the same process can also read from that file.

Benefits of Threads

- Takes less time to create a new thread than a process
- Less time to terminate a thread than a process
- Switching between two threads takes less time that switching processes
- Threads can communicate with each other
  - without invoking the kernel
- If there is an application or function that should be implemented as a set of related units of execution, it is far more efficient to do so as a collection of threads - rather than a collection of separate processes.

Thread use in a Single-User System

- Foreground and background work e.g. spreadsheet
- Asynchronous processing e.g. protection against power failure within a word processor
- Speed of execution e.g. one thread computes one batch of date while other reading next batch from a device
- Modular program structure e.g. input/output

Threads

- Several actions that affect all of the threads in a process
  - The OS must manage these at the process level.
- Examples:
  - Suspending a process involves suspending all threads of the process (why??)
  - Termination of a process, terminates all threads within the process

Activities similar to Processes

- Threads have execution states and may synchronize with one another.
  - Similar to processes
- We look at these two aspects of thread functionality in turn.
  - States
  - Synchronisation

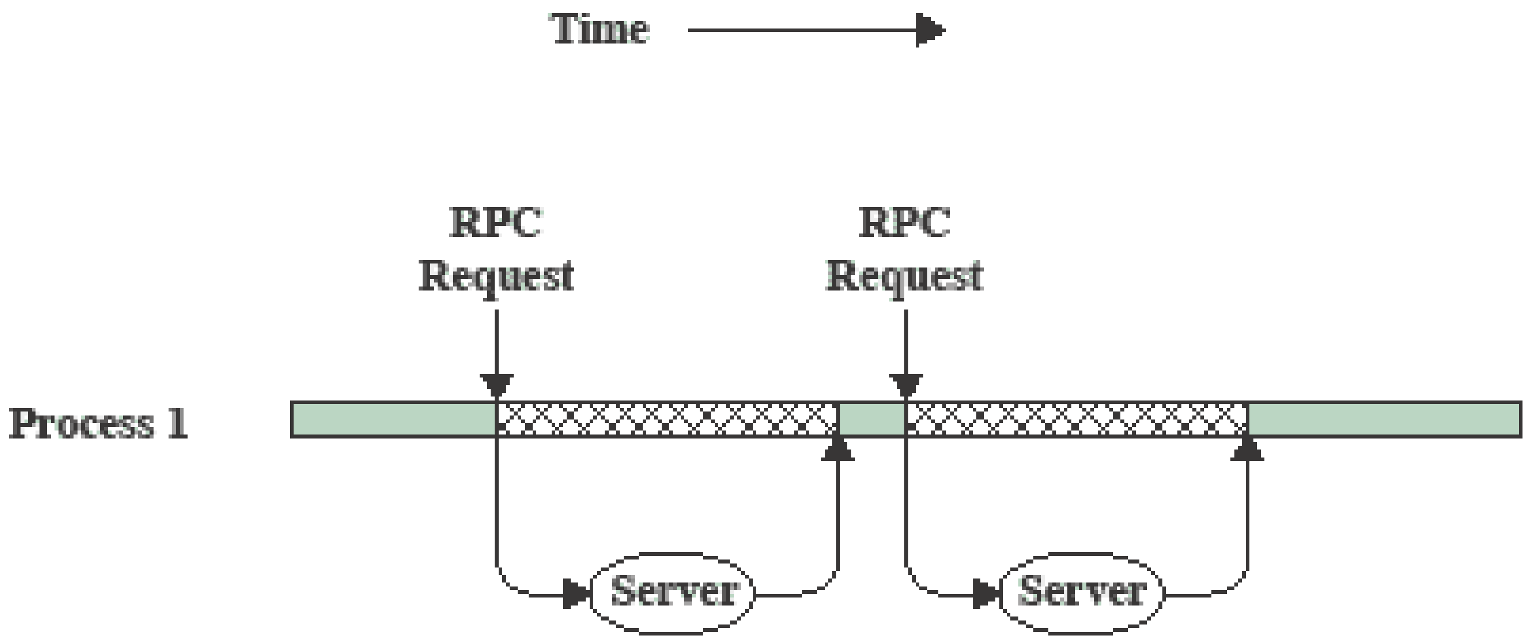
Thread Execution States

- States associated with a change in thread state
  - Spawn (another thread)
  - Block
    - Issue: will blocking a thread block other, or *all,* threads
  - Unblock
  - Finish (thread)
    - Deallocate register context and stacks

Example:
Remote Procedure Call

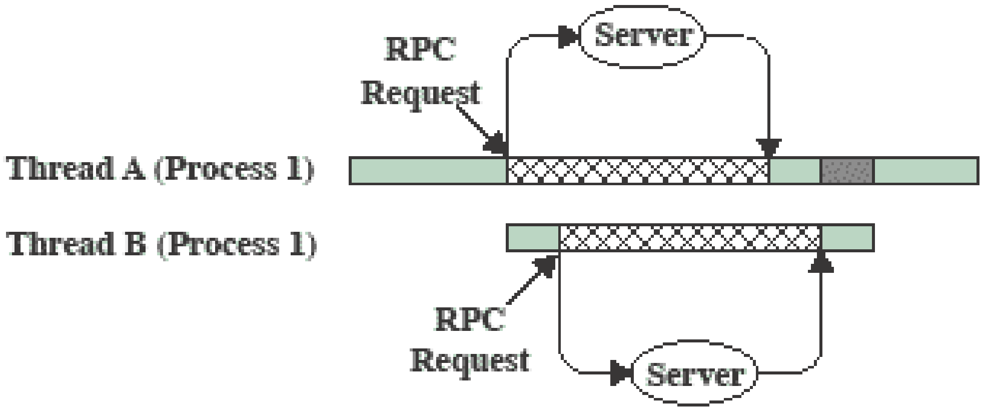
- Consider:
  - A program that performs two remote procedure calls (RPCs)
  - to two different hosts
  - to obtain a combined result.

RPC Using Single Thread



(a) RPC Using Single Thread

RPC Using One Thread per Server



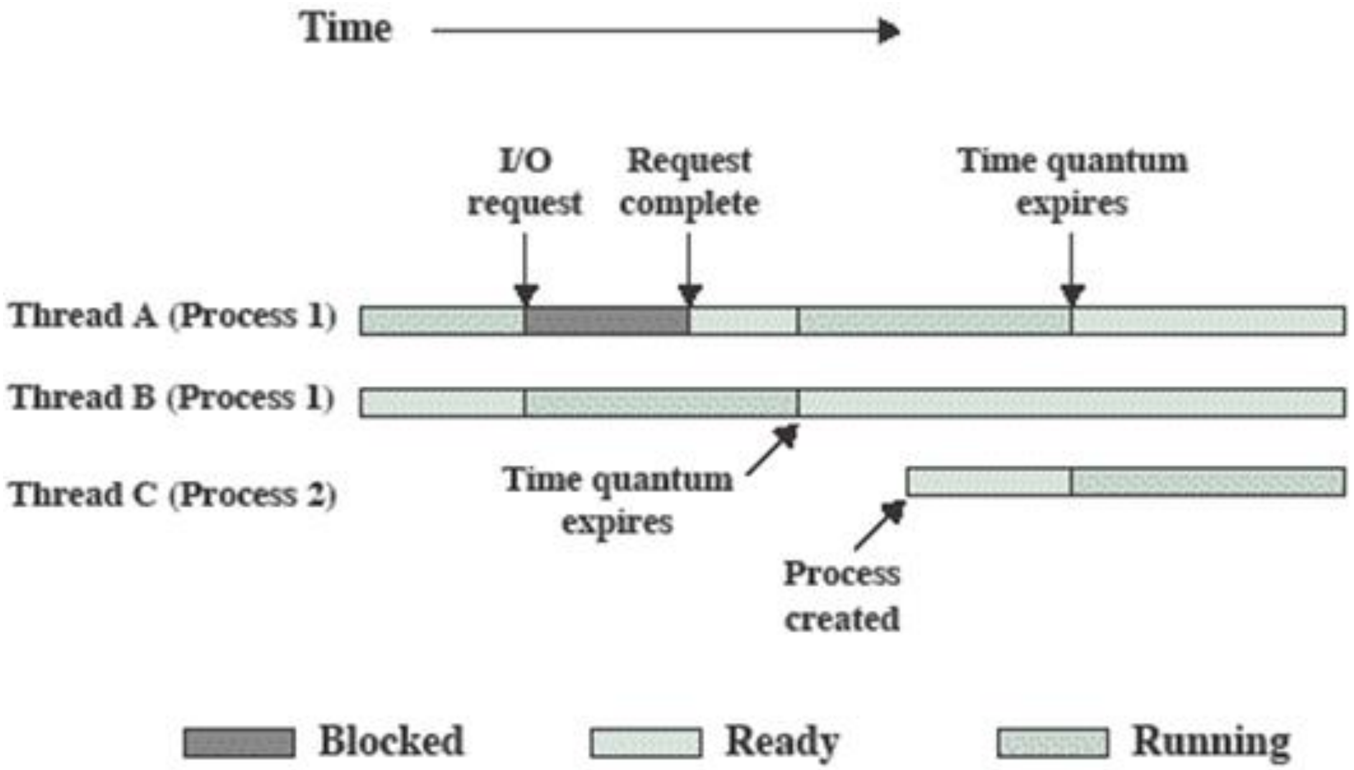
(b) RPC Using One Thread per Server (on a uniprocessor)

▨ Blocked, waiting for response to RPC

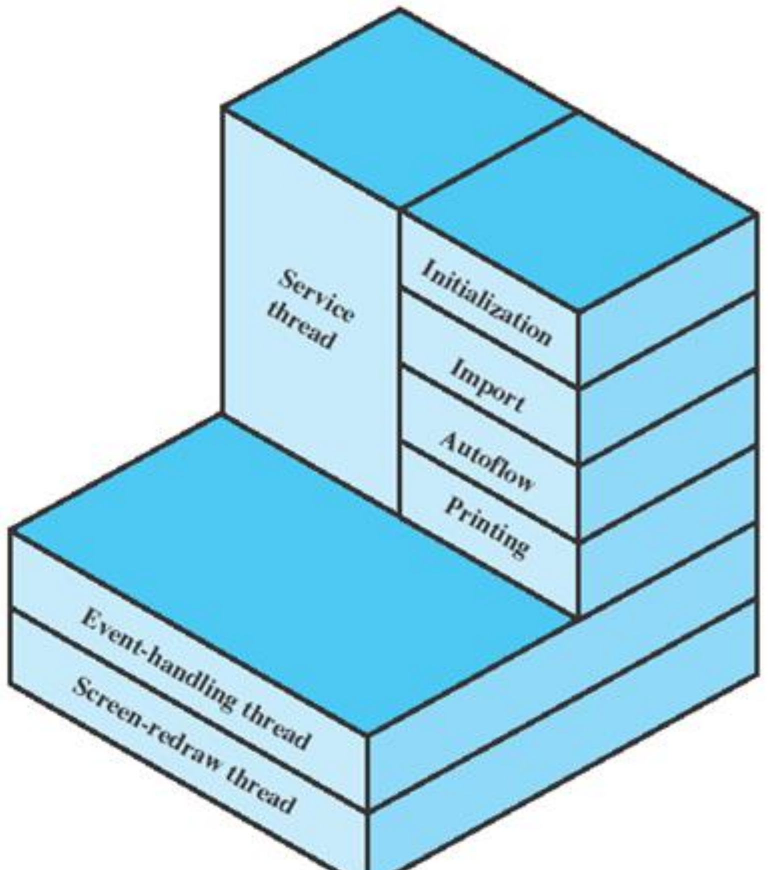
▩ Blocked, waiting for processor, which is in use by Thread B

🟩 Running

Multithreading on a Uniprocessor



Adobe PageMaker



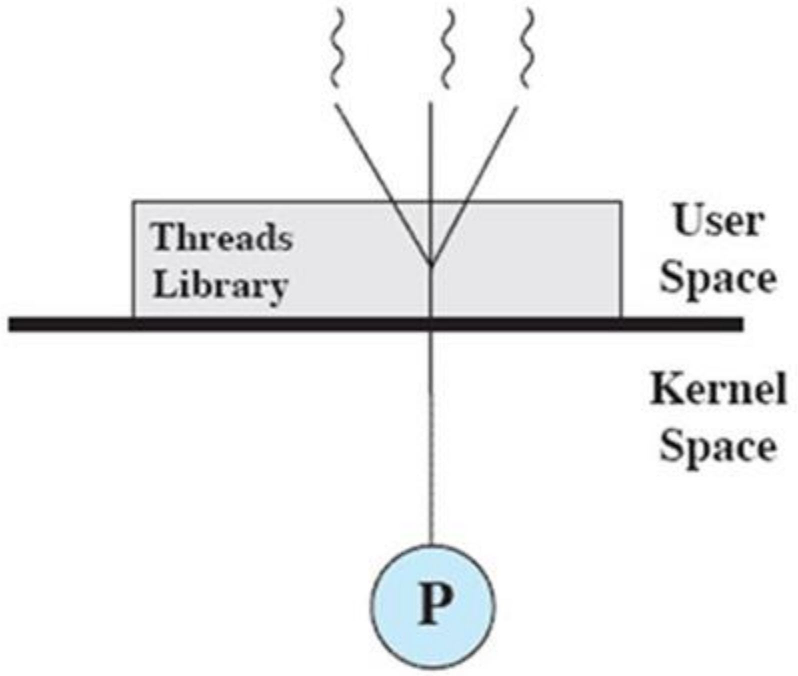
Categories of Thread Implementation

- User Level Thread (ULT)

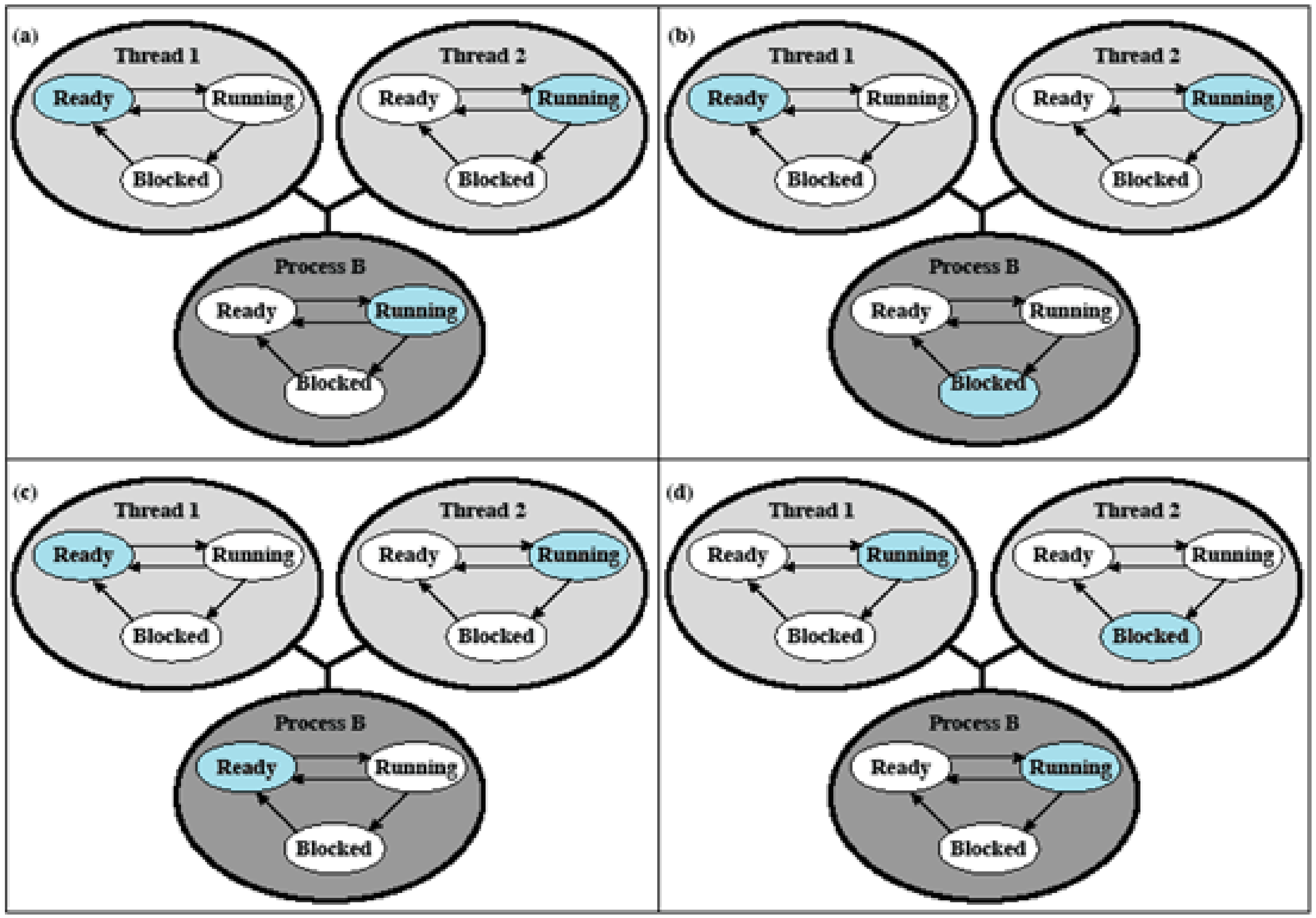
- Kernel level Thread (KLT) also called:
  - kernel-supported threads
  - lightweight processes.

User-Level Threads

- All thread management is done by the application
- The kernel is not aware of the existence of threads



Relationships between ULT Thread and Process States



Colored state
is current state

- Application executing in thread 2 makes a system call that blocks B
- Due to clock interrupt kernel determines that time slice is exhausted and places process B in ready state
- Thread 2 reached a point where it needs some action performed by thread 1 which blocks thread 2 and makes thread 1 running

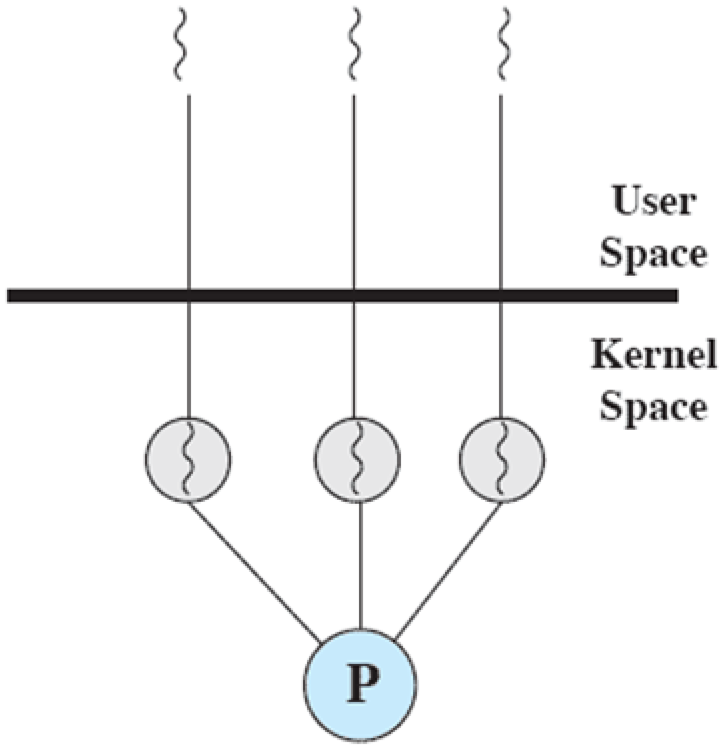
Advantages of ULT

- Thread switching does not require kernel mode privileges bcoz all thread management is within user address space
- Scheduling can be application specific
- ULTs can run on any OS. No changes are required to kernel to support ULTs
- The threads library is a set of application-level functions shared by all applications.

Disadvantages of ULTs

- In a typical OS, many system calls are blocking. When a ULT executes a system call, all of the threads within the process are blocked.
- In a pure ULT strategy, a multithreaded application can’t take advantage of multiprocessing

Kernel-Level Threads



- Kernel maintains context information for the process and the threads
  - No thread management done by application
- Scheduling is done on a thread basis
- Windows is an example of this approach

Advantages of KLT

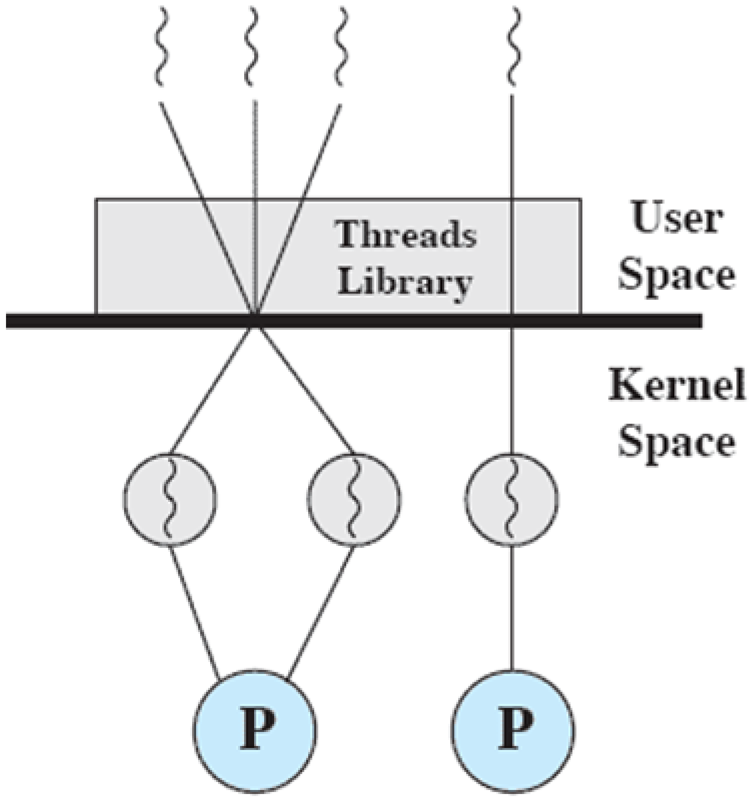
- The kernel can simultaneously schedule multiple threads from the same process on multiple processors.
- If one thread in a process is blocked, the kernel can schedule another thread of the same process.
- Kernel routines themselves can be multithreaded.

Disadvantage of KLT

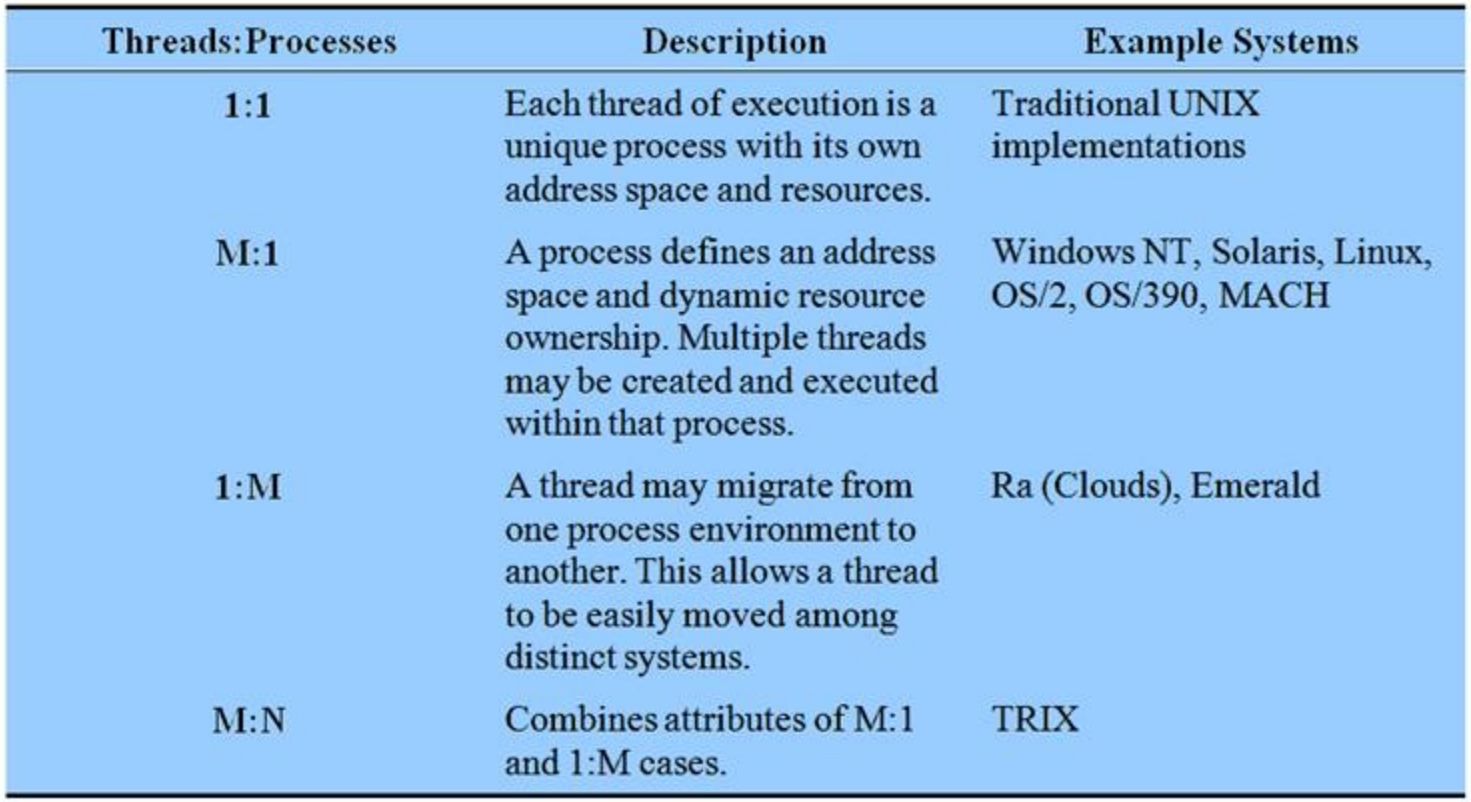
- The transfer of control from one thread to another within the same process requires a mode switch to the kernel

Combined Approaches

- Thread creation done in the user space
- Bulk of scheduling and synchronization of threads by the application
- Example is Solaris



Relationship Between Thread and Processes



Threads:Processes	Description	Example Systems
1:1	Each thread of execution is a unique process with its own address space and resources.	Traditional UNIX implementations
M:1	A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.	Windows NT, Solaris, Linux, OS/2, OS/390, MACH
1:M	A thread may migrate from one process environment to another. This allows a thread to be easily moved among distinct systems.	Ra (Clouds), Emerald
M:N	Combines attributes of M:1 and 1:M cases.	TRIX

Traditional View

- Traditionally, the computer has been viewed as a sequential machine.
  - A processor executes instructions one at a time in sequence
  - Each instruction is a sequence of operations
- Two popular approaches to providing parallelism
  - Symmetric MultiProcessors (SMPs)
  - Clusters (ch 16)

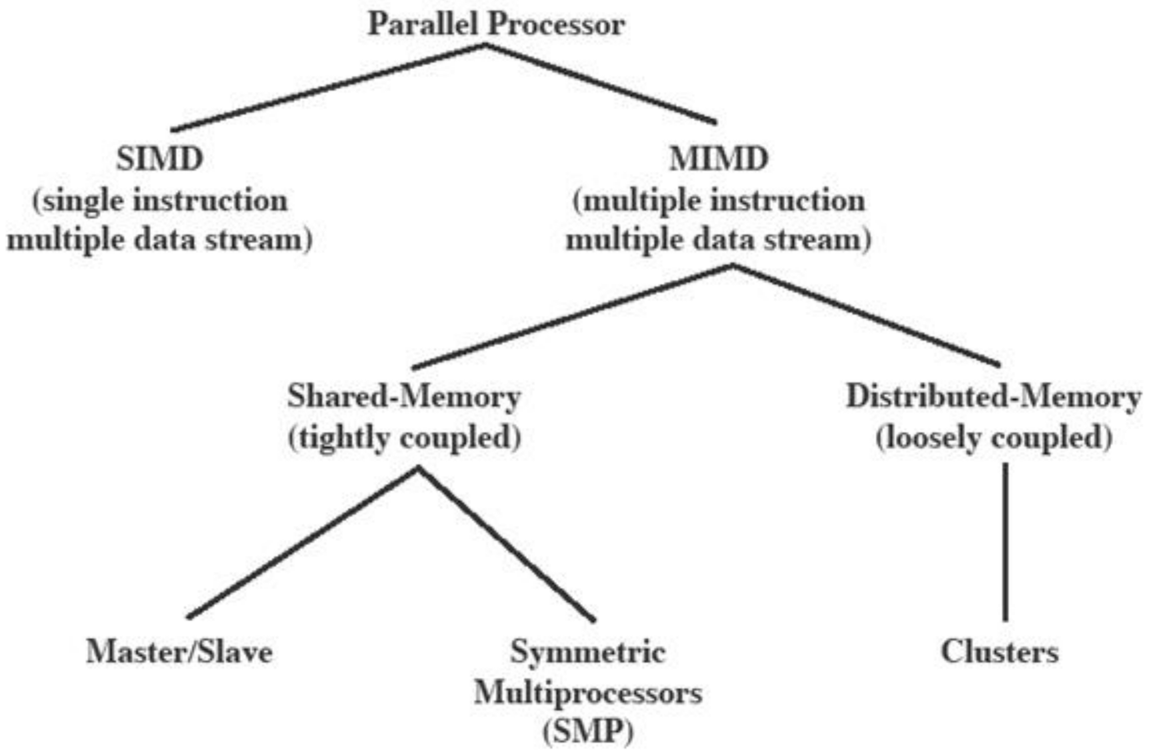
Categories of Computer Systems

- Single Instruction Single Data (SISD) stream
  - Single processor executes a single instruction stream to operate on data stored in a single memory
- Single Instruction Multiple Data (SIMD) stream
  - Each instruction is executed on a different set of data by the different processors

Categories of Computer Systems

- Multiple Instruction Single Data (MISD) stream (Never implemented)
  - A sequence of data is transmitted to a set of processors, each of execute a different instruction sequence
- Multiple Instruction Multiple Data (MIMD)
  - A set of processors simultaneously execute different instruction sequences on different data sets

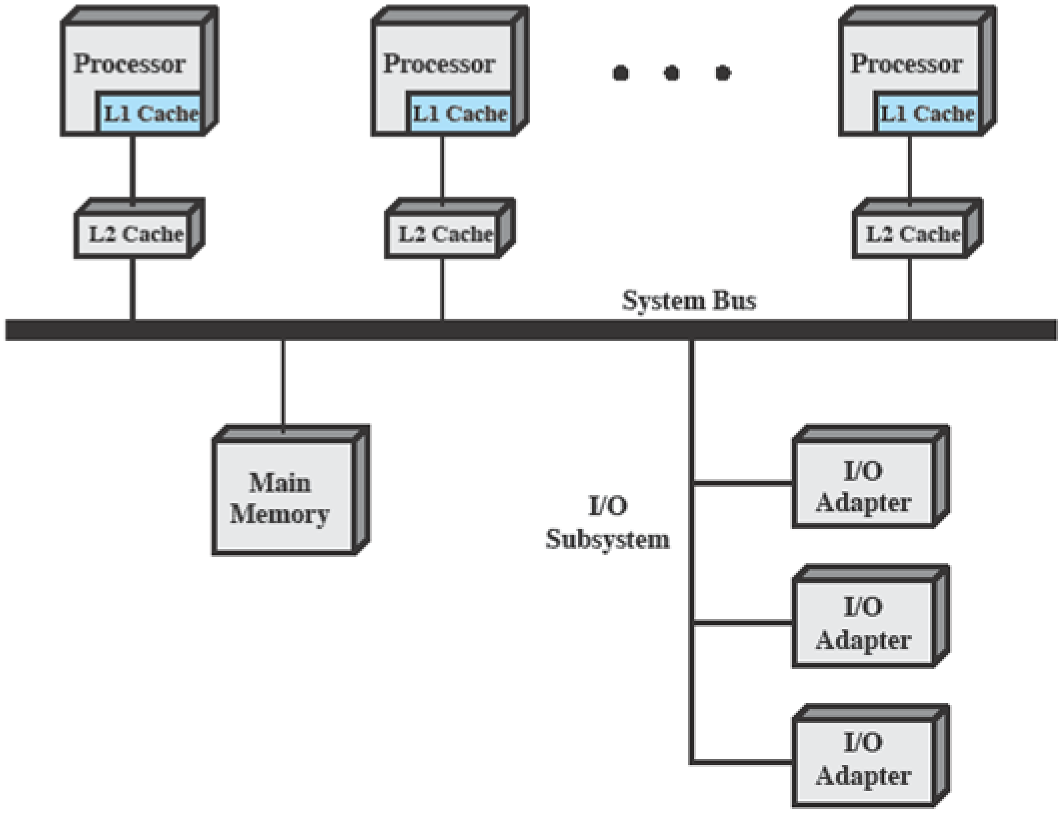
Parallel Processor Architectures


Parallel Processor
SIMD (single instruction multiple data stream)
MIMD (multiple instruction multiple data stream)
Shared-Memory (tightly coupled)
Distributed-Memory (loosely coupled)
Master/Slave
Symmetric Multiprocessors (SMP)
Clusters

Symmetric Multiprocessing

- Kernel can execute on any processor
  - Allowing portions of the kernel to execute in parallel
- Typically each processor does self-scheduling from the pool of available process or threads

Typical SMP Organization



Multiprocessor OS Design Considerations

- The key design issues include
  - Simultaneous concurrent processes or threads
  - Scheduling
  - Synchronization
  - Memory Management
  - Reliability and Fault Tolerance