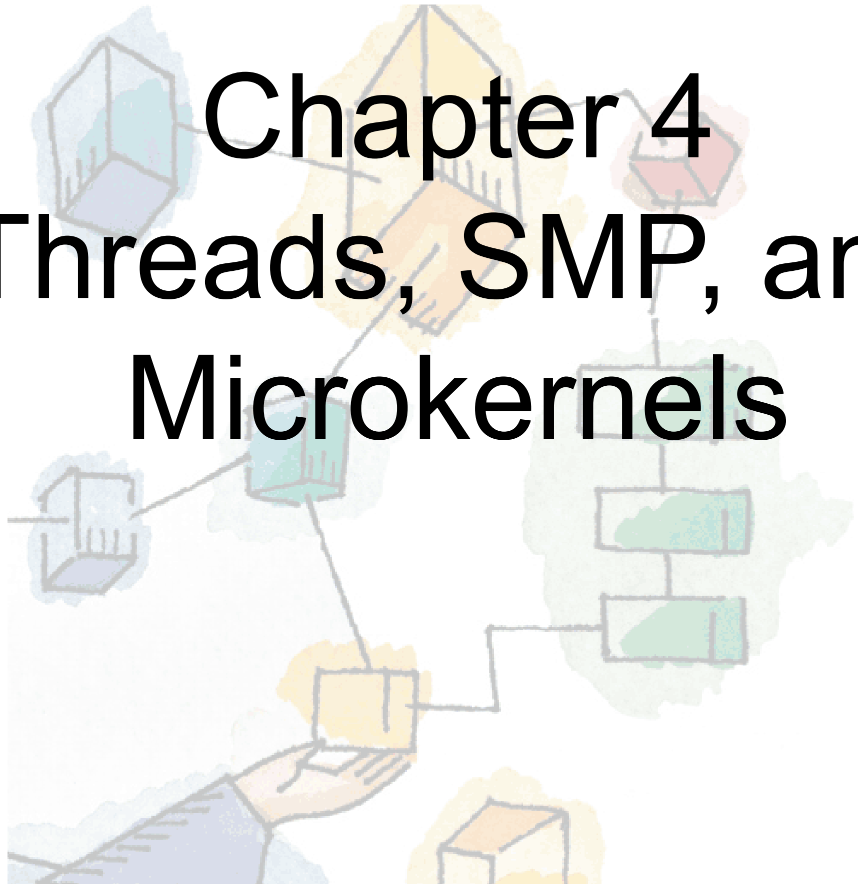
Operating Systems:
Internals and Design Principles, 6/E
William Stallings



Chapter 4 Threads, SMP, and Microkernels

Dave Bremer
Otago Polytechnic, N.Z.

Roadmap

- Threads: Resource ownership and execution
- Symmetric multiprocessing (SMP).
- Microkernel
- Case Studies of threads and SMP:
  - Windows
  - Solaris
  - Linux

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

Processes and Threads

- 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.

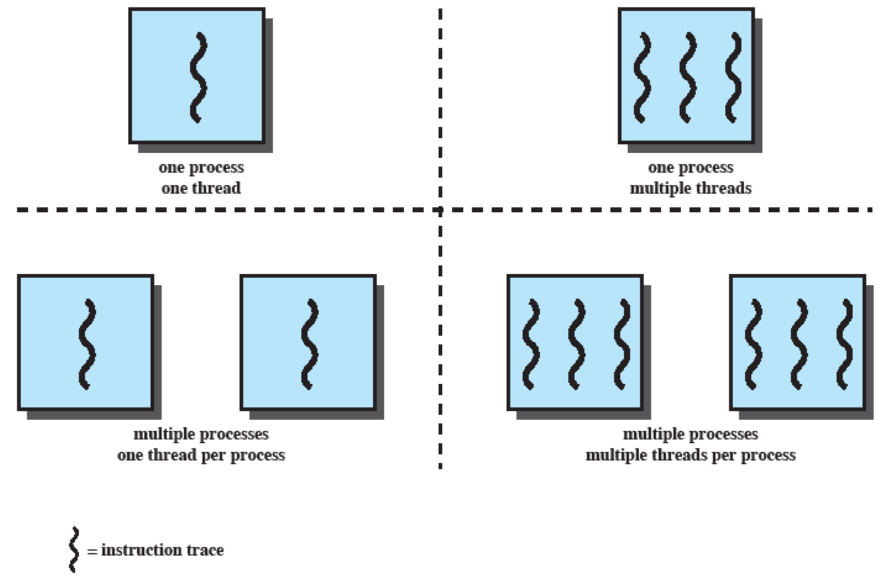


Figure 4.1 Threads and Processes [ANDE97]

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

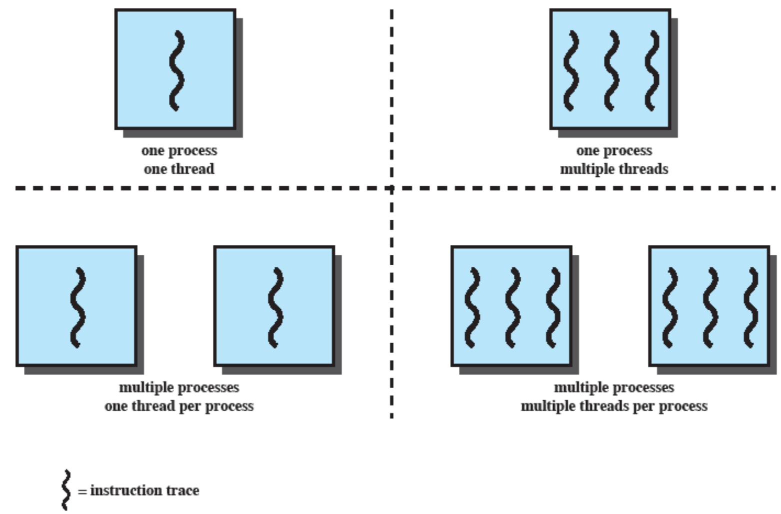


Figure 4.1 Threads and Processes [ANDE97]

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

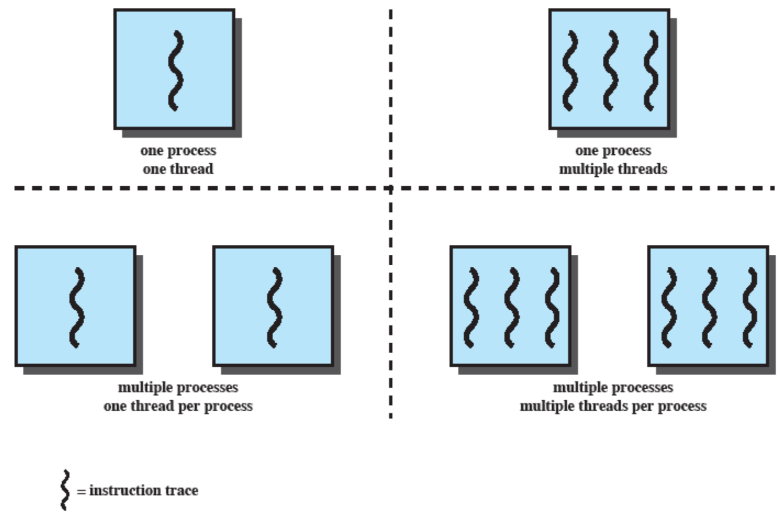


Figure 4.1 Threads and Processes [ANDE97]

Processes

- 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

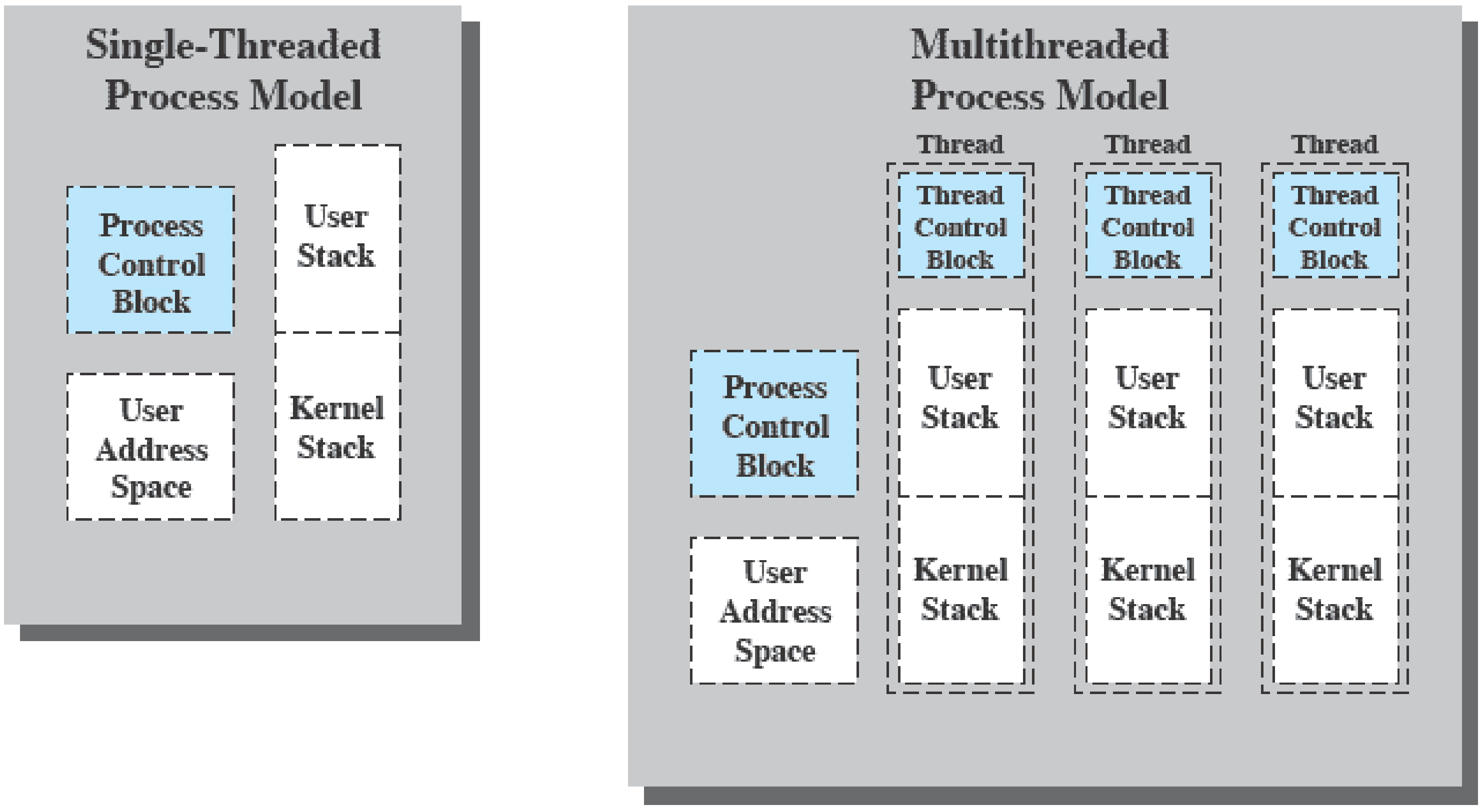


Figure 4.2 Single Threaded and Multithreaded Process Models

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

Thread use in a Single-User System

- Foreground and background work
- Asynchronous processing
- Speed of execution
- Modular program structure

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
  - 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

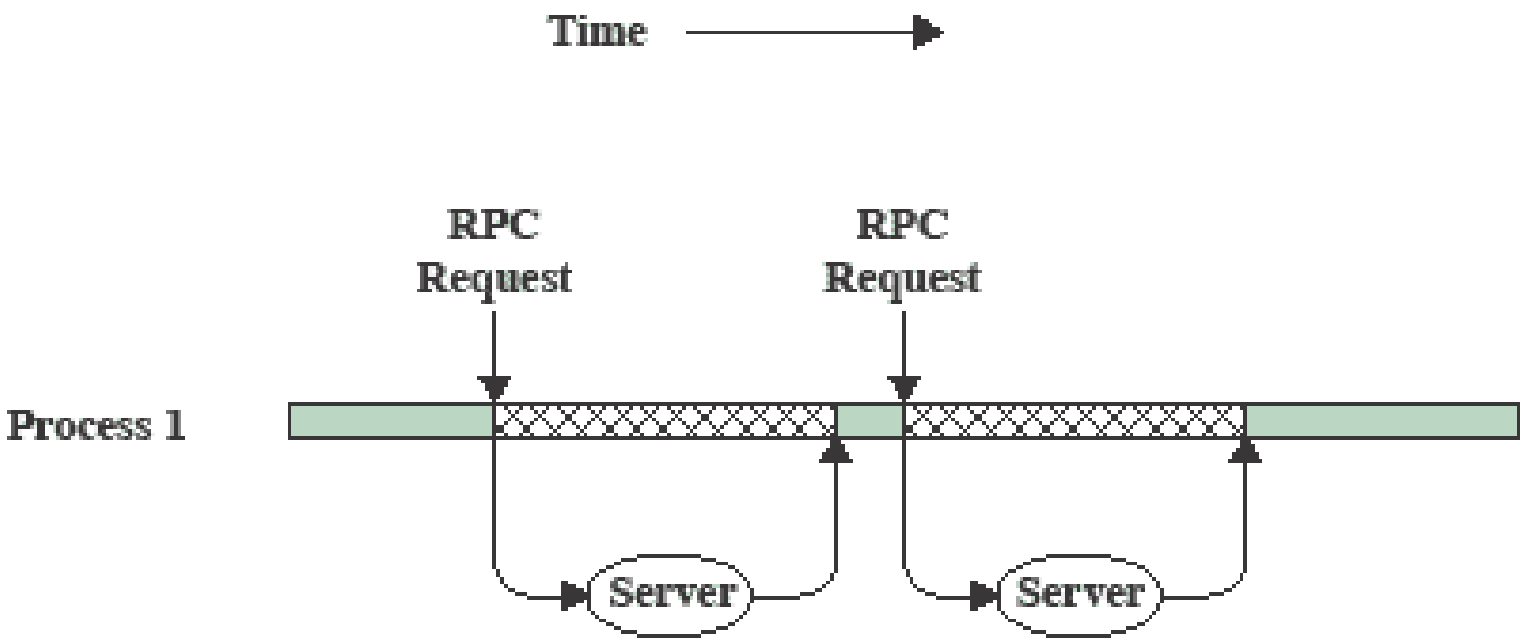
Thread Execution States

- 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

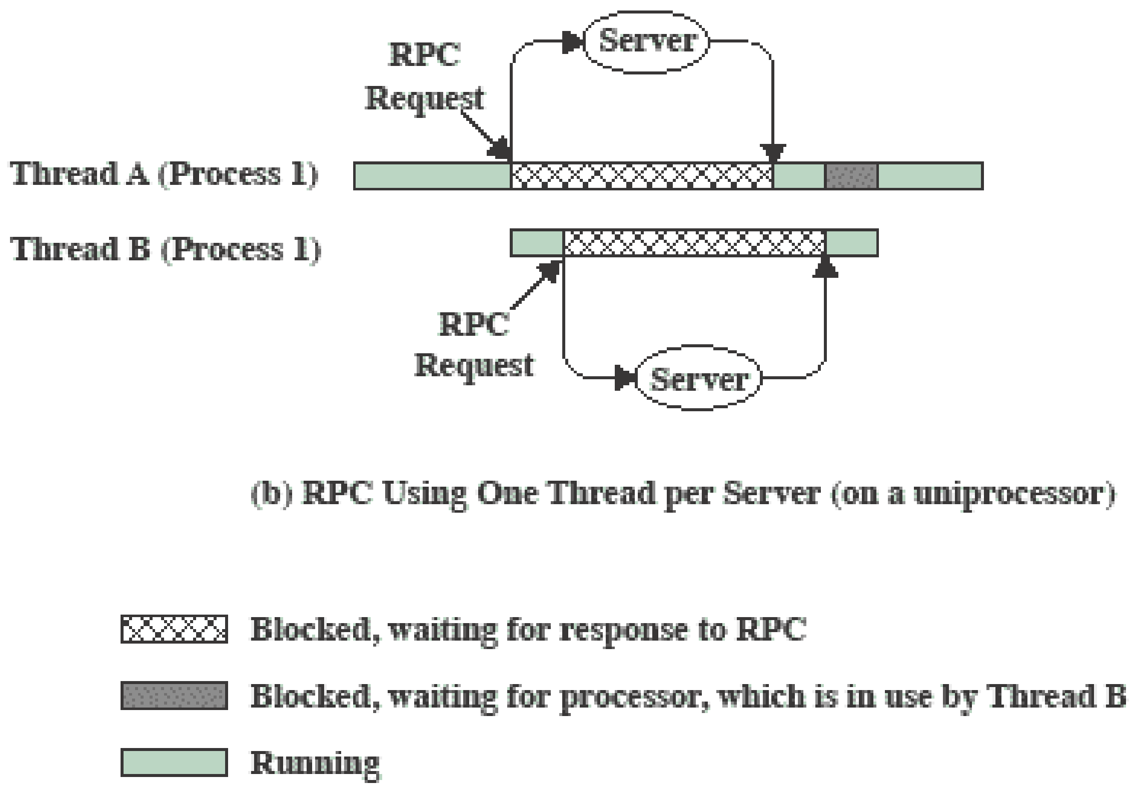
- Consider:
  - A program that performs two remote procedure calls (RPCs)
  - to two different hosts
  - to obtain a combined result.

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)

Multithreading on a Uniprocessor

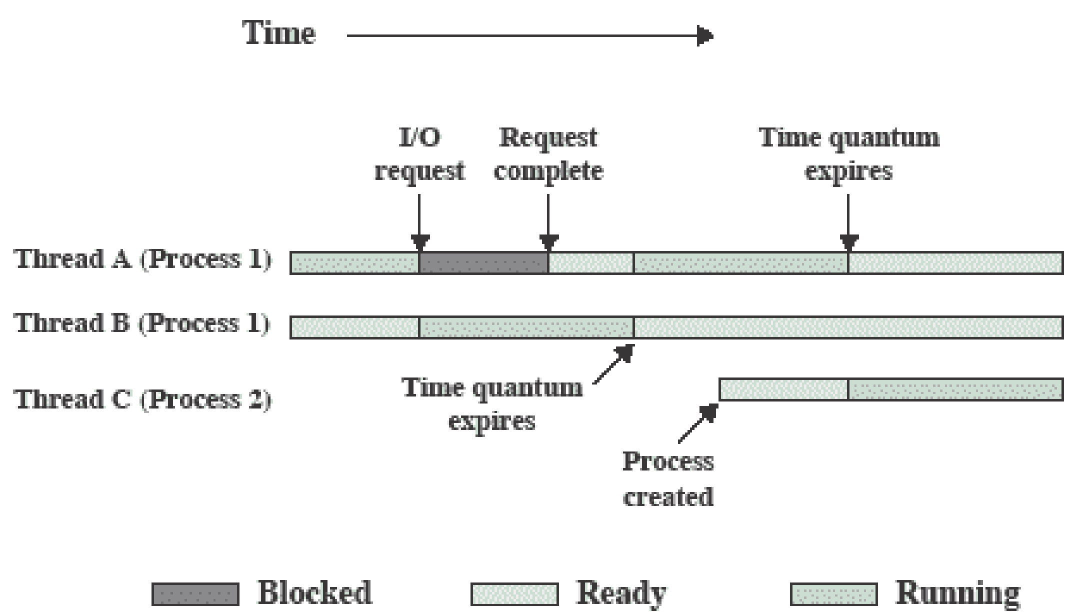


Figure 4.4 Multithreading Example on a Uniprocessor

Adobe PageMaker

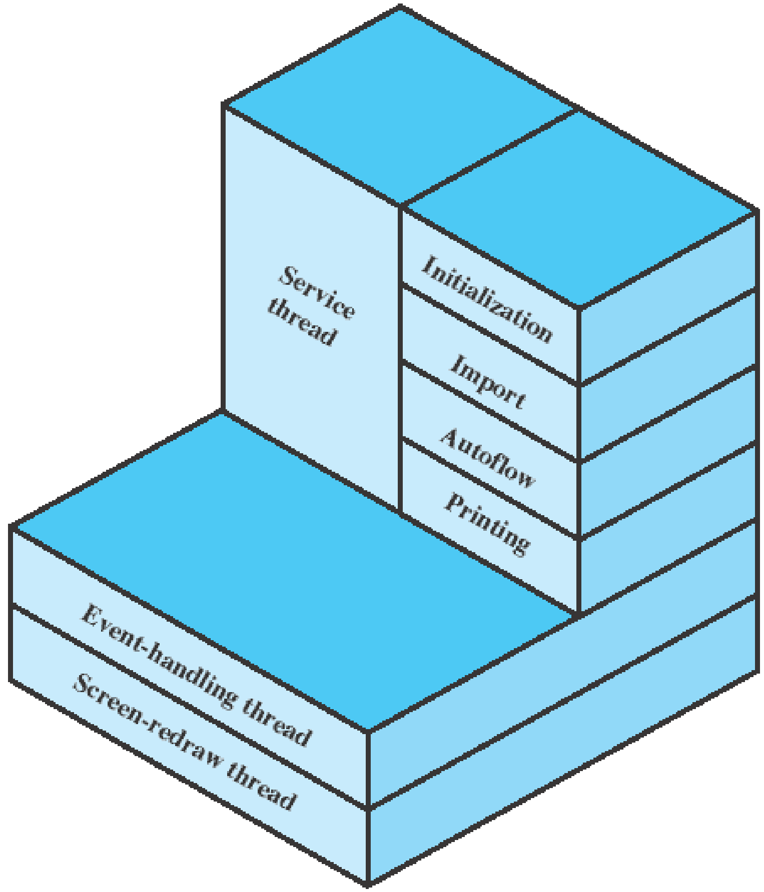


Figure 4.5 Thread Structure for Adobe PageMaker

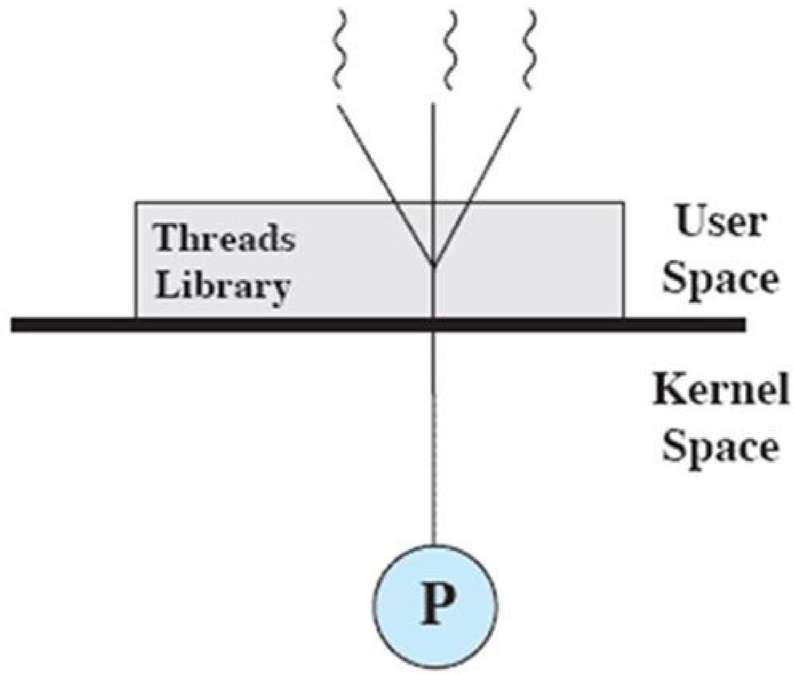
Categories of Thread Implementation

- 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



(a) Pure user-level

Relationships between ULT Thread and Process States

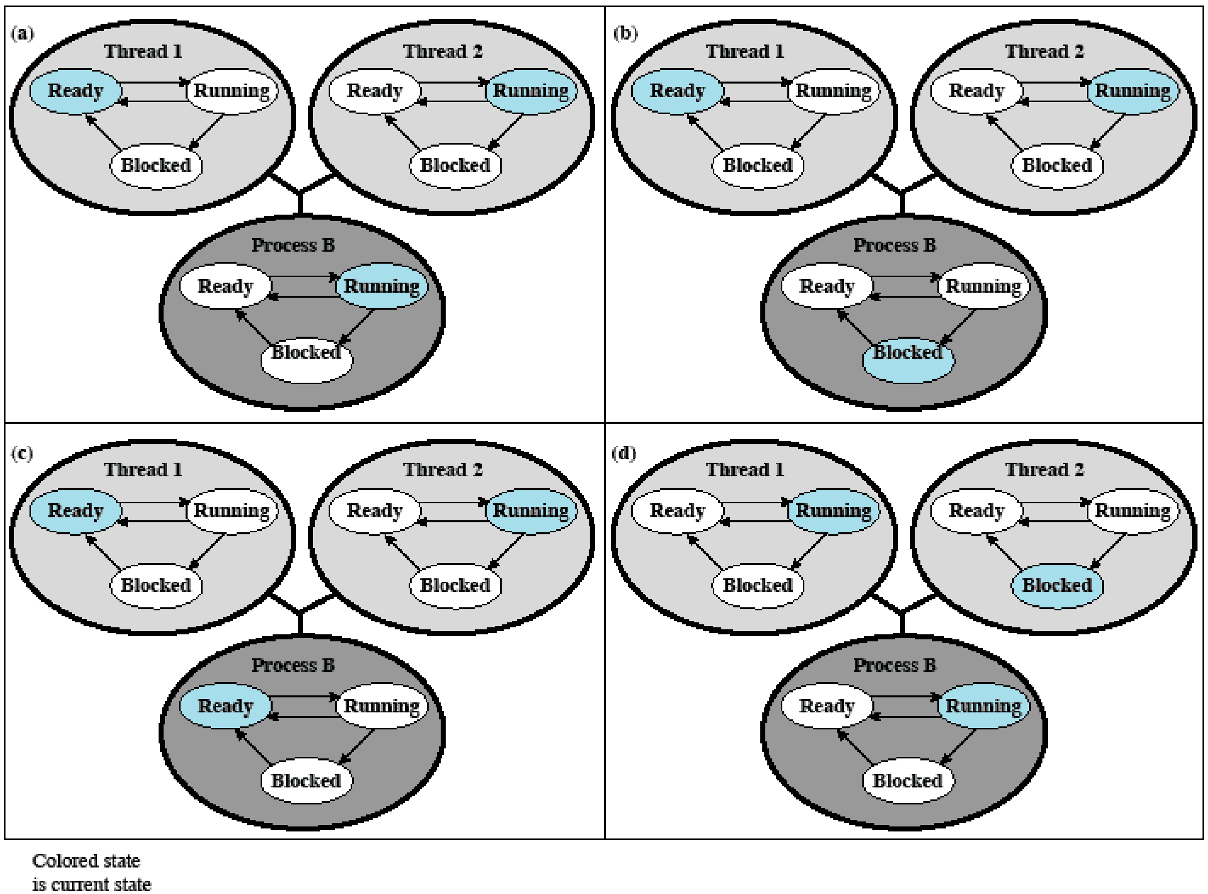
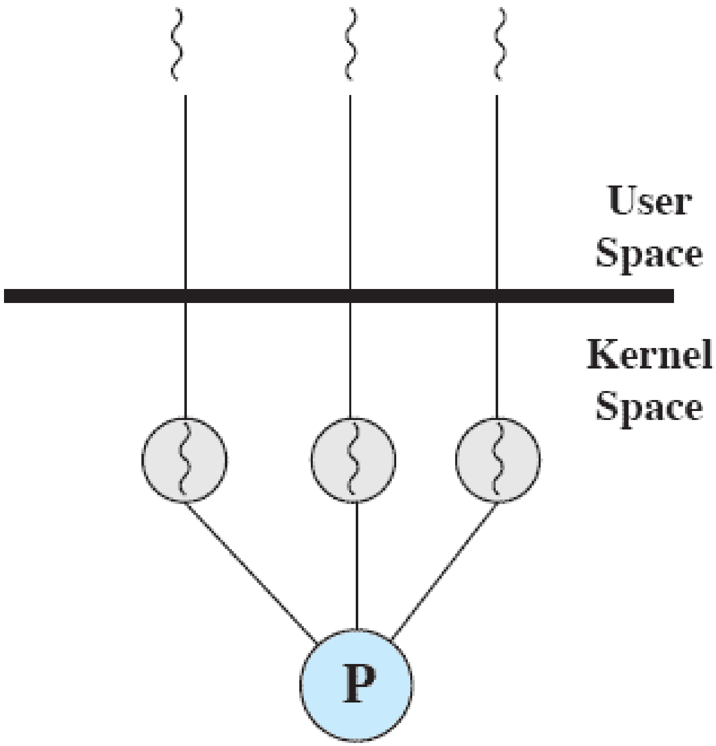


Figure 4.7 Examples of the Relationships Between User-Level Thread States and Process States

Kernel-Level Threads



(b) Pure kernel-level

- 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.

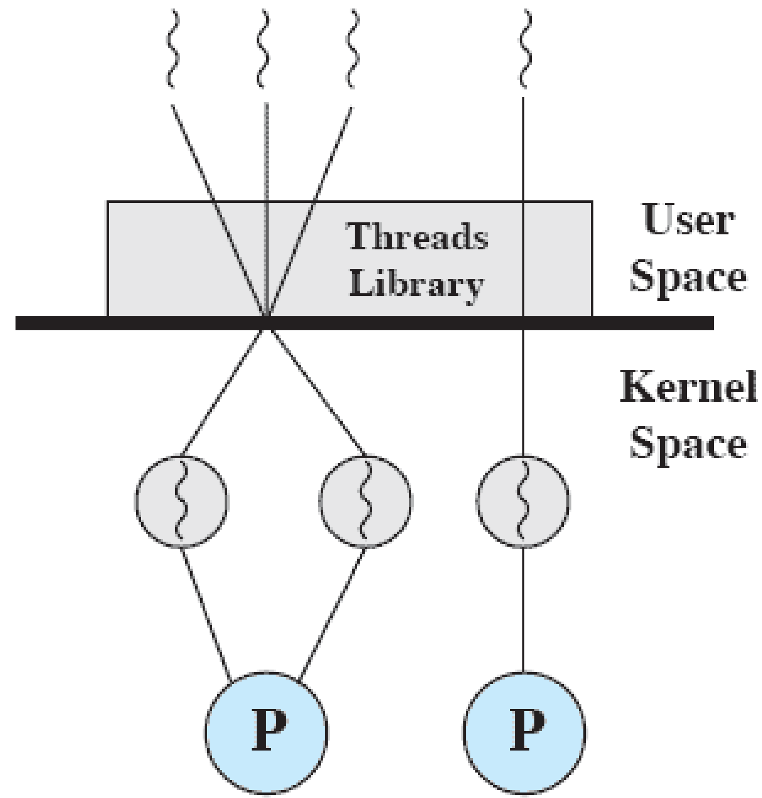
Disadvantage of KLT

- 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



(c) Combined

Relationship Between Thread 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

Roadmap

- Threads: Resource ownership and execution
- Symmetric multiprocessing (SMP).
- Microkernel
- Case Studies of threads and SMP:
  - Windows
  - Solaris
  - Linux

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

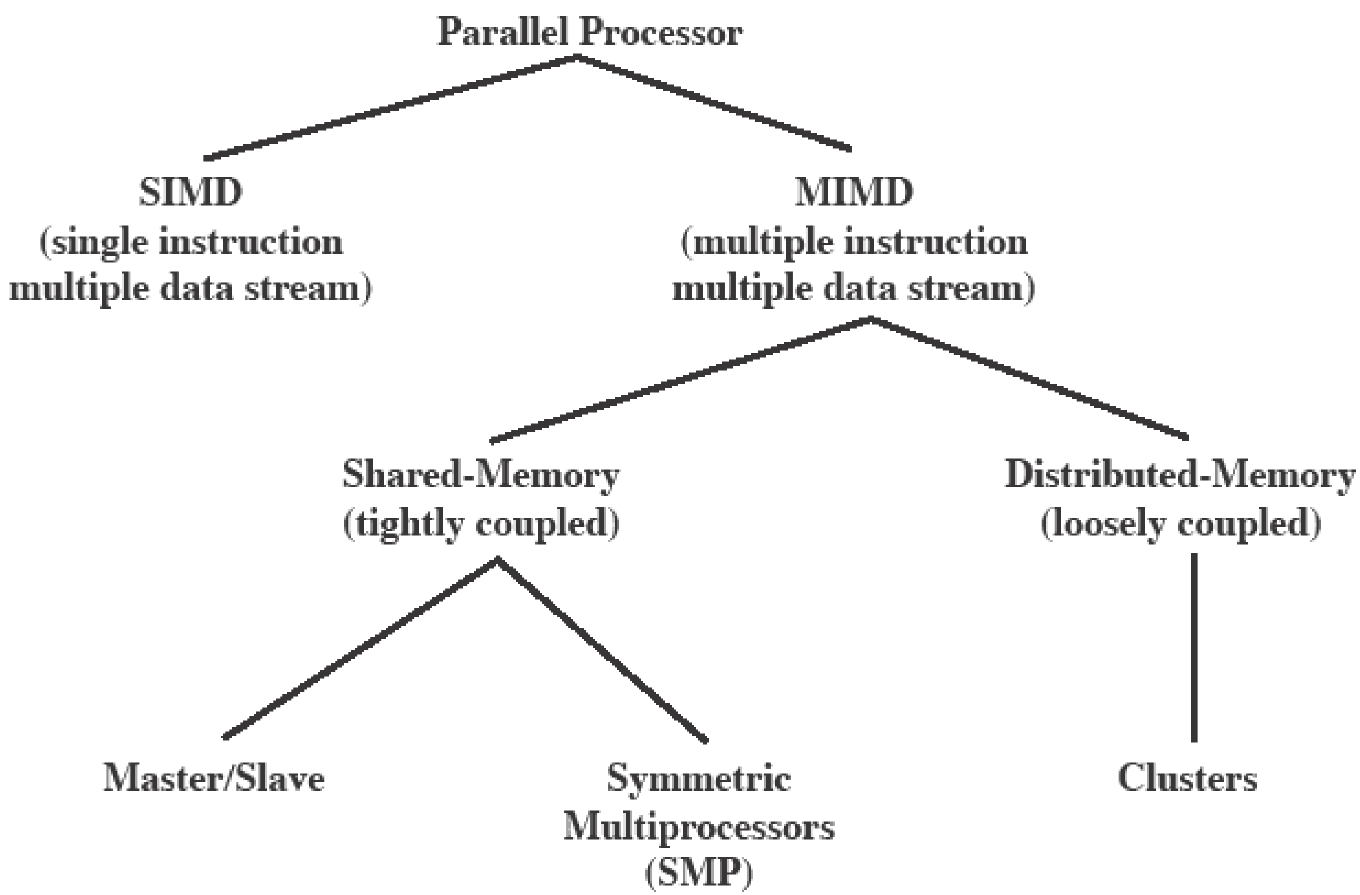


Figure 4.8 Parallel Processor Architectures

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

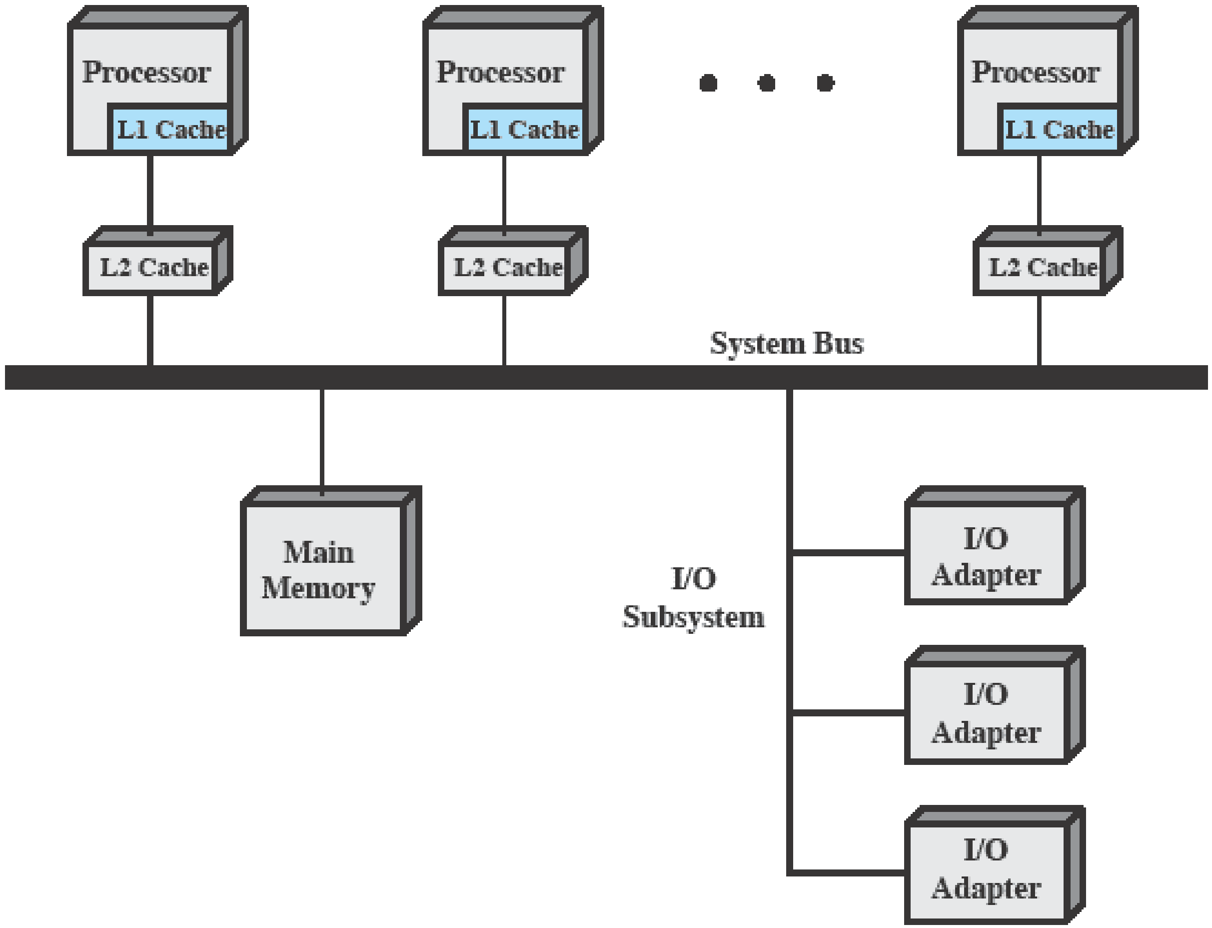


Figure 4.9 Symmetric Multiprocessor Organization

Multiprocessor OS Design Considerations

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

Roadmap

- Threads: Resource ownership and execution
- Symmetric multiprocessing (SMP).

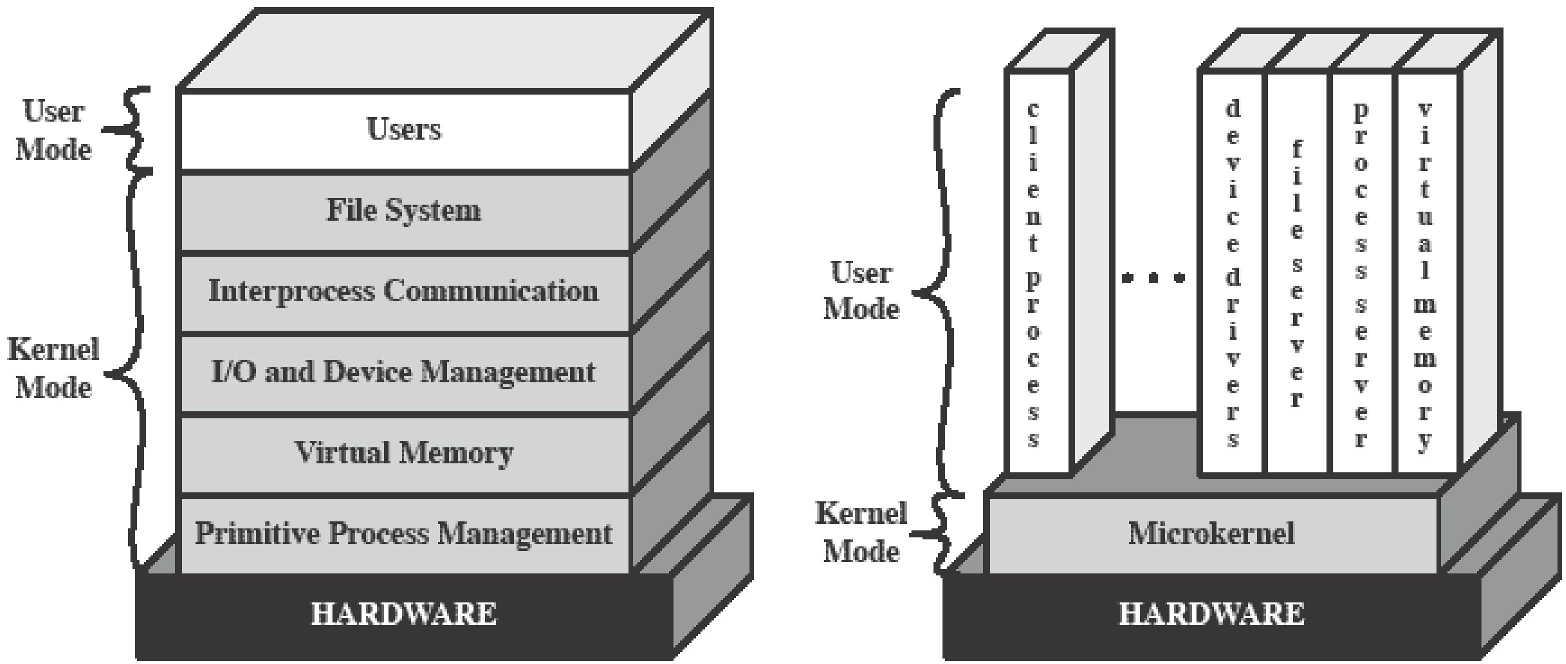
Microkernel

- Case Studies of threads and SMP:
  - Windows
  - Solaris
  - Linux

Microkernel

- A microkernel is a small OS core that provides the foundation for modular extensions.
- Big question is how small must a kernel be to qualify as a microkernel
  - *Must* drivers be in user space?
- In theory, this approach provides a high degree of flexibility and modularity.

Kernel Architecture



(a) Layered kernel

(b) Microkernel

Figure 4.10 Kernel Architecture

Microkernel Design: Memory Management

- Low-level memory management - Mapping each virtual page to a physical page frame
  - Most memory management tasks occur in user space

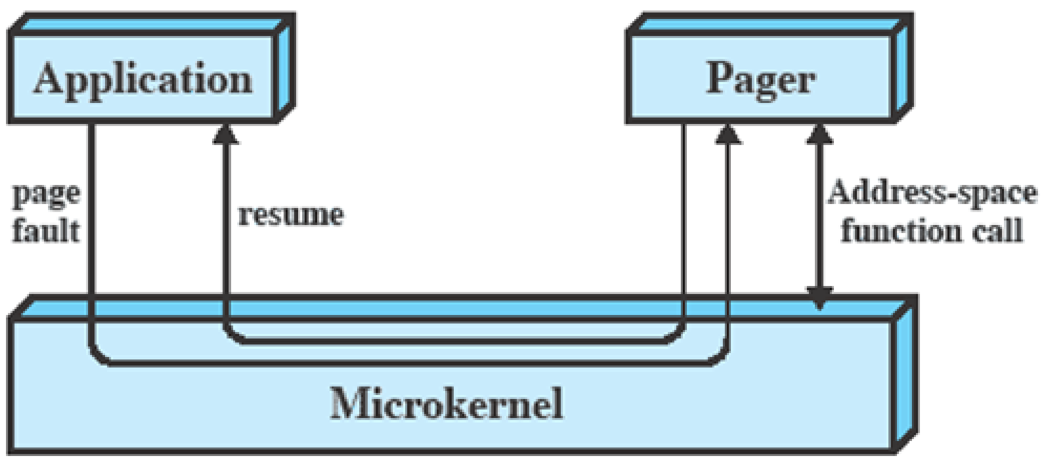


Figure 4.11 Page Fault Processing

Microkernel Design: Interprocess Communication

- Communication between processes or threads in a microkernel OS is via messages.
- A message includes:
  - A header that identifies the sending and receiving process and
  - A body that contains direct data, a pointer to a block of data, or some control information about the process.

Microkernal Design: I/O and interrupt management

- Within a microkernel it is possible to handle hardware interrupts as messages and to include I/O ports in address spaces.
  - a particular user-level process is assigned to the interrupt and the kernel maintains the mapping.

Benefits of a Microkernel Organization

- Uniform interfaces on requests made by a process.
- Extensibility
- Flexibility
- Portability
- Reliability
- Distributed System Support
- Object Oriented Operating Systems