Operating Systems

Chapter 4 Threads, SMP, and Microkernels

Agenda

- 4.1 Processes and Threads
- 4.2 Types of Threads

4.1 Processes and Threads

- 4.1.1 Introduction
- 4.1.2 Multithreading
- 4.1.3 Thread Functionality

4.1.1 Introduction(1/2)

- These two characteristics are treated independently by the operating system　OS 区别对待以下问题
  - Resource ownership(资源所有权) - process includes a virtual address space to hold the process image
  - Scheduling/execution(调度 / 执行)- follows an execution path that may be interleaved with other processes

4.1.1 Introduction(2/2)

- Scheduling/execution
  - Dispatching is referred to as a thread or lightweight process(调度的单位称为线程或轻量进程)

- Resource
  - Resource of ownership is referred to as a process or task(资源所有权的单位称为进程或者任务)

4.1 Processes and Threads

- 4.1.1 Introduction
- 4.1.2 Multithreading
- 4.1.3 Thread Functionality

4.1.2 Multithreading(1/9)

- Single-Thread(单线程)
  - *Single-threaded approach* refers to the traditional approach of a single thread of execution per process, in which the concept of a thread is not recognized （单线程指一个进程中只有一个线程在执行的传统方法，线程的概念并不明确)
    - MS-DOS supports a single thread(单进程、单线程)
    - Some UNIX supports multiple user processes but only supports one thread per process （ 多进程，每个进程单线程)

4.1.2 Multithreading(2/9)

- Multithreading(多线程)
  - *Multithreading* refers to the ability of an OS to support multiple threads of execution within a single process(指操作系统支持在一个进程中执行多个线程的能力)
    - Windows, OpenEuler,Solaris, Linux, Mach, and OS/2 support multiple threads

4.1.2 Multithreading(3/9)

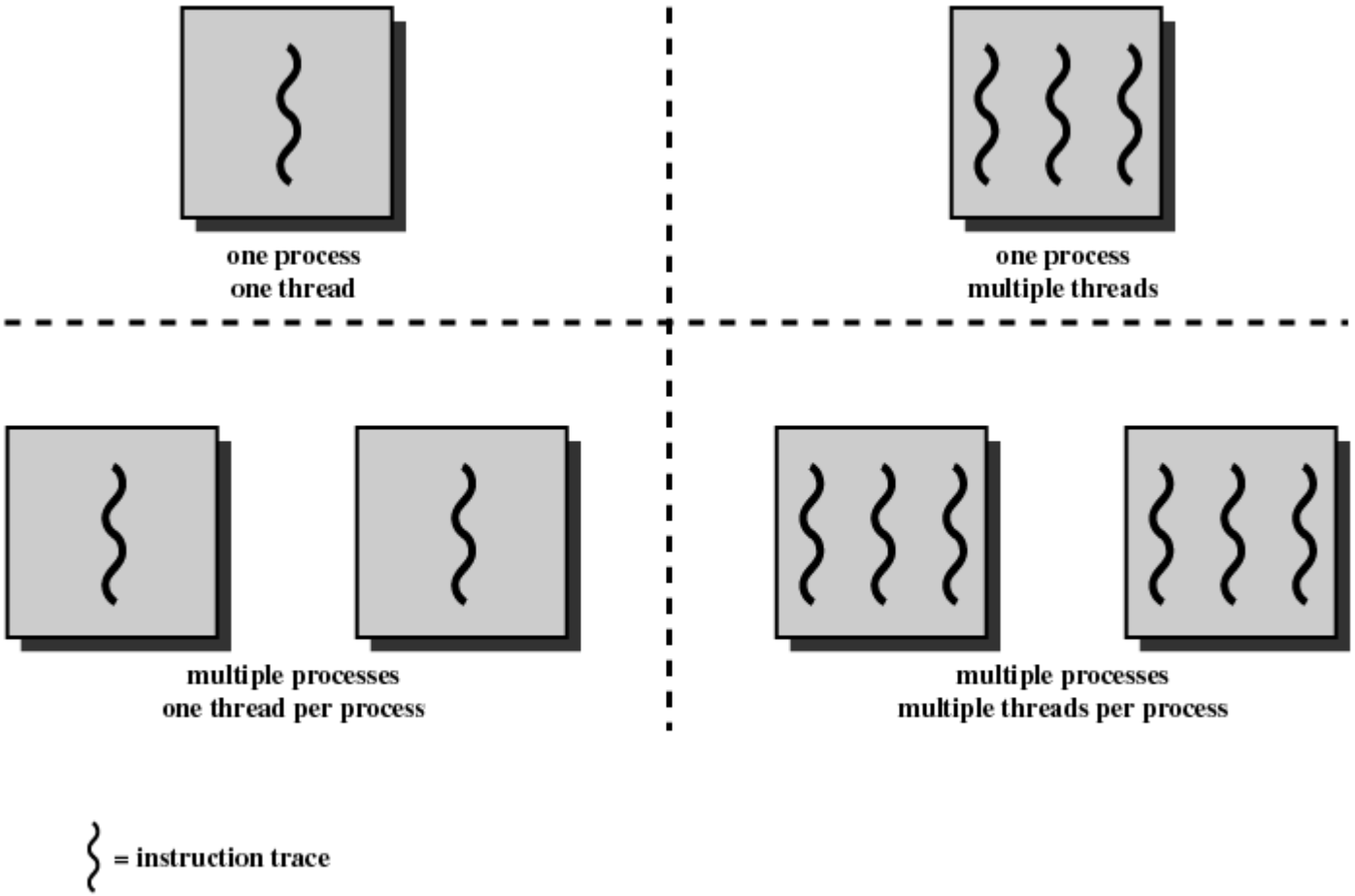


Figure 4.1 Threads and Processes [ANDE97]

4.1.2 Multithreading(4/9)

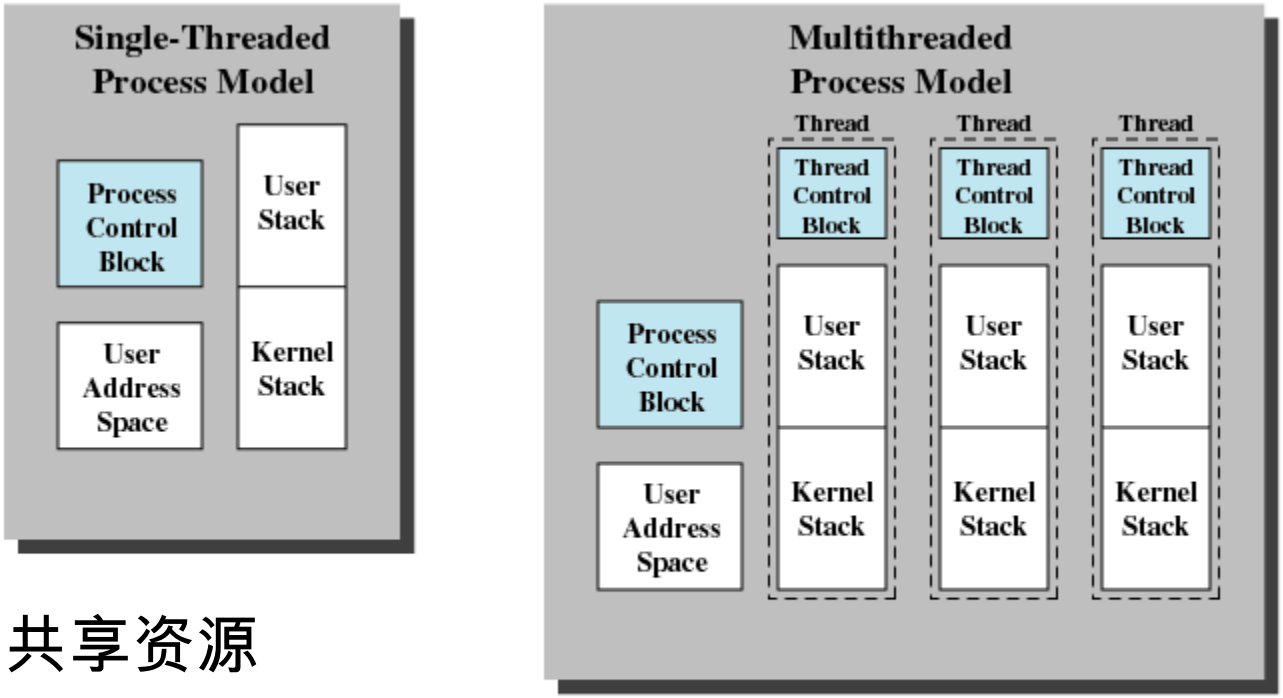
- **Process :** Unit of Resource allocation and Unit of Protection(资源分配和保护的单位)
  - Have a virtual address space(虚拟地址空间) which holds the process image(code, data, stack and PCB)
  - Protected access to processors, memory, other processes(与其他进程通信), files, and I/O resources
  - Contains one or more threads

4.1.2 Multithreading(5/9)

- **Thread :** Unit of Scheduling/Execution(调度执行的单位)
  - Each thread has:
    - An execution state (running, ready, etc.)
    - Saved thread context when not running (pc)
    - Has an execution stack
    - Some per-thread static storage for local variables 局部变量
    - Access to the memory and resources of its process 比如全局变量

4.1.2 Multithreading(6/9)

Distinction Between Threads and Processes From the Point of View of Process Management (从进程管理的角度看线程和进程)

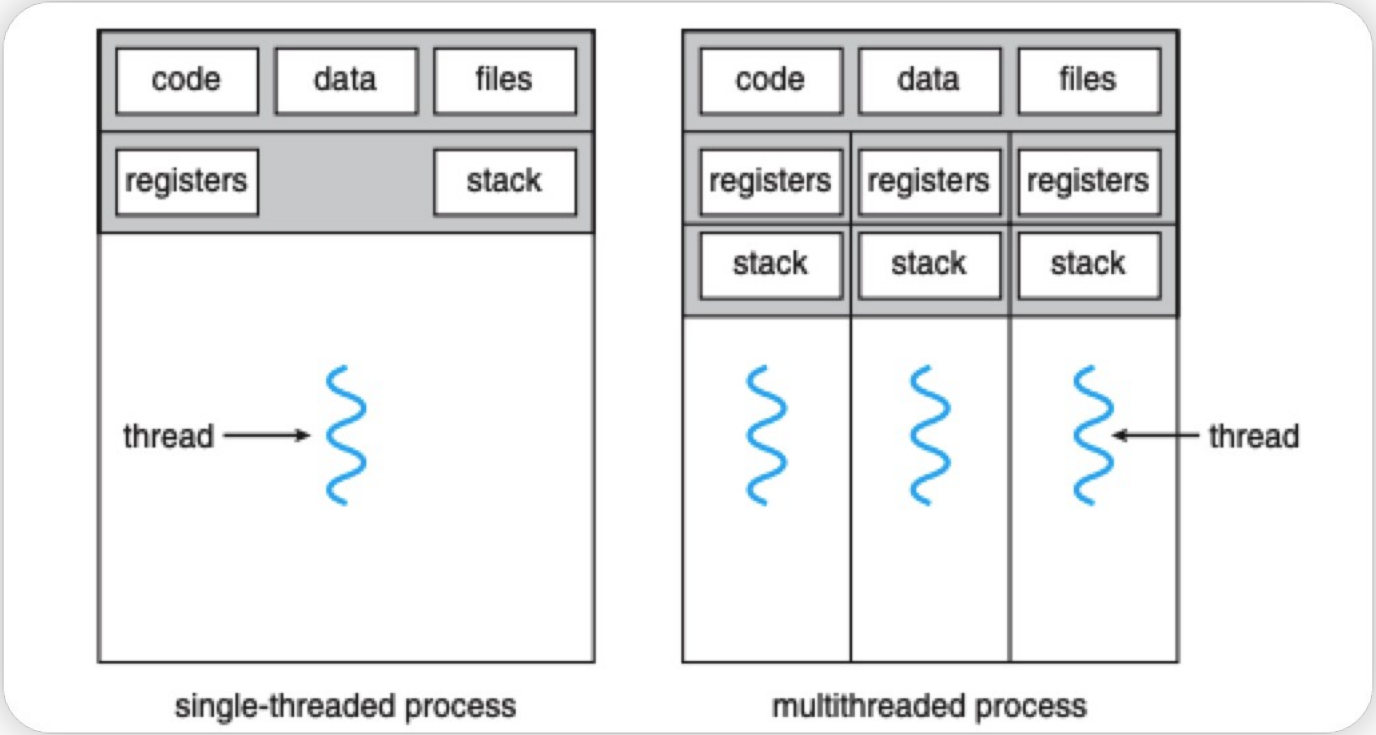


线程 = 进程 - 共享资源

Figure 4.2 Single Threaded and Multithreaded Process Models

4.1.2 Multithreading(7/9)

多线程共享进程的代码段和数据段 (全局变量) ，每个线程有自己的程序计数值和局部变量 stack



4.1.2 Multithreading(8/9)

Benefits(优点) of Threads

1. Takes less time to create a new thread than a process (创建快) 不重头分配内存等资源
2. Less time to terminate a thread than a process (结束快)
3. Less time to switch between two threads within the same process (切换快) 相同镜像，不需要切换页表
4. Since threads within the same process share memory and files, they can communicate with each other without invoking the kernel (通信快) 不经过内核通信

4.1.2 Multithreading(9/9)

- Threads are Affected by Many Process Action
  - Suspending a process involves suspending all threads of the process since all threads share the same address space (挂起进程会挂起该进程的所有线程)
  - Termination of a process, terminates all threads within the process (终止进程会终止该进程的所有线程)

4.1 Processes and Threads

- 4.1.1 Introduction
- 4.1.2 Multithreading
- 4.1.3 Thread Functionality (线程功能特性)

4.1.3 Thread Functionality(1/5)

- key states for a thread
  - Running, Ready,Blocked.

- Operations associated with a change in thread state
  - Spawn(派生)
    - Spawn another thread
  - Block(阻塞)
  - Unblock(解除阻塞)
  - Finish
    - Deallocate register context and stacks

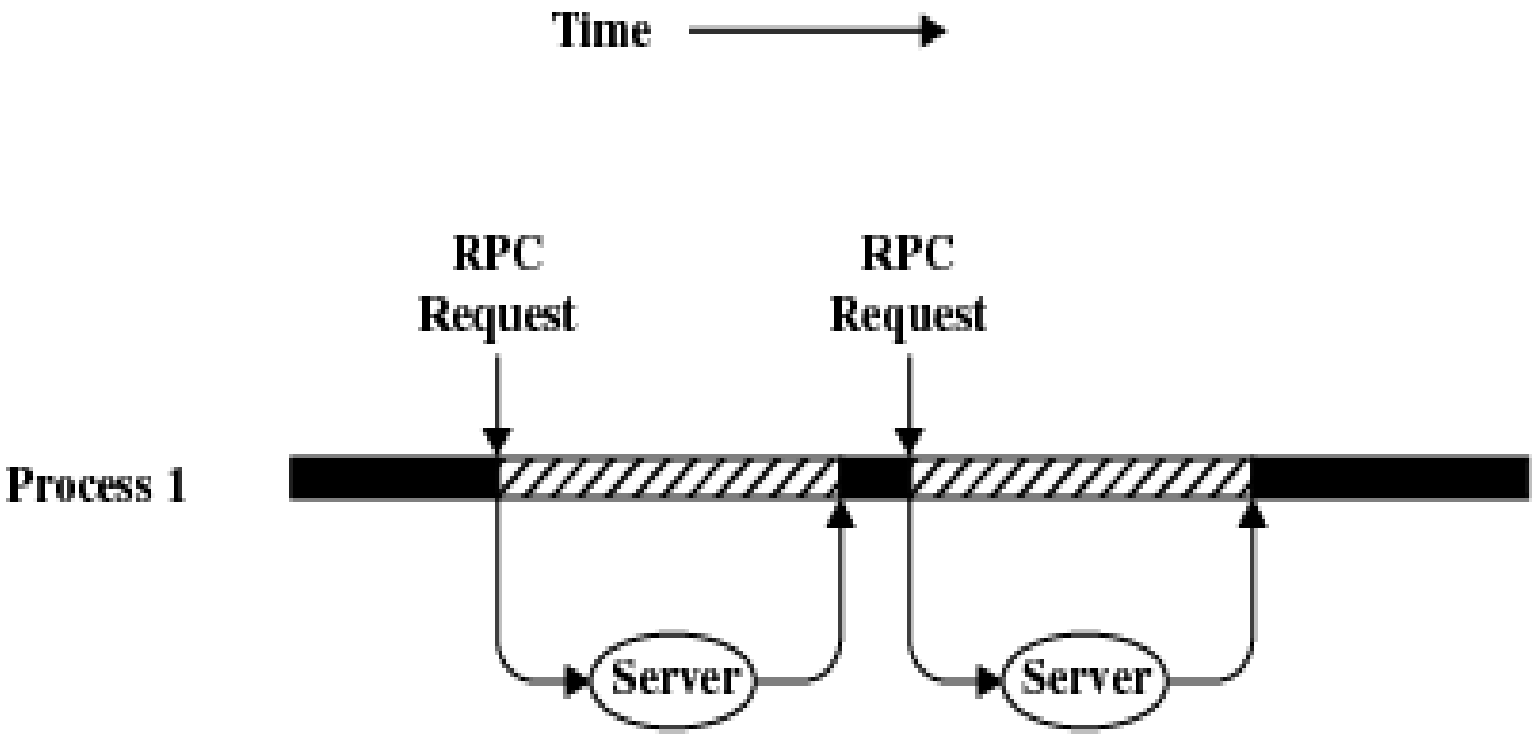
4.1.3 Thread Functionality(2/5)

- Suspending a process involves suspending all threads of the process
  - swap address space to disk

- Blocking a thread involves blocking the process , right ?
  - Better not

4.1.3 Thread Functionality(3/5)

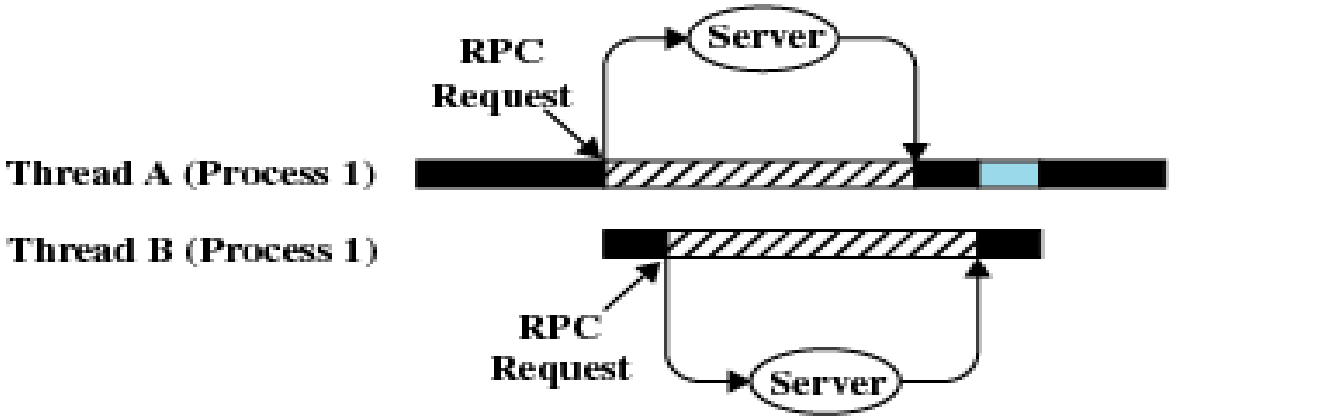
Remote Procedure Call Using Single Thread



(a) RPC Using Single Thread

4.1.3 Thread Functionality(4/5)

Remote Procedure Call Using Two Threads



(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

Figure 4.3 Remote Procedure Call (RPC) Using Threads

4.1.3 Thread Functionality(5/5)

- THREAD Synchronization 线程同步
- Why?

- In a process:
  - All threads share the same address space and other resources(open files).
  - Any alteration of a resource by one thread affects the others.

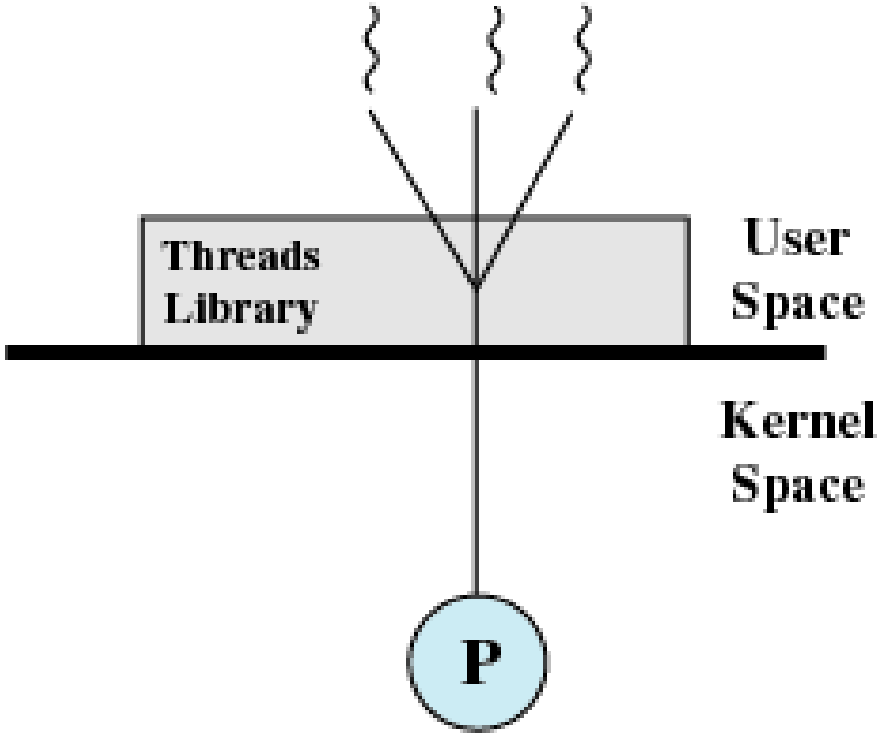
Agenda

- 4.1 Processes and Threads
- 4.2 Types of Threads
  - 4.2.1 User-LeveIntroduction
  - 4.2.2 Kernel-Level Thread
  - 4.2.3 Combined Approaches
  - 4.2.4 Other Arrangements

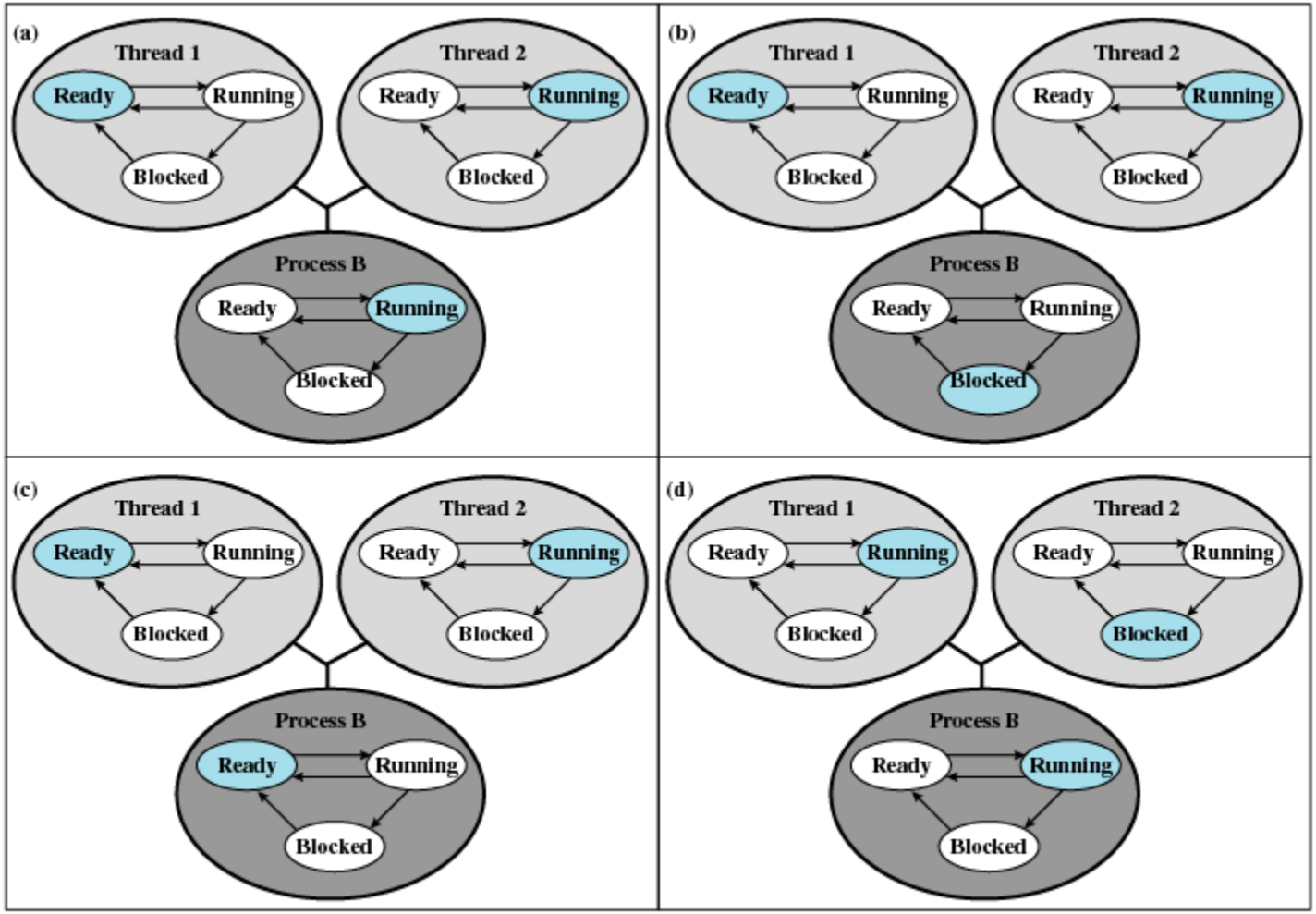
4.2 Types of Threads(1/10)

1. User-Level Threads (ULT ，用户级线程)

- Multithread implemented by a threads library 线程库
- All thread management is done by the application
- The kernel is not aware of the existence of threads & scheduling is done on a process basis



4.2 Types of Threads(2/10)



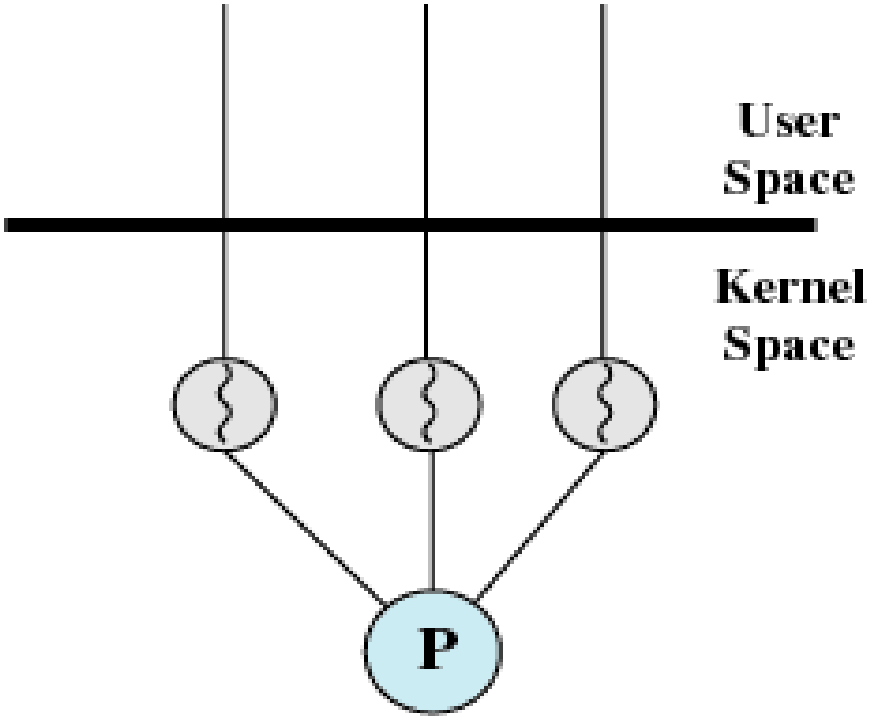
Colored state
is current state

Figure 4.7 Examples of the Relationships Between User-Level Thread States and Process States

4.2 Types of Threads(3/10)

2.Kernel-Level Threads(内核级线程)

- Kernel maintains context information for the process and the threads
- Scheduling is done on a thread basis



(b) Pure kernel-level

4.2 Types of Threads(4/10)

- Advantages of ULT to KLT
  - Less overhead of thread switches(mode switches do not required) 切换开销小
  - Scheduling can be application specific 调度策略根据应用可以不同
  - ULTs can run on any operating system without modify the underlying kernel 无需底层内核修改

4.2 Types of Threads(5/10)

- Disadvantages of ULT to KLT
  - One thread is blocked, all other threads of the process are blocked(ULT 按进程调度)
  - A multithreaded application cannot take advantage of multiprocessing (线程不能分配到多核)
  - Ways to work around these drawbacks:
    - Multiple processes 用多进程代替多线程
    - Jacketing 套管 针对阻塞问题

4.2 Types of Threads(6/10)

- KLT
  - Kernel manage the threads
    - Windows provides API
  - Kernel routines themselves can be multithreaded

- Advantages of KLT to ULT
  - Overcomes the two principal drawbacks of the ULT
    - Multiple threads in one process can simultaneously run on multiple processors 多 CPU 执行
    - One threads blocked cannot make the other threads within the same process blocked 仅阻塞单个线程

4.2 Types of Threads(7/10)

- Disadvantages of KLT to ULT
  - The principal disadvantage is that thread switch requires mode switches(模式切换) to the kernel

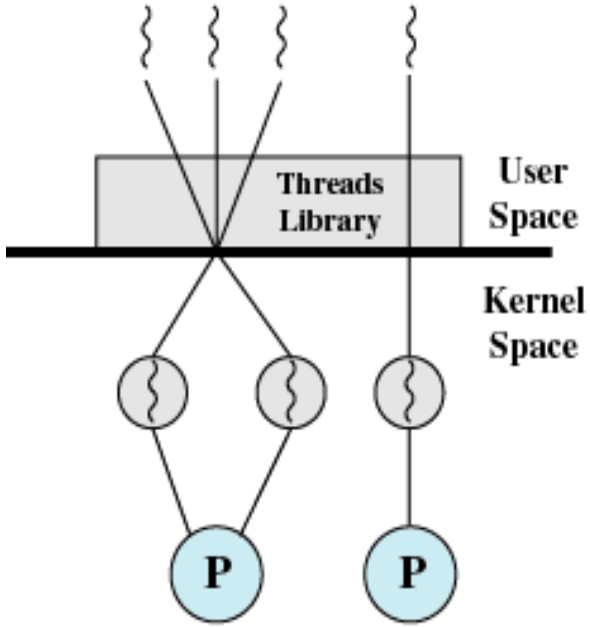
Table 4.1 Thread and Process Operation Latencies (μs)

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

4.2 Types of Threads(8/10)

3. Combined Approaches 混合方式

- thread creation/scheduling/synchronization is done completely in user space 分配调度同步可以在用户空间
- The multiple ULTs from a single application are mapped onto some (smaller or equal) number (adjustable) of KLTs. 允许映射到内核



(c) Combined

4.2 Types of Threads(9/10)

- Advantages of Combined Approaches
  - multiple threads within the same application run in parallel on multiple processors 多 cpu 运行
  - A blocking system call only blocks the thread. 仅阻塞单个线程

4.2 Types of Threads(10/10)

Relationship Between Threads and Processes

Table 4.2 Relationship Between Threads 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