Operating Systems

Chapter 4 Threads, SMP, and Microkernels

Agenda

- 4.1 Processes and Threads
- 4.2 Symmetric Multiprocessing
- 4.3 Microkernel
- 4.4 Summary

4.1 Processes and Threads

- <u>4.1.1 Introduction</u>
- 4.1.2 Multithreading
- 4.1.3 Thread Functionality
- 4.1.4 User-Level and Kernel-Level Thread
- 4.1.5 Other Arrangements

Introduction

- 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
- These two characteristics are treated independently by the operating system

Introduction

- 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 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
- 4.1.4 User-Level and Kernel-Level Thread
- 4.1.5 Other Arrangements

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）

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, Solaris, Linux, Mach, and OS/2 support multiple threads

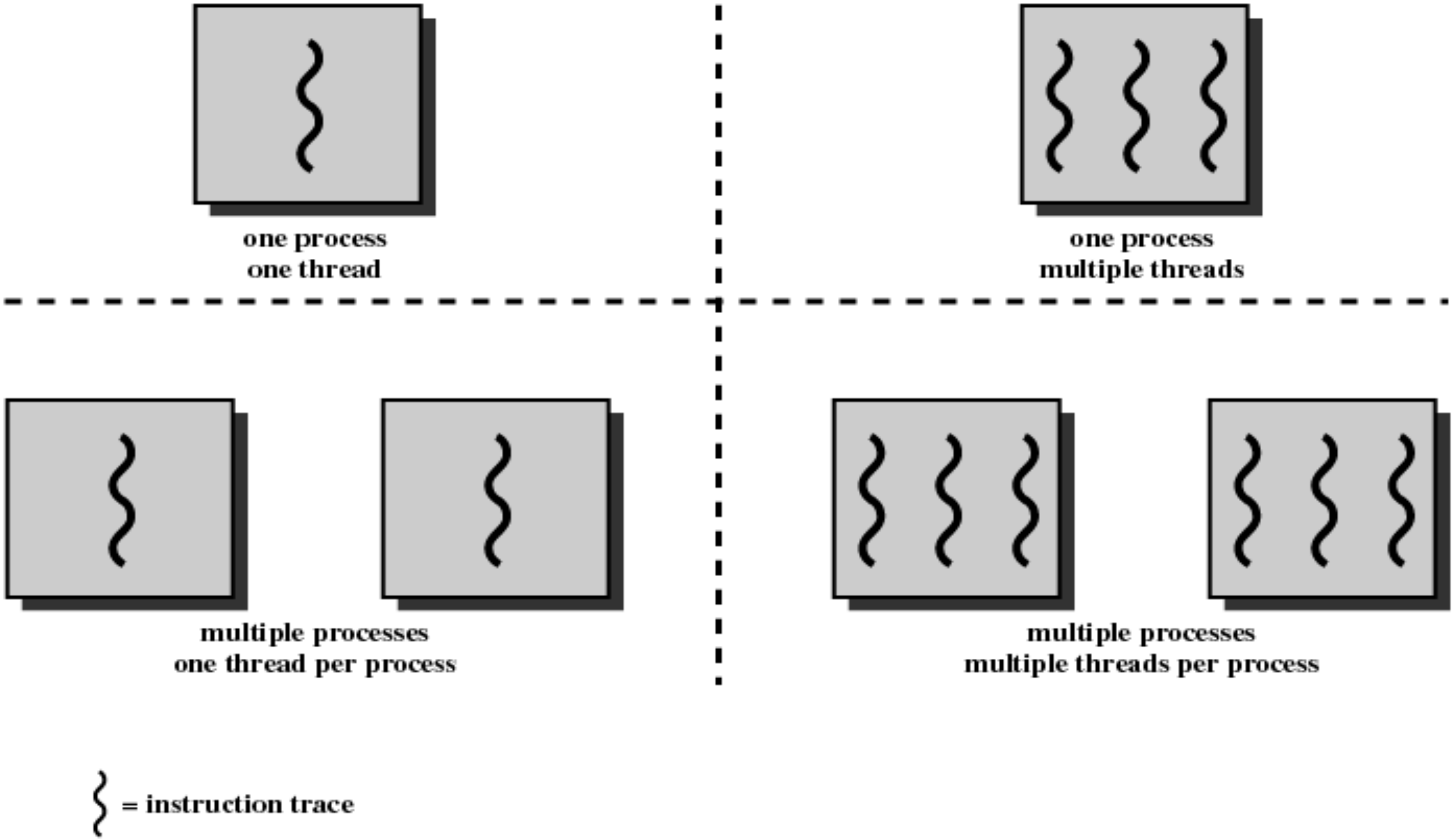


Figure 4.1 Threads and Processes [ANDE97]

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

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
  - Has an execution stack
  - Some per-thread static storage for local variables 局部变量存储空间
  - Access to the memory and resources of its process
    - all threads of a process share this

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)

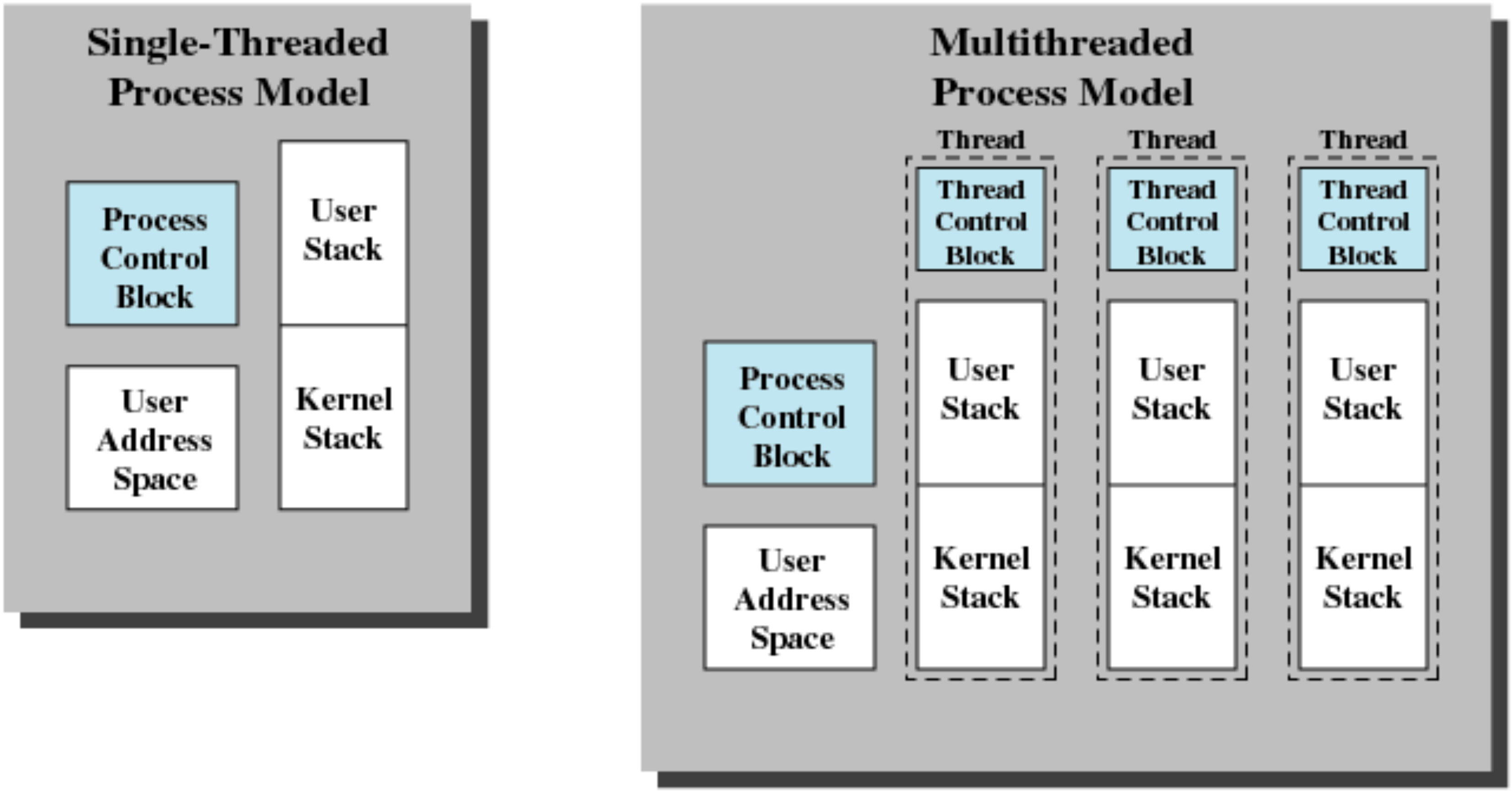


Figure 4.2 Single Threaded and Multithreaded Process Models

Benefits(优点) of Threads

1. Takes less time to create a new thread than a process (创建快)
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)
4. Since threads within the same process share memory and files, they can communicate with each other without invoking the kernel (通信快)

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
- <u>4.1.3 Thread Functionality (线程功能特性)</u>
- 4.1.4 User-Level and Kernel-Level Thread
- 4.1.5 Other Arrangements

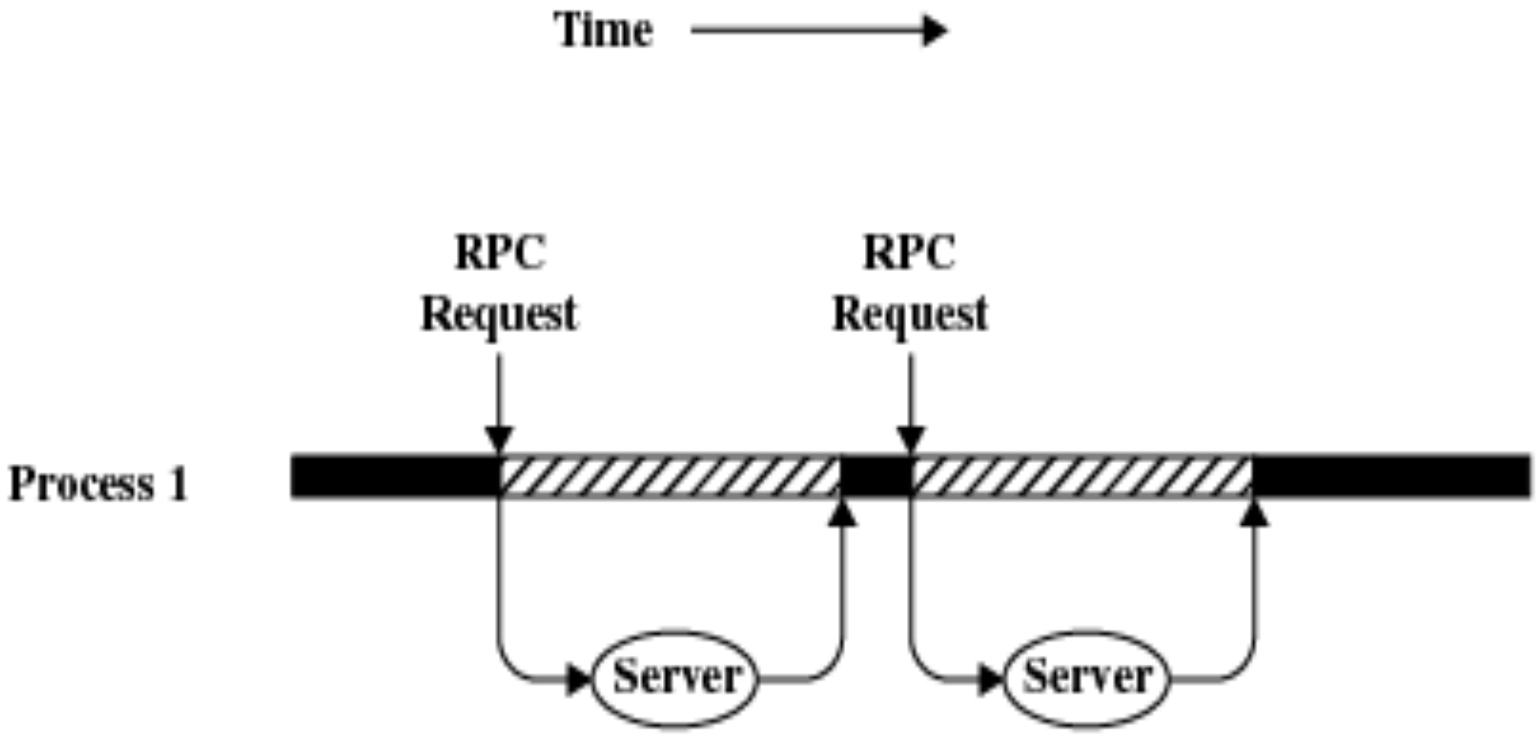
Thread States

- 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

Thread States

- Suspending a process involves suspending all threads of the process
- Blocking a thread involves blocking the process ， right?

Remote Procedure Call Using Single Thread



(a) RPC Using Single Thread

Remote Procedure Call Using Threads

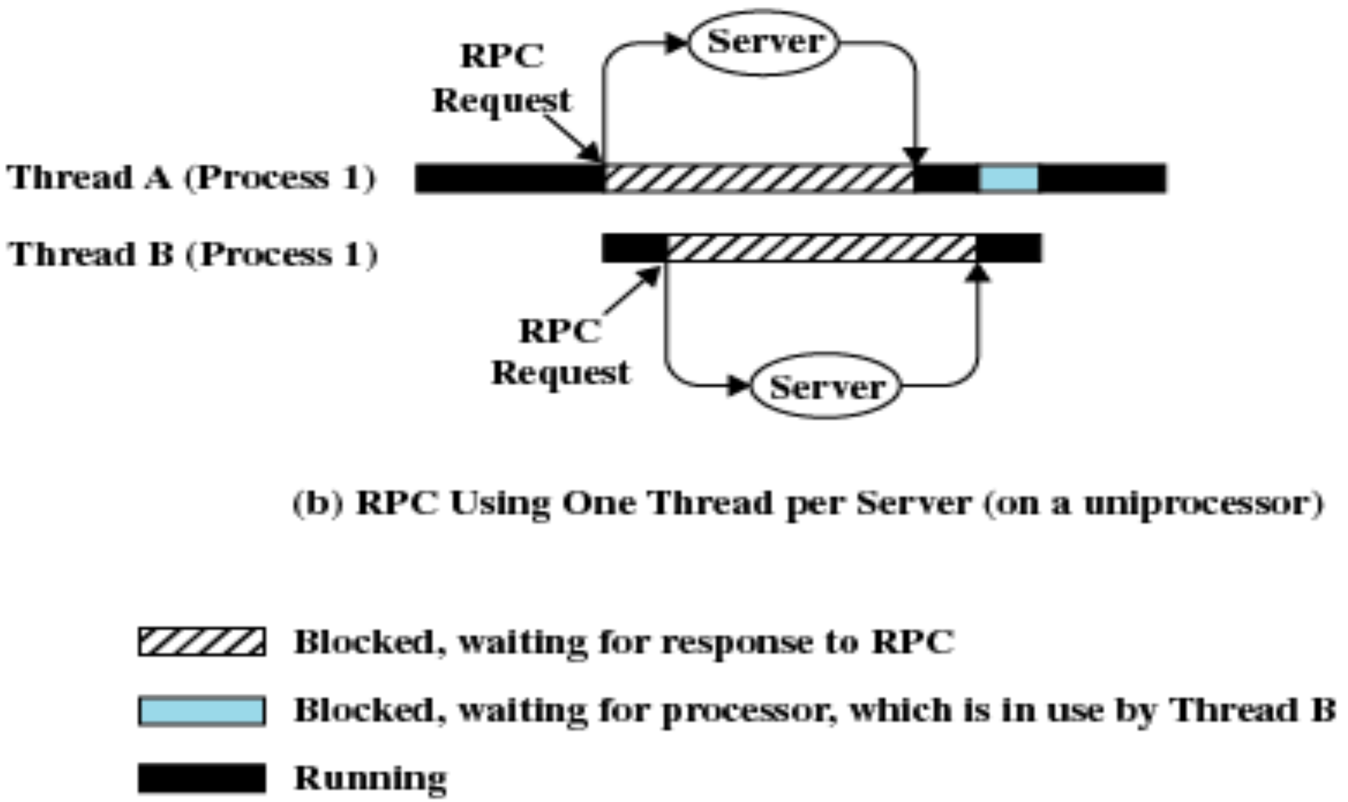


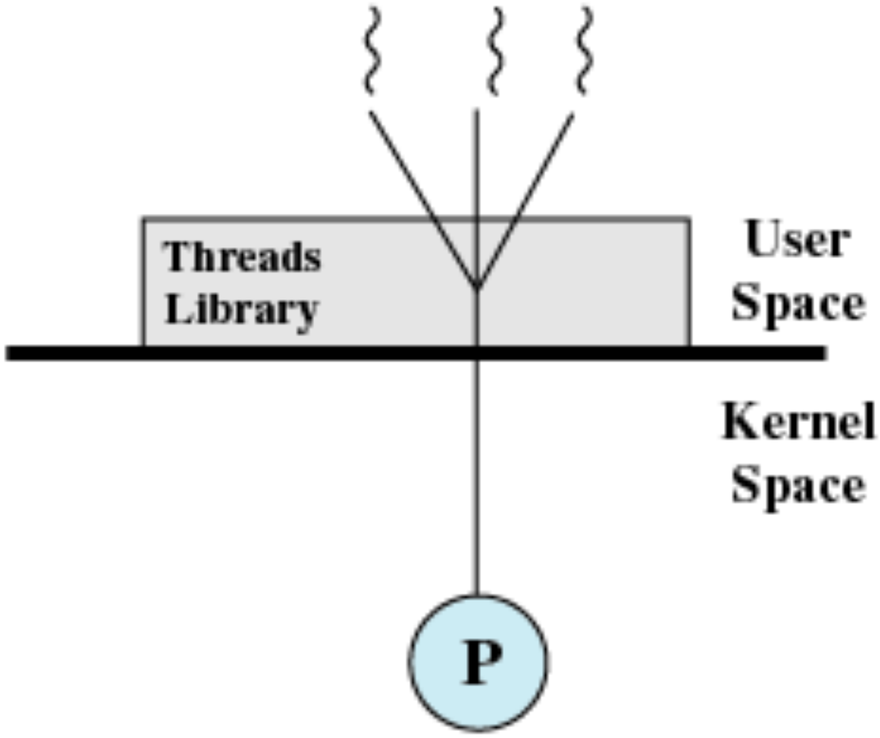
Figure 4.3 Remote Procedure Call (RPC) Using Threads

4.1 Processes and Threads

- 4.1.1 Introduction
- 4.1.2 Multithreading
- 4.1.3 Thread Functionality
- <u>4.1.4 User-Level and Kernel-Level Thread</u>
- 4.1.5 Other Arrangements

User-Level Threads (ULT，用户级线程)

- Multithread implemented by a threads library
- 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



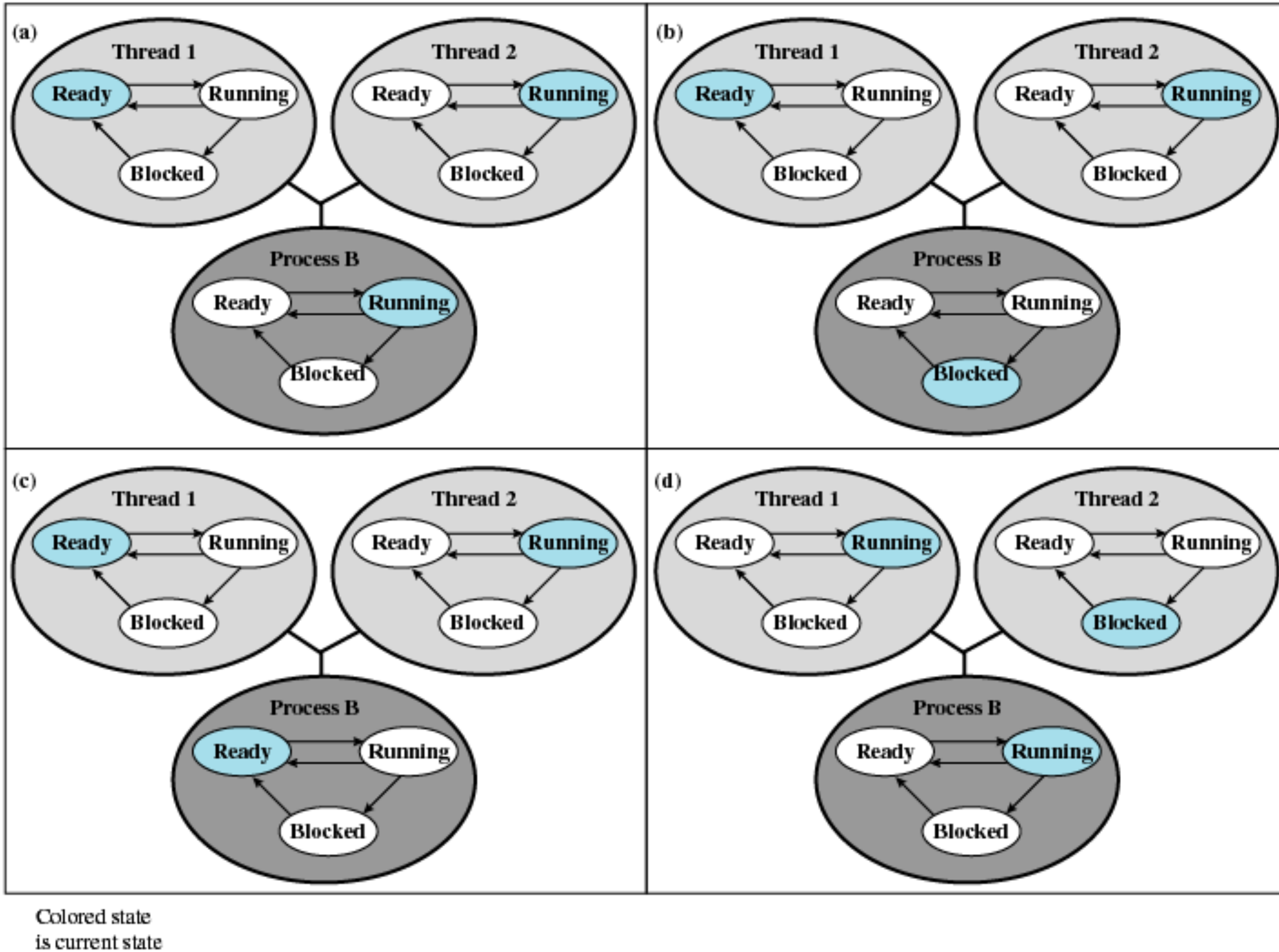
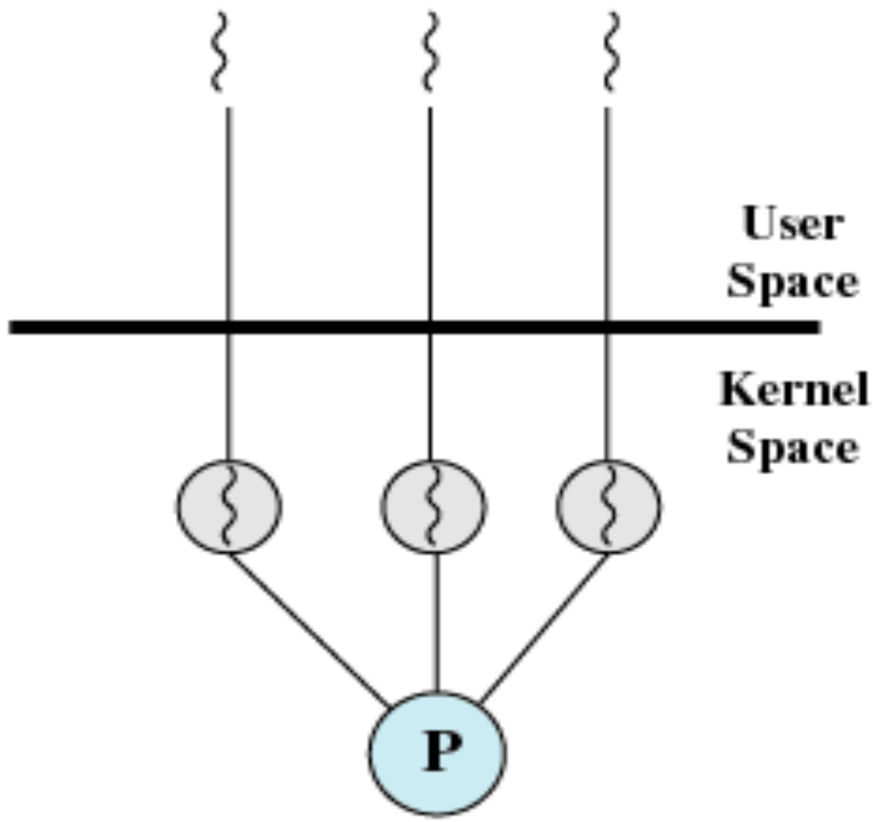


Figure 4.7 Examples of the Relationships Between User-Level Thread States and Process States

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

Advantages of ULT to KLT

- Less overhead更少的代价 of thread switches(mode switches do not required)
- Scheduling can be application specific
- ULTs can run on any operating system

Disadvantages of ULT to KLT

- One thread is blocked, all other threads of the process are blocked
- A multithreaded application cannot take advantage of multiprocessing
- Ways to work around these drawbacks:
  - Multiple processes 多进程
  - Jacketing套管 非阻塞调用

Advantages of KLT to ULT

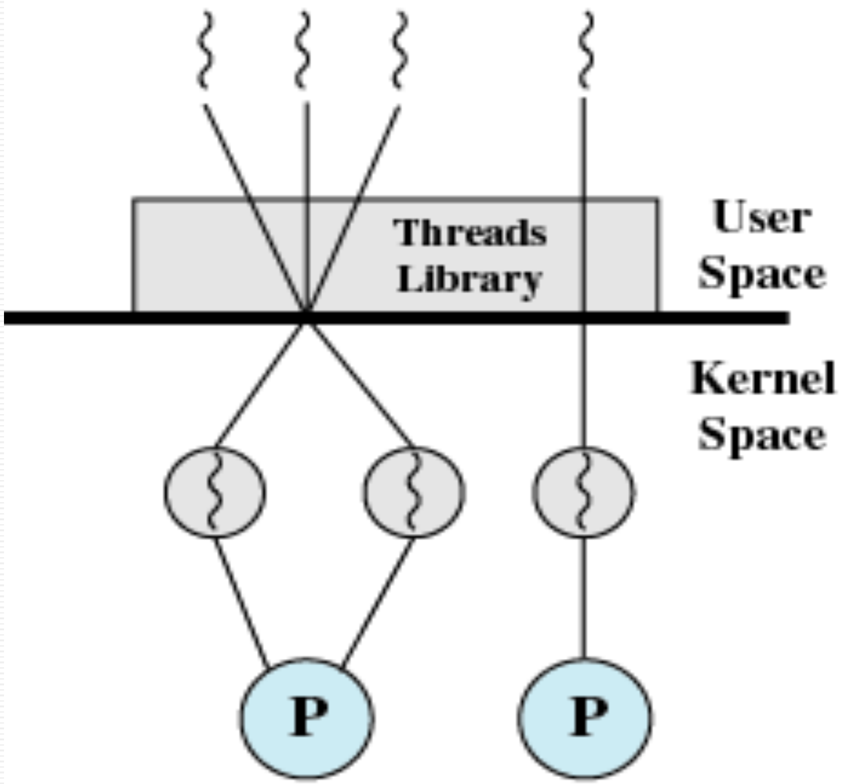
- Overcomes the two principal drawbacks of the ULT
  - Multiple threads in one process can simultaneously run on multiple processors
  - One threads blocked cannot make the other threads within the same process blocked
  - Kernel routines例程 themselves can be multithreaded

Disadvantages of KLT to ULT

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

Combined Approaches

- Example is Solaris
- Thread creation done in the user space
- The multiple ULTs from a single application are mapped onto some (smaller or equal) number of KLTs
- Bulk of scheduling and synchronization大量的调度和同步of threads within application



(c) Combined

4.1 Processes and Threads

- 4.1.1 Introduction
- 4.1.2 Multithreading
- 4.1.3 Thread Functionality
- 4.1.4 User-Level and Kernel-Level Thread
- 4.1.5 Other Arrangements方案

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

Agenda

- 4.1 Processes and Threads
- <u>4.2 Symmetric均衡 Multiprocessing</u>
- 4.3 Microkernel
- 4.4 Summary

4.2 Symmetric Multiprocessing

- 4.2.1 SMP Architecture
- 4.2.2 SMP Organization
- 4.2.3 Multiprocessor Operating System Design Considerations

Categories of Computer Systems

- Single Instruction Single Data (SISD) stream(单指令单数据流)
  - Single processor executes a single instruction stream to operate on data stored in a single memory
- Single Instruction Multiple Data (SIMD) stream(单指令多数据流)
  - A single instruction controls the simultaneous execution of a number of processing elements on a lockstep basis. Each instruction is executed on a different set of data by the different processors
  - E.g. vector and array processors

Categories of Computer Systems

- Multiple Instruction Single Data (MISD) stream(多指令单数据流)
  - A sequence of data is transmitted to a set of processors, each of which executes a different instruction sequence. Never implemented
- Multiple Instruction Multiple Data (MIMD) stream (多指令多数据流)
  - A set of processors simultaneously execute different instruction sequences on different data sets

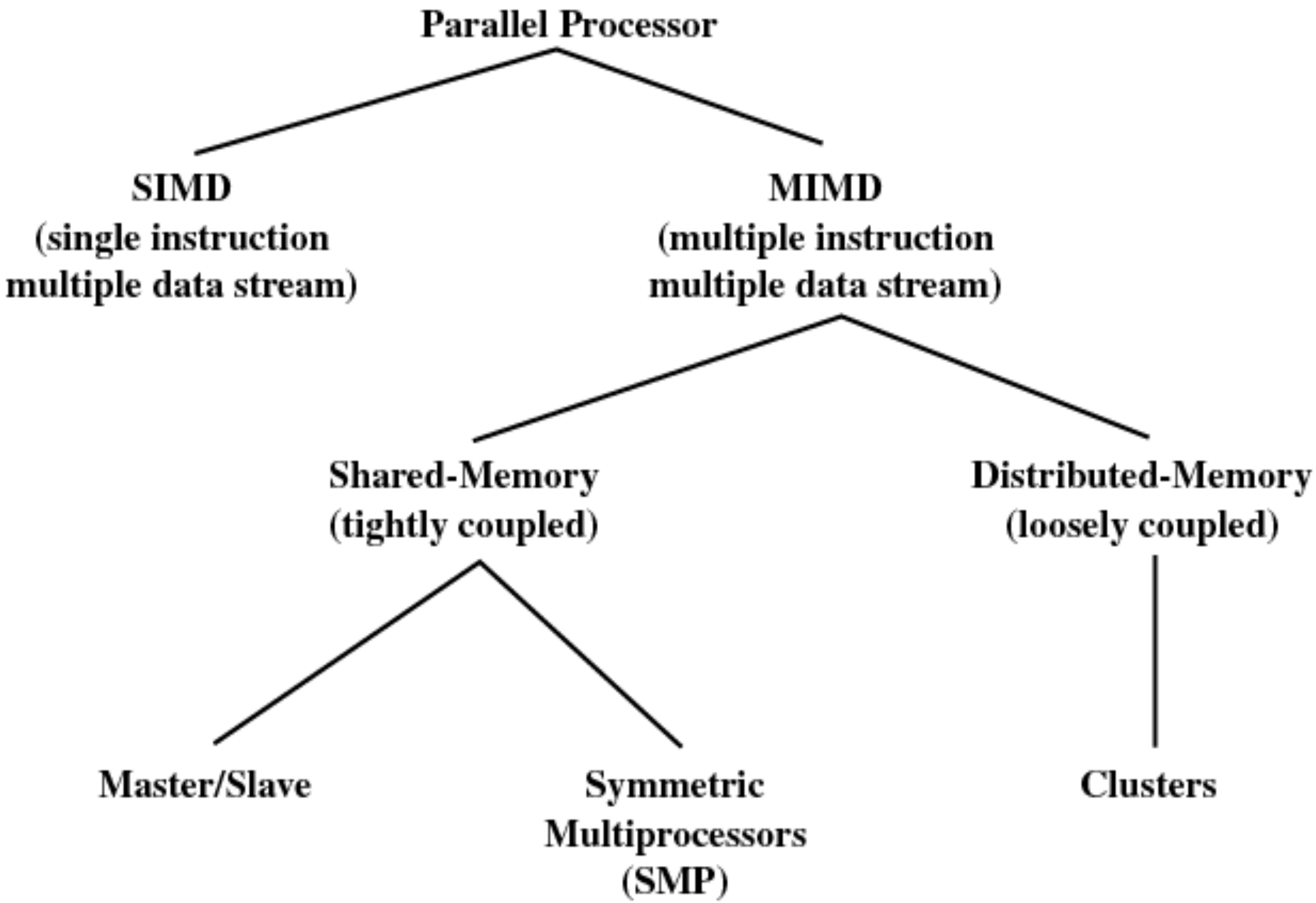


Figure 4.8 Parallel Processor Architectures

Symmetric Multiprocessing

- Kernel can execute on any processor
- Typically each processor does self-scheduling form the pool of available process or threads自己调度形成可用的进程或线程池

4.2 Symmetric Multiprocessing

- 4.2.1 SMP Architecture
- <u>4.2.2 SMP Organization</u>
- 4.2.3 Multiprocessor Operating System Design Considerations

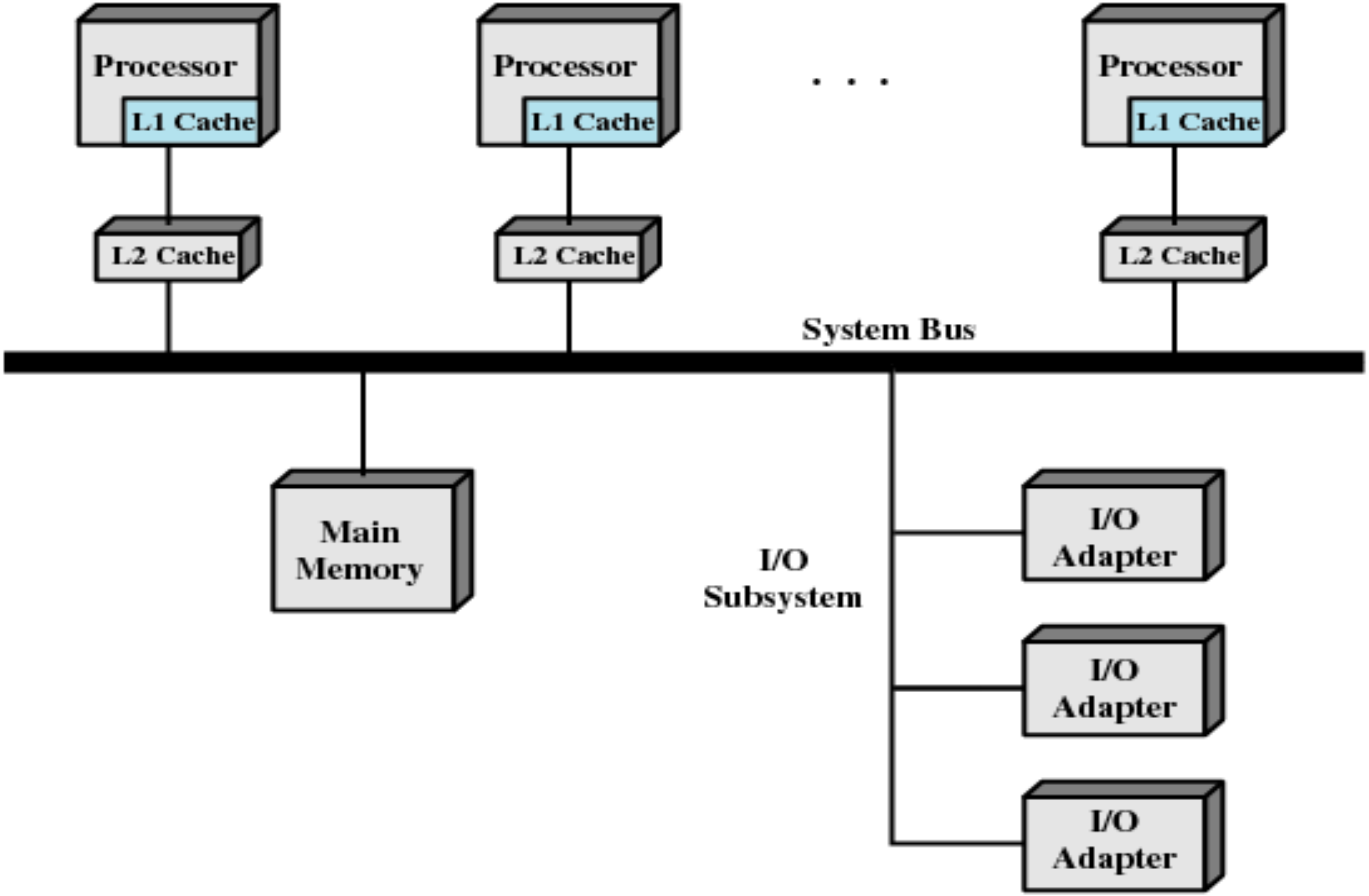


Figure 4.9 Symmetric Multiprocessor Organization

4.2 Symmetric Multiprocessing

- 4.2.1 SMP Architecture
- 4.2.2 SMP Organization
- 4.2.3 Multiprocessor Operating System Design Considerations

Multiprocessor Operating System Design Considerations

- Simultaneous(同时) concurrent(并发) processes or threads（内核例程要求可重入reentrant，避免死锁)
- Scheduling
- Synchronization(同步)
- Memory management
- Reliability(可靠性) and fault tolerance(容错)

Agenda

- 4.1 Processes and Threads
- 4.2 Symmetric Multiprocessing
- <u>4.3 Microkernel</u>
- 4.4 Summary

4.3 Microkernel(微内核)

- 4.3.1 Microkernel Architecture
- 4.3.2 Benefits of a Microkernel Organization
- 4.3.3 Microkernel Performance
- 4.3.4 Microkernel Design

Microkernel

- Small operating system core
- Contains only essential core operating systems functions
- Many services traditionally included in the operating system are now external subsystems
  - Device drivers
  - File systems
  - Virtual memory manager
  - Windowing system
  - Security services
- 相对的内核称为单体内核—monolithic kernel

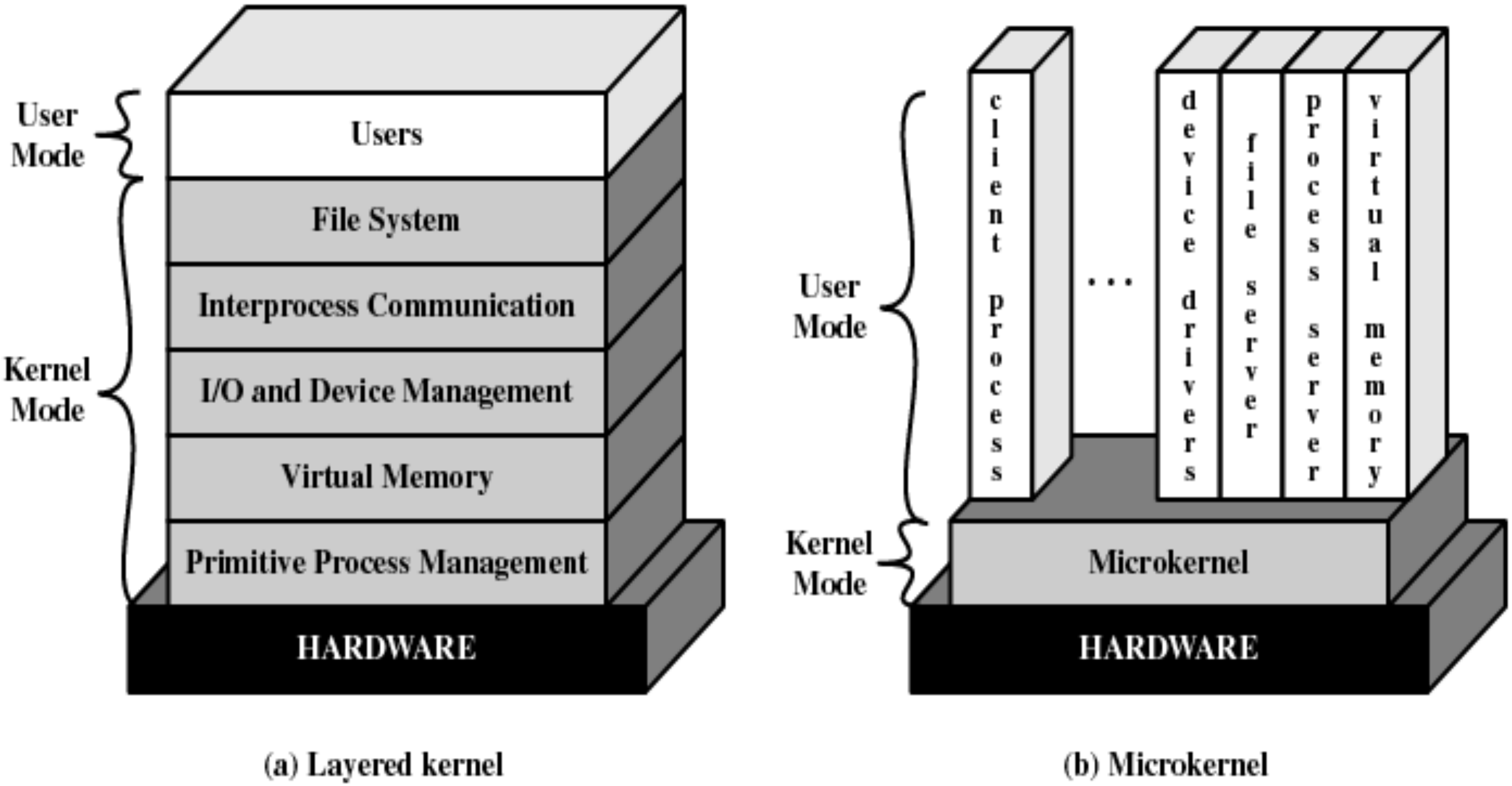


Figure 4.10 Kernel Architecture

4.3 Microkernel

- 4.3.1 Microkernel Architecture
- <u>4.3.2 Benefits of a Microkernel Organization</u>
- 4.3.3 Microkernel Performance
- 4.3.4 Microkernel Design

Benefits of a Microkernel Organization

1. Uniform interface(一致接口) on request made by a process
   – Don't distinguish between kernel-level and user-level services
   – All services are provided by means of message passing
2. Extensibility(可扩展性)
   – Allows the addition of new services
3. Flexibility(灵活性)
   – New features added
   – Existing features can be subtracted(删减)

Benefits of a Microkernel Organization

4. Portability(可移植性)
   - Changes needed to port the system to a new processor is changed in the microkernel - not in the other services
5. Reliability(可靠性)
   - Modular design
   - Small microkernel can be rigorously tested

Benefits of a Microkernel Organization

6. Distributed system support
   - Message are sent without knowing what the target machine is （消息的发送不用关心目标是本机还是异机）
7. Object-oriented operating system
   - Components are objects with clearly defined interfaces that can be interconnected to form software (定义接口明晰的组件，以搭积木方式通过组件互联构造软件)

4.3 Microkernel

- 4.3.1 Microkernel Architecture
- 4.3.2 Benefits of a Microkernel Organization
- <u>4.3.3 Microkernel Performance</u>
- 4.3.4 Microkernel Design

4.3.3 Microkernel Performance

- A potential disadvantage of microkernels that is often cited is that of performance (消息传递机制是微内核工作的主要机制，消息传递必须通过内核，因此性能较差)
- One response to this problem was to enlarge the microkernel by reintegrating critical servers and drivers back into the OS (把关键服务重新纳入内核，减少消息传递开销)

4.3 Microkernel

- 4.3.1 Microkernel Architecture
- 4.3.2 Benefits of a Microkernel Organization
- 4.3.3 Microkernel Performance
- <u>4.3.4 Microkernel Design</u>

Microkernel Design

The microkernel must include those functions that depend directly on the hardware. Those functions fall into the following general categories:

- Low level memory management
- Interprocess communication (IPC)
- I/O and interrupt management

Microkernel Design

1. Low-level memory management
   - Mapping each virtual page to a physical page frame

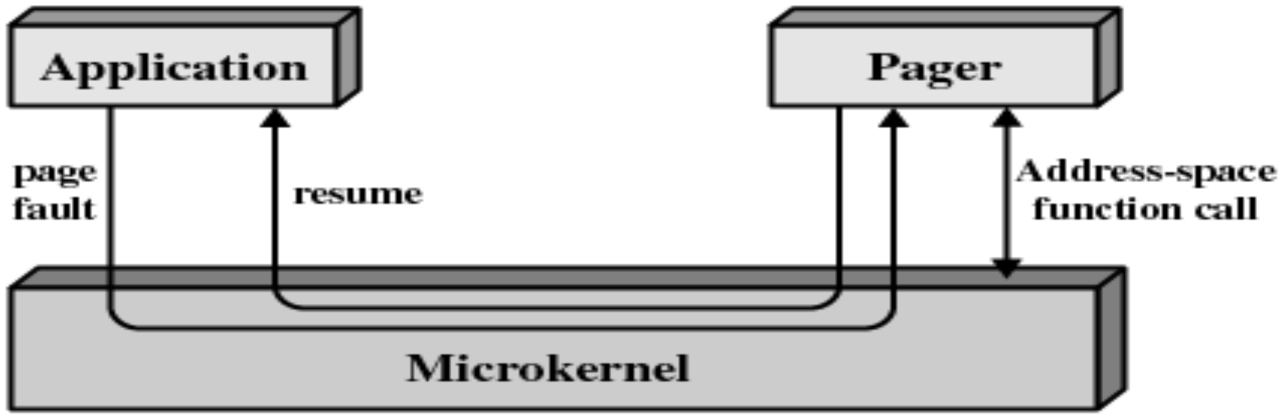


Figure 4.11 Page Fault Processing

Microkernel Design

2. Interprocess communication
   - Message
   - Port

Microkernel Design

3. I/O and interrupt management
   - The microkernel transforms interrupts into messages and dispatches messages to device-specific user-level interrupt handling.
   - Driver thread:

```
do
        waitFor (msg, sender);
        if (sender == my_hardware_interrupt)
        {
            read/write I/O ports;
            reset hardware interrupt
        }
        else …
while (true);
```

Agenda

- 4.1 Processes and Threads
- 4.2 Symmetric Multiprocessing
- 4.3 Microkernel
- <u>4.4 Summary</u>