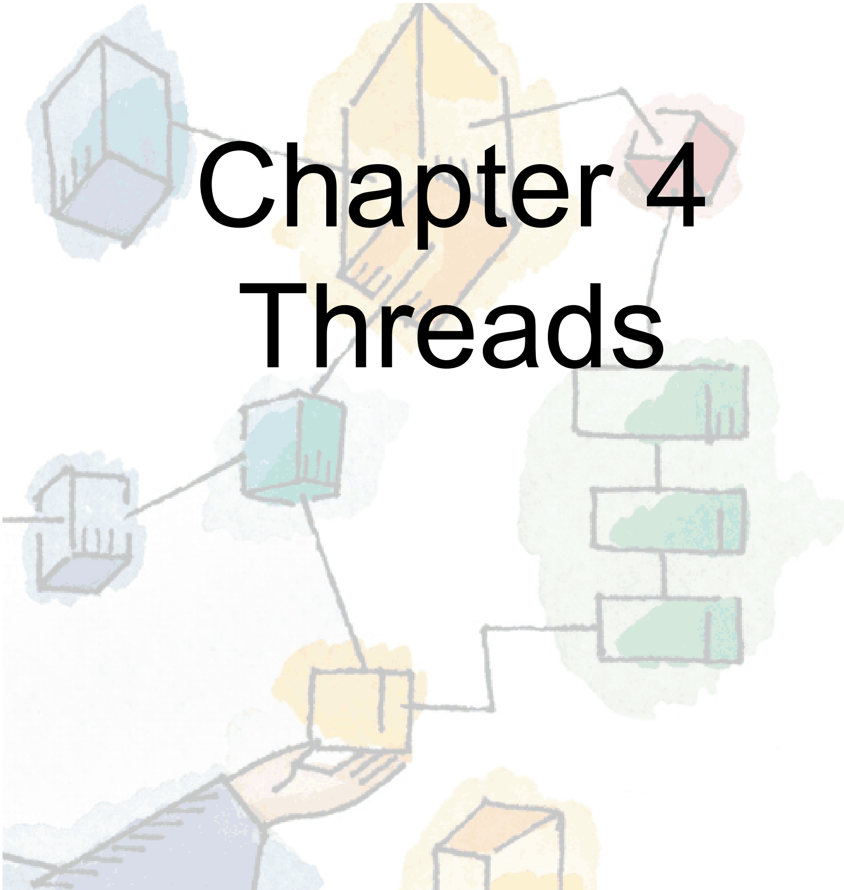
Operating Systems: Internals and Design Principles

William Stallings



Chapter 4 Threads

Roadmap

- Threads: Resource ownership and execution
- Categories of thread implementation
- Thread library
  - POSIX Threads (Pthreads)

Processes and Threads

- Processes have two characteristics:
  - **Resource ownership**
    - A process is allocated ownership of resources including a virtual address space to hold the process image
    - The OS performs a protection function to prevent unwanted interference between processes with respect to resources
  - **Dispatching/scheduling/execution**
    - The execution of a process follows an execution path that may be interleaved with other processes
    - A process has an execution state (Running, Ready, etc.) and a dispatching priority, and is the entity that is scheduled and dispatched by the OS

Multithreading

- These two characteristics can be treated independently by the OS
  - 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*** is the ability of an OS to support multiple, concurrent paths of execution within a single process.

Single-threaded Approaches

- A single thread of execution per process, in which the concept of a thread is not recognized.
  - MS-DOS supports a single-user process and a single thread
  - Some variants of UNIX support multiple user processes but only support one thread per process

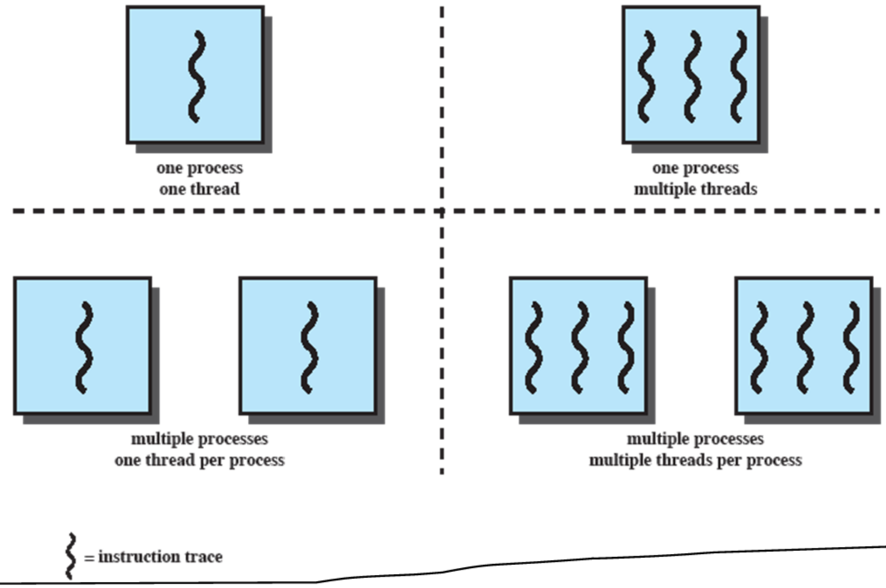


Figure 4.1 Threads and Processes [ANDE97]

Multithreaded Approaches

- A Java run-time environment is a system of one process with multiple threads.
- The use of multiple processes, each of which supports multiple threads are found in Windows, Solaris, and many modern versions of UNIX.

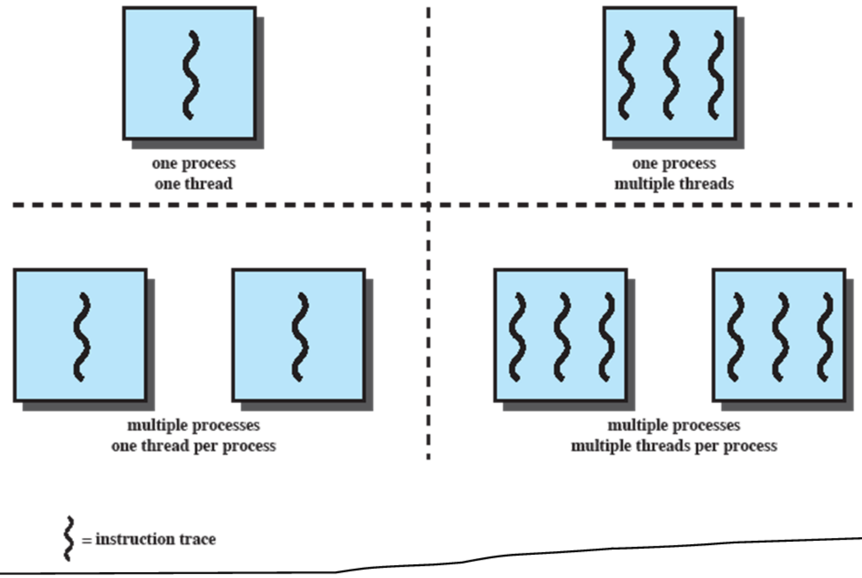
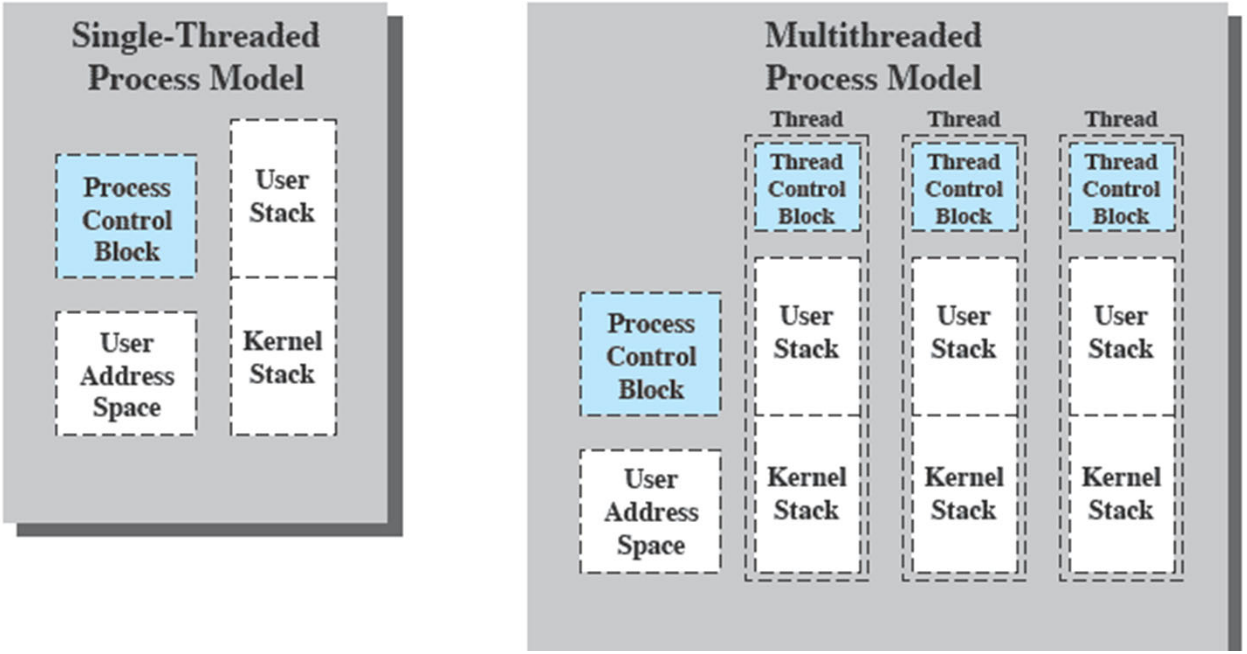


Figure 4.1 Threads and Processes [ANDE97]

Process vs. Thread (1)

- In an OS, a process is
  - A unit of resource allocation: a virtual address space that holds the ***process image*** (code + data + stack + PCB)
  - A unit of protection: protected access to processors, other processes (for inter-process communication), files, I/O resources
- In a process, each thread has
  - An execution state (running, ready, etc.)
  - A 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, shared by all threads in that process

Process vs. Thread (2)



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

Activities Similar to Processes

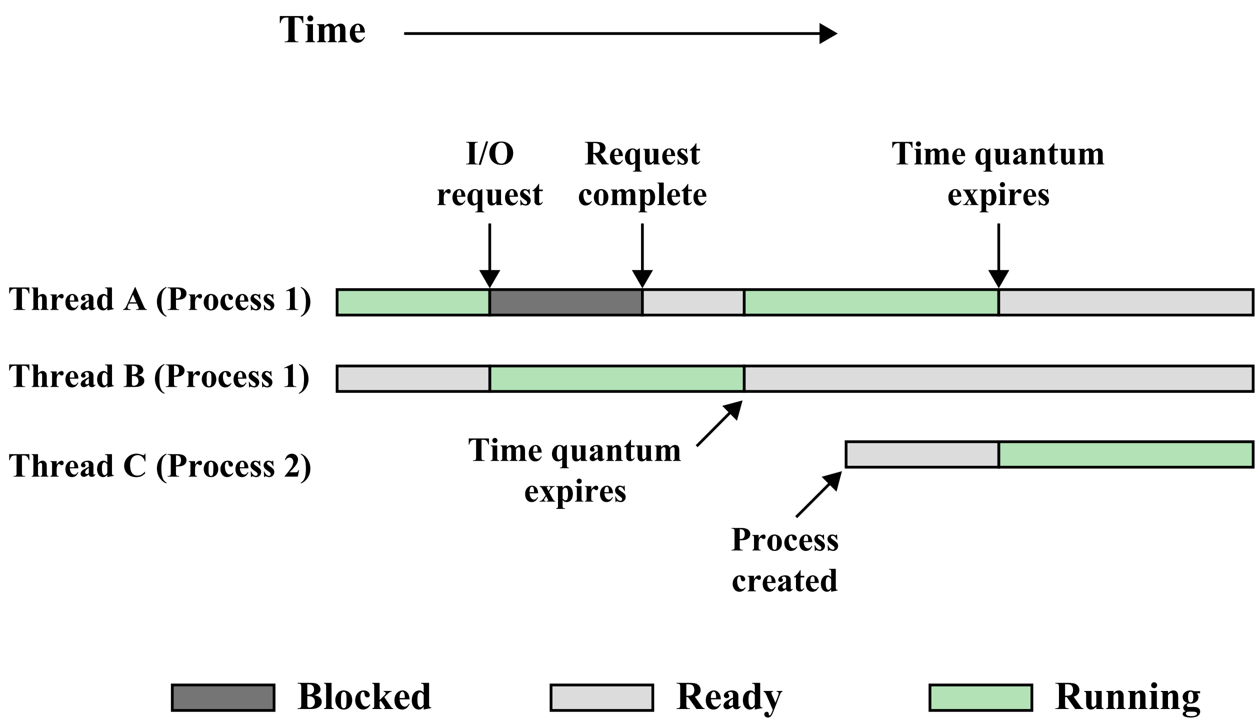
- Similar to processes, threads have *execution states* and need to *synchronize* with one another.
  - Execution states
    - *Reminder*: In an OS that supports threads, scheduling and dispatching is done on a thread basis.
    - Most of the state information dealing with execution is maintained in thread-level data structures.
    - The key states for a thread are: Running, Ready, Blocked.
    - Some states are at process-level.
      - Suspending a process involves suspending all threads of the process because they share the address space.
      - Termination of a process terminates all threads within the process.

Activities Similar to Processes

- Similar to processes, threads have *execution states* and need to *synchronize* with one another.
  - Threads need to synchronize with one another so that they don't interfere with each other or corrupt data structures.
    - All threads of a process share the same address space and other resources.
    - Any alteration of a resource by one thread affects the other threads in the same process.

Multithreading on a Uniprocessor

- Multiprogramming enables the interleaving of multiple threads within multiple processes.



Thread Use Examples

- Foreground and background work
  - In a spreadsheet program, one thread could display menus and read user input, while another thread executes user commands and updates the spreadsheet.
- Asynchronous processing
  - In a word processor, a thread can be created to do periodic backup to write its RAM buffer to disk once every minute by scheduling itself directly with the OS.
- Speed of execution
  - In computer graphics, matrix data can be divided and distributed into multiple threads to be calculated in parallel (on multiple cores)

Benefits of Threads

- If an application is 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. Reasons include:
  - 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 between processes
  - Threads enhance efficiency in communication because threads within the same process share memory and files, they can communicate with each other without invoking the kernel

Roadmap

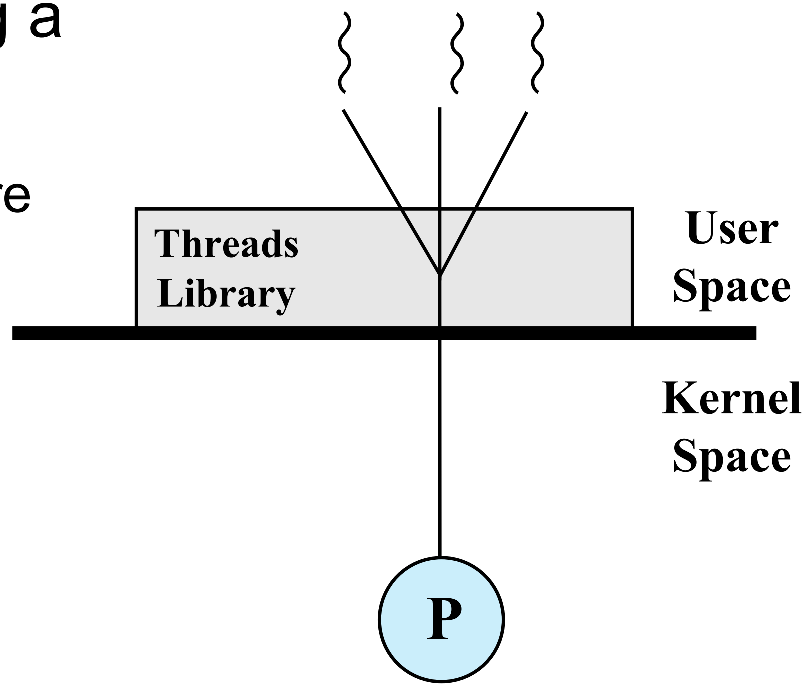
- Threads: Resource ownership and execution
- Categories of thread implementation
- Thread library
  - POSIX Threads (Pthreads)

Categories of Thread Implementation

- User Level Thread (ULT)
- Kernel level Thread (KLT), also called:
  - kernel-supported threads
  - lightweight processes

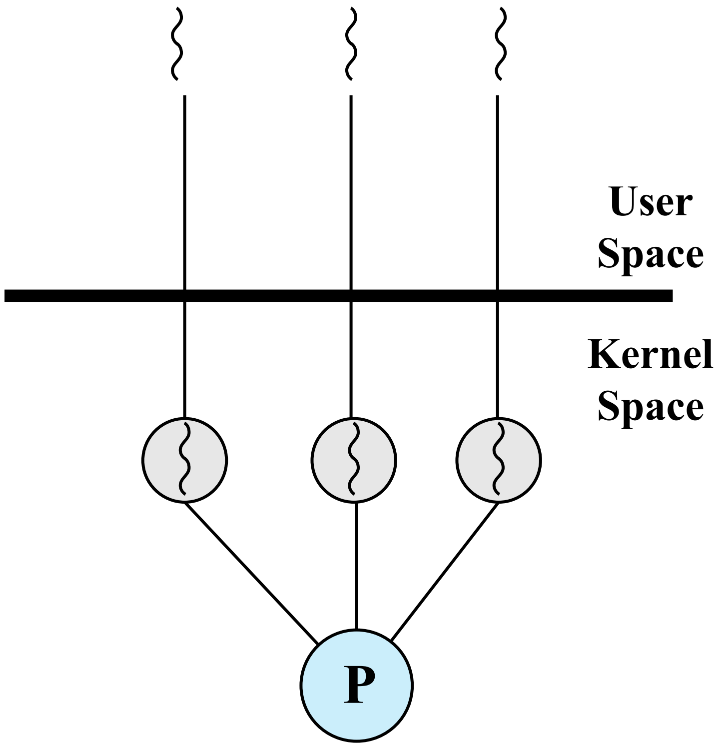
User-Level Threads

- All thread management is done within the application by calling a threads library.
  - The application and its threads are allocated to a single process managed by the kernel.
  - The kernel is not aware of the existence of threads.
  - Kernel scheduling is done on a process basis.



(a) Pure user-level

Kernel-Level Threads



(b) Pure kernel-level

- Thread management is done by the kernel.
  - No thread management done by application, just an API to the kernel thread facility.
  - Each user-level thread is mapped to a kernel-level thread.
  - Kernel maintains context information for the whole process and individual threads within the process.
  - Scheduling is done on a thread basis.
- Example: Windows

Pros & Cons (ULT)

- Pros
  - Process does not switch to the kernel mode to do thread management → saves the overhead of two mode switches.
  - Scheduling can be application specific.
  - Can run on any OS because the threads library is a set of application-level functions.
- Cons
  - Only a single thread within a process can execute at a time → a multithreaded application cannot take advantage of multiprocessing.
  - When a ULT executes a blocking system call, all of the threads within the process are blocked.

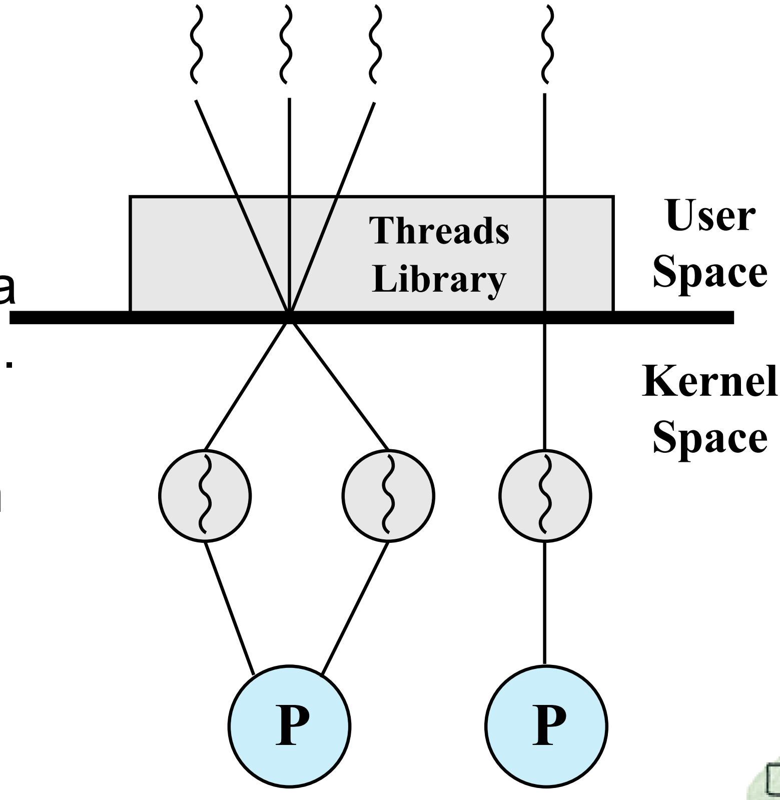
Pros & Cons (KLT)

- Pros
  - The kernel can simultaneously schedule multiple threads from the same process onto 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.
- Cons
  - The transfer of control from one thread to another within the same process requires a mode switch to the kernel.
  - Managing KLTs is slower than ULTs.
  - KLT implementation needs OS support.

Operation	User-Level Threads	Kernel-Level Threads	Processes
Null Fork	34	948	11,300
Signal Wait	37	441	1,840

Combined Approach

- m-to-n mapping hybrid implementation
  - Application creates *m* ULTs.
  - OS provides pool of *n* KLTs.
  - Multiple ULTs are mapped onto a smaller or equal number of KLTs.
  - Multiple threads within the same application can run in parallel on multiple processors.
  - A blocking system call need not block the entire process.
  - Example: Solaris



(c) Combined

Roadmap

- Threads: Resource ownership and execution
- Categories of thread implementation
- **Thread library**
  - – POSIX Threads (Pthreads)
    - Source: https://computing.llnl.gov/tutorials/pthreads/

Thread Libraries

- **Thread library** provides programmer with API (application program interface) for creating and managing threads.
- Three main thread libraries are in use today:
  - POSIX Pthreads
  - Win32
  - Java
- UNIX and Linux systems often use Pthreads.

Pthreads (POSIX Threads)

- Historically, hardware vendors have implemented their own proprietary versions of threads.
- For UNIX systems, a standardized C language ***threads programming interface*** has been specified by the IEEE POSIX 1003.1c standard (*Portable Operating System Interface*).
  - Implementations that adhere to this standard are referred to as **POSIX threads**, or **Pthreads**.
  - Most hardware vendors now offer Pthreads in addition to their proprietary API's.

What are Pthreads?

- Pthreads are defined as a set of C language programming types and procedure calls.
- Implemented with a **`pthread.h`** header/include file and a thread library.
- This makes it easy for programmers to develop *portable* threaded applications.

The Pthreads API

- The subroutines which comprise the Pthreads API can be informally grouped into four major groups:
  - ***Thread management***
    - Create, detach, join
    - Set/query thread attributes
  - ***Mutexes***
    - Deal with synchronization via a "***mutex***" (mutual exclusion)
  - ***Condition variables***
    - Address communications between threads that share a mutex
  - ***Synchronization***
    - Manage read/write locks and barriers

The Pthreads API

- Some of the thread-management function calls

Thread call	Description
Pthread_create	Create a new thread
Pthread_exit	Terminate the calling thread
Pthread_join	Wait for a specific thread to exit
Pthread_yield	Release the CPU to let another thread run
Pthread_attr_init	Create and initialize a thread's attribute structure
Pthread_attr_destroy	Remove a thread's attribute structure

- **Naming conventions**
  - All identifiers in the threads library begin with **pthread_**

Multithreading Consequences

- Because threads within the same process share resources
  - Changes made by one thread to shared system resources (such as closing a file) will be seen by all other threads
  - Two pointers having the same value point to the same data
  - Reading and writing to the same memory locations is possible
  - No guarantee as to the order that threads will run
  - Therefore requires explicit *synchronization* by the programmer