操作系统原理

第三章：操作系统结构

洪明坚

重庆大学软件学院

February 19, 2016

目录

Outline

System structure

System structure

- A system as large and complex as a modern operating system must be engineered carefully if it's to function properly and be modified easily.

System structure

- A system as large and complex as a modern operating system must be engineered carefully if it's to function properly and be modified easily.
- A common approach is to partition the task into small components, rather than have one monolithic system.

System structure

- A system as large and complex as a modern operating system must be engineered carefully if it's to function properly and be modified easily.
- A common approach is to partition the task into small components, rather than have one monolithic system.
  - Each of these components should be a well-defined portion of the system, with carefully defined inputs, outputs and function.

System structure

- A system as large and complex as a modern operating system must be engineered carefully if it's to function properly and be modified easily.
- A common approach is to partition the task into small components, rather than have one monolithic system.
  - Each of these components should be a well-defined portion of the system, with carefully defined inputs, outputs and function.
- How does the system designer organize these components?

System structure

- A system as large and complex as a modern operating system must be engineered carefully if it's to function properly and be modified easily.
- A common approach is to partition the task into small components, rather than have one monolithic system.
  - Each of these components should be a well-defined portion of the system, with carefully defined inputs, outputs and function.
- How does the system designer organize these components?
  - Simple structure (or no structure)

System structure

- A system as large and complex as a modern operating system must be engineered carefully if it's to function properly and be modified easily.
- A common approach is to partition the task into small components, rather than have one monolithic system.
  - Each of these components should be a well-defined portion of the system, with carefully defined inputs, outputs and function.
- How does the system designer organize these components?
  - Simple structure (or no structure)
  - Layered structure

System structure

- A system as large and complex as a modern operating system must be engineered carefully if it's to function properly and be modified easily.
- A common approach is to partition the task into small components, rather than have one monolithic system.
  - Each of these components should be a well-defined portion of the system, with carefully defined inputs, outputs and function.
- How does the system designer organize these components?
  - Simple structure (or no structure)
  - Layered structure
  - Micro-kernel

System structure

- A system as large and complex as a modern operating system must be engineered carefully if it's to function properly and be modified easily.
- A common approach is to partition the task into small components, rather than have one monolithic system.
  - Each of these components should be a well-defined portion of the system, with carefully defined inputs, outputs and function.
- How does the system designer organize these components?
  - Simple structure (or no structure)
  - Layered structure
  - Micro-kernel
  - Virtual machine

Simple structure

Simple structure

- Many systems do not have a well-defined structure.

Simple structure

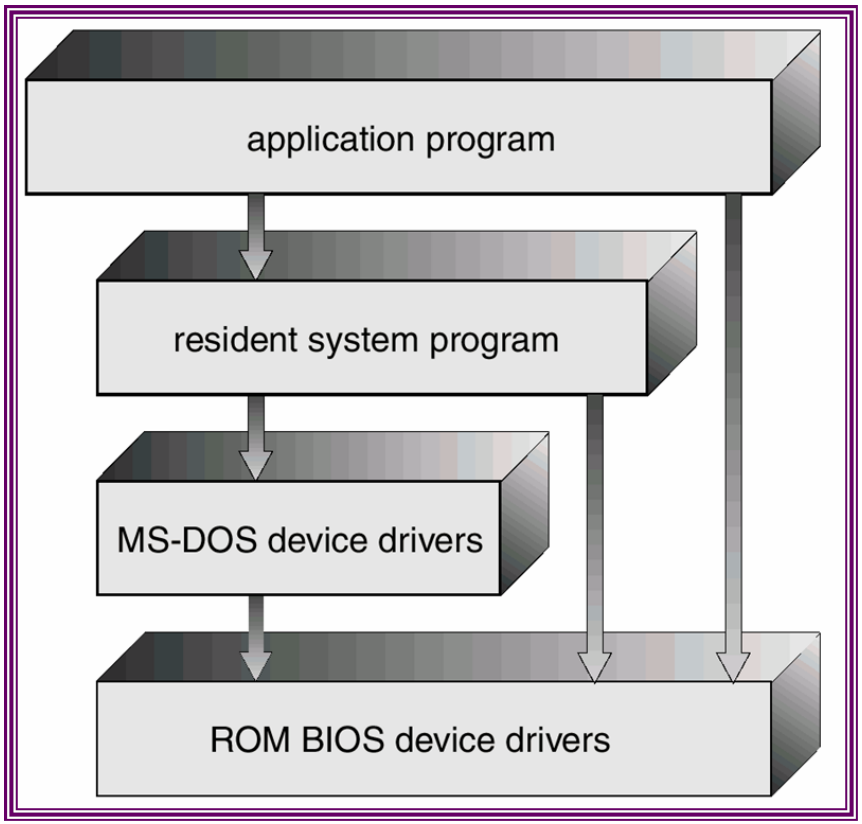
- Many systems do not have a well-defined structure.
  - They started as small, simple and limited system, and then evolved into a complex one.

Simple structure

- Many systems do not have a well-defined structure.
  - They started as small, simple and limited system, and then evolved into a complex one.
- Examples: MS-DOS and Unix

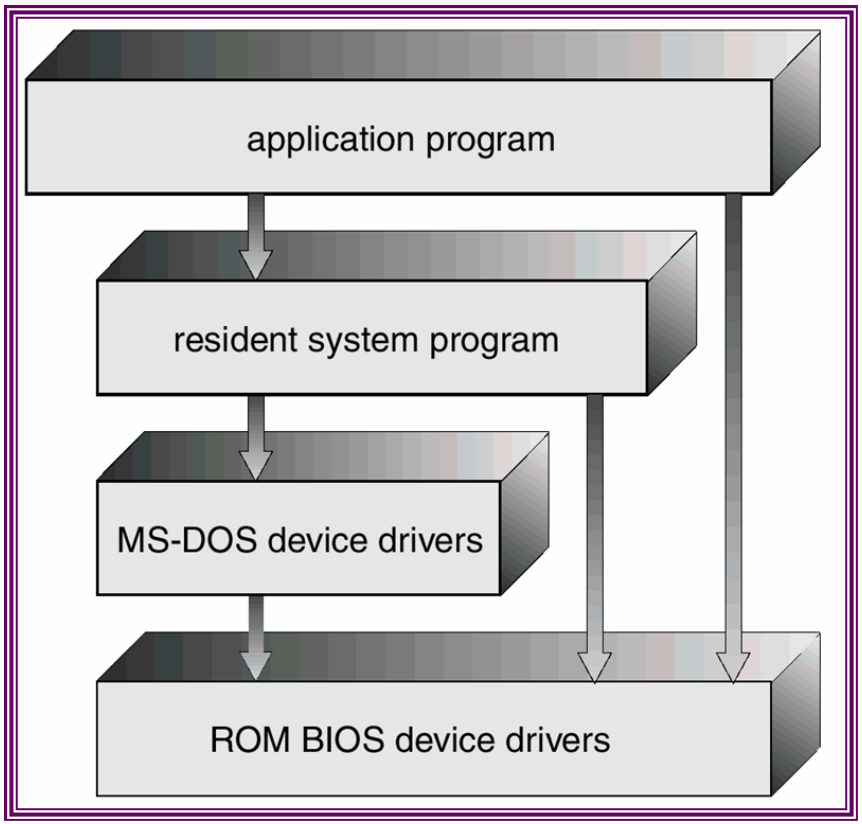
Simple structure

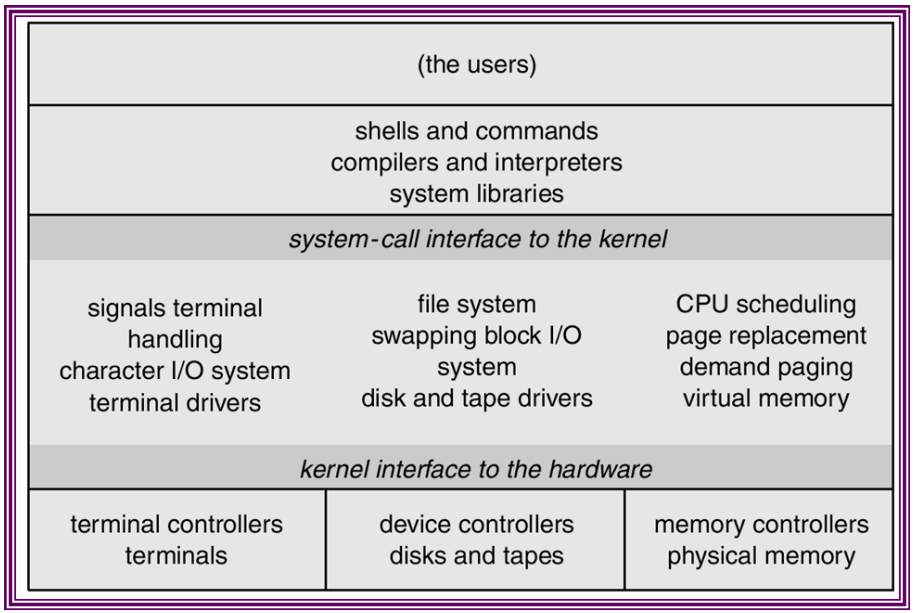
- Many systems do not have a well-defined structure.
  - They started as small, simple and limited system, and then evolved into a complex one.
- Examples: MS-DOS and Unix



Simple structure

- Many systems do not have a well-defined structure.
  - They started as small, simple and limited system, and then evolved into a complex one.
- Examples: MS-DOS and Unix





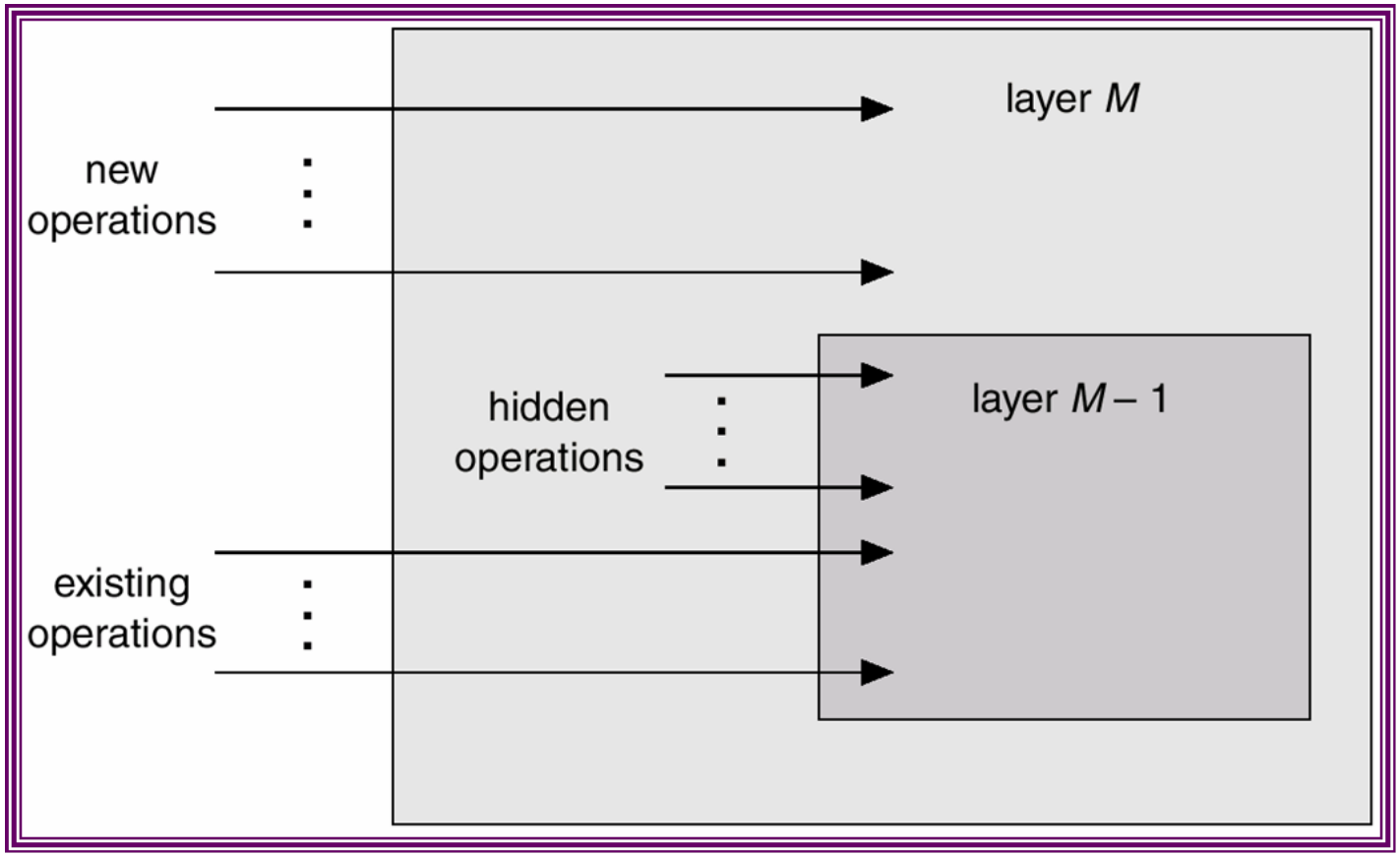
Layered structure (1/4)

Layered structure (1/4)

- The operating system is broken up into a number of layers, each built on top lower layers.

Layered structure (1/4)

- The operating system is broken up into a number of layers, each built on top lower layers.



Layered structure (2/4)

Layered structure (2/4)

- Example 1:

Layered structure (2/4)

- Example 1:
  - The **THE** operating system by Dijkstra.

Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

Layered structure (3/4)

Layered structure (3/4)

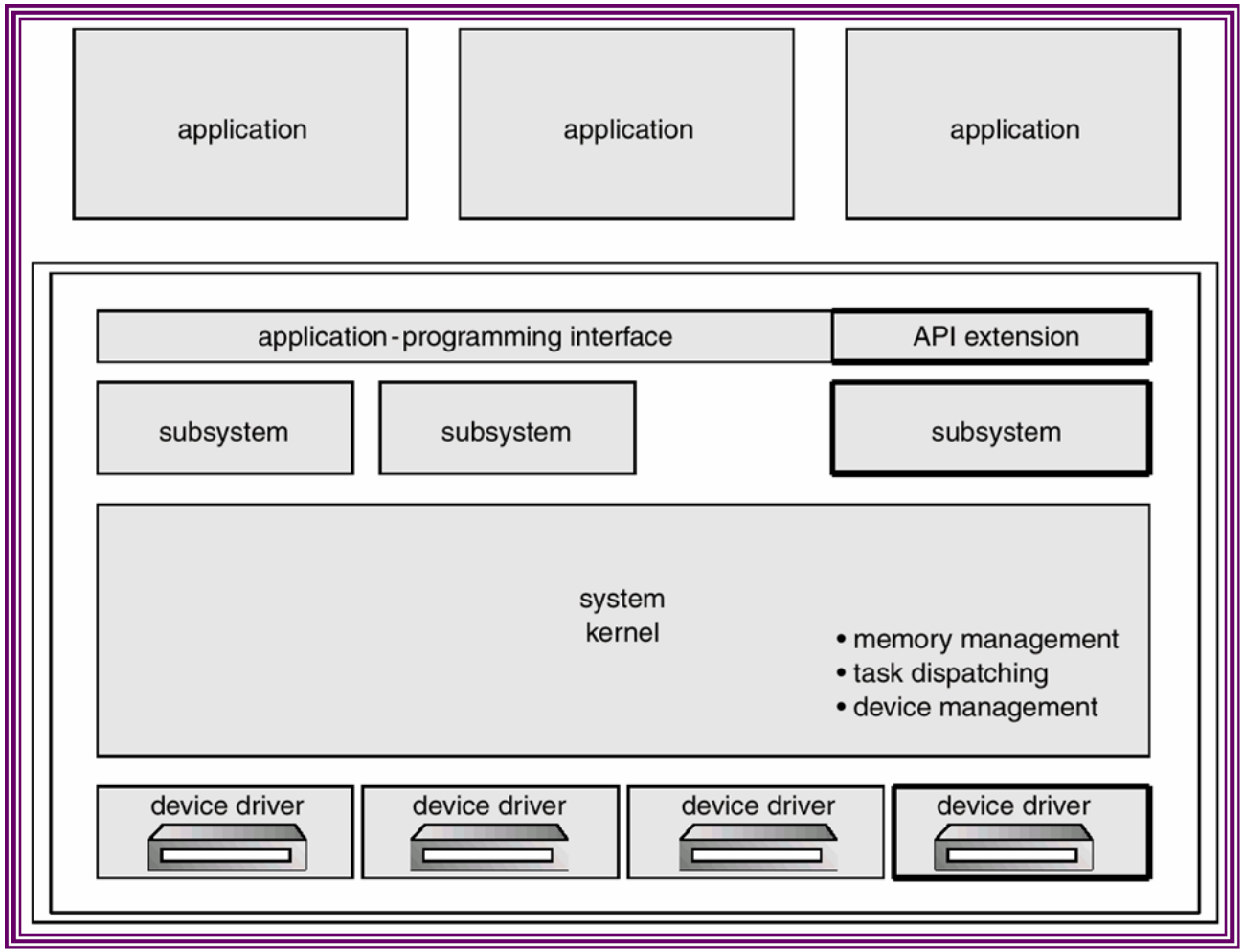
- Example 2:

Layered structure (3/4)

- Example 2:
  - The IBM OS/2 operating system

Layered structure (3/4)

- Example 2:
  - The IBM OS/2 operating system



Layered structure (4/4)

Layered structure (4/4)

- The major difficulty with the layered structure involves

Layered structure (4/4)

- The major difficulty with the layered structure involves
  1. careful definition of the layers and

Layered structure (4/4)

- The major difficulty with the layered structure involves
  1. careful definition of the layers and
  2. less efficient.

Micro-kernel (1/2)

Micro-kernel (1/2)

- As the Unix operating system expanded, the kernel became large and difficult to manage.

Micro-kernel (1/2)

- As the Unix operating system expanded, the kernel became large and difficult to manage.
  - The **micro-kernel** approach structures the operating system by removing all non-essential components from the kernel and implementing them as system- and user-level programs.

Micro-kernel (1/2)

- As the Unix operating system expanded, the kernel became large and difficult to manage.
  - The **micro-kernel** approach structures the operating system by removing all non-essential components from the kernel and implementing them as system- and user-level programs.
- Which components should remain in the micro-kernel?

Micro-kernel (1/2)

- As the Unix operating system expanded, the kernel became large and difficult to manage.
  - The **micro-kernel** approach structures the operating system by removing all non-essential components from the kernel and implementing them as system- and user-level programs.
- Which components should remain in the micro-kernel?
  - CPU management

Micro-kernel (1/2)

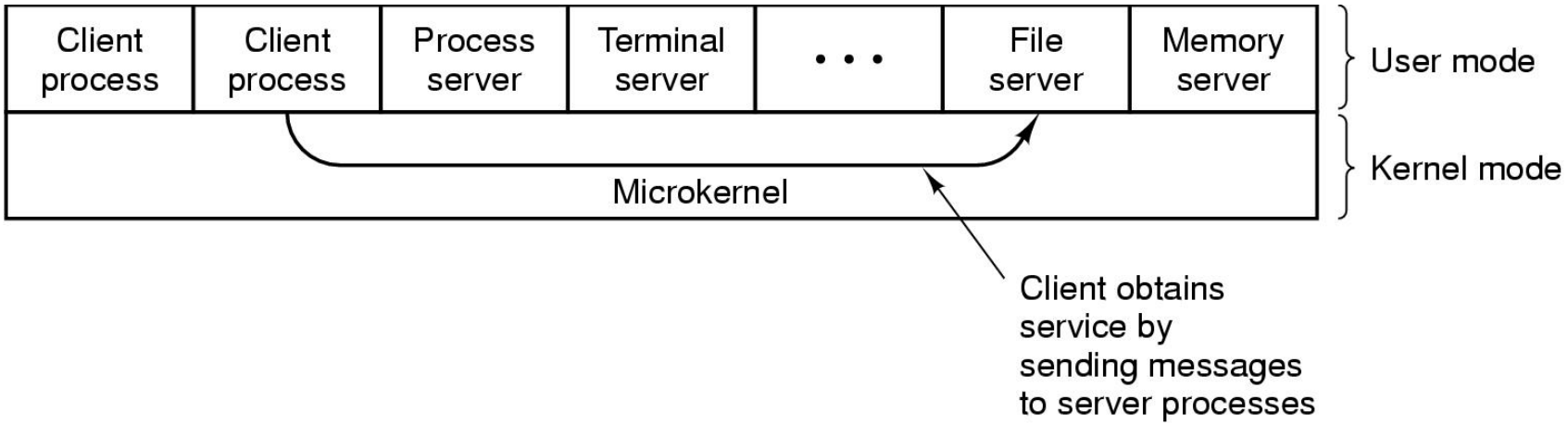
- As the Unix operating system expanded, the kernel became large and difficult to manage.
  - The **micro-kernel** approach structures the operating system by removing all non-essential components from the kernel and implementing them as system- and user-level programs.
- Which components should remain in the micro-kernel?
  - CPU management
  - Memory management

Micro-kernel (1/2)

- As the Unix operating system expanded, the kernel became large and difficult to manage.
  - The **micro-kernel** approach structures the operating system by removing all non-essential components from the kernel and implementing them as system- and user-level programs.
- Which components should remain in the micro-kernel?
  - CPU management
  - Memory management
  - Communication facility

Micro-kernel (1/2)

- As the Unix operating system expanded, the kernel became large and difficult to manage.
  - The **micro-kernel** approach structures the operating system by removing all non-essential components from the kernel and implementing them as system- and user-level programs.
- Which components should remain in the micro-kernel?
  - CPU management
  - Memory management
  - Communication facility



Micro-kernel (2/2)

Micro-kernel (2/2)

- Example 1

Micro-kernel (2/2)

- Example 1
  - The open source **Mach** from Carnegie Mellon University.

Micro-kernel (2/2)

- Example 1
  - The open source **Mach** from Carnegie Mellon University.
    - Used as the kernel of the Apple Mac OS X and the DEC Tru64 Unix.

Micro-kernel (2/2)

- Example 1
  - The open source **Mach** from Carnegie Mellon University.
    - Used as the kernel of the Apple Mac OS X and the DEC Tru64 Unix.
- Example 2

Micro-kernel (2/2)

- Example 1
  - The open source **Mach** from Carnegie Mellon University.
    - Used as the kernel of the Apple Mac OS X and the DEC Tru64 Unix.
- Example 2
  - QNX real-time operating system by QNX Inc.

Micro-kernel (2/2)

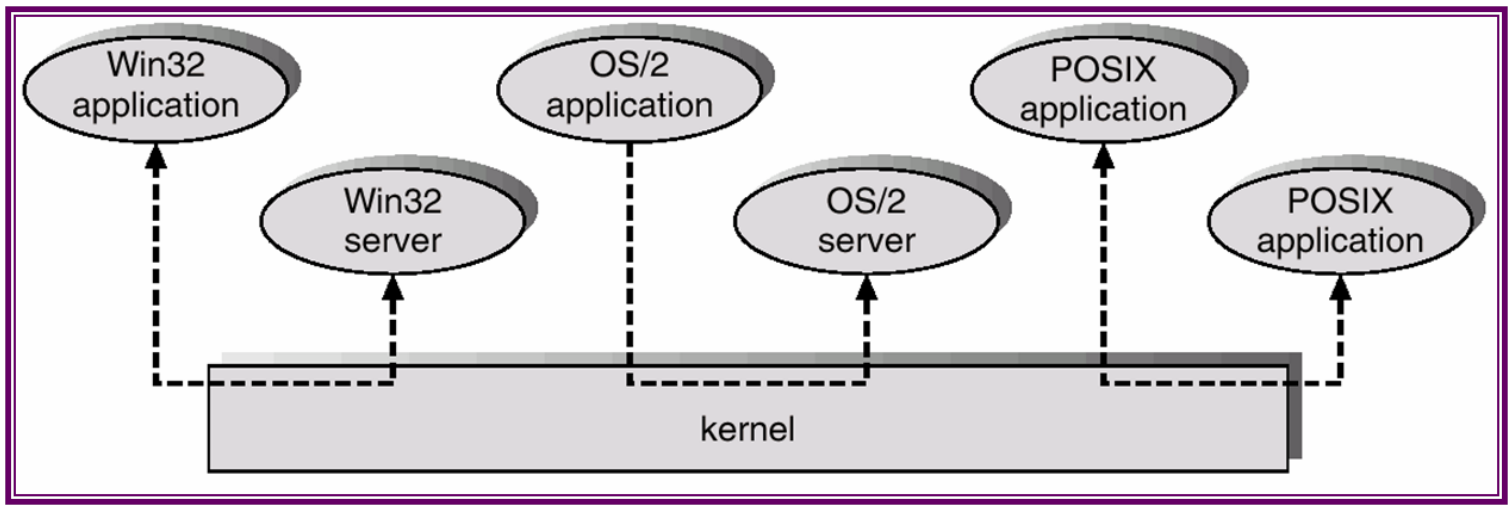
- Example 1
  - The open source **Mach** from Carnegie Mellon University.
    - Used as the kernel of the Apple Mac OS X and the DEC Tru64 Unix.
- Example 2
  - QNX real-time operating system by QNX Inc.
- Example 3

Micro-kernel (2/2)

- Example 1
  - The open source **Mach** from Carnegie Mellon University.
    - Used as the kernel of the Apple Mac OS X and the DEC Tru64 Unix.
- Example 2
  - QNX real-time operating system by QNX Inc.
- Example 3
  - Micorsoft Windows NT/XP

Micro-kernel (2/2)

- Example 1
  - The open source **Mach** from Carnegie Mellon University.
    - Used as the kernel of the Apple Mac OS X and the DEC Tru64 Unix.
- Example 2
  - QNX real-time operating system by QNX Inc.
- Example 3
  - Micorsoft Windows NT/XP



Questions

- Any questions?

Virtual machine (1/4)

Virtual machine (1/4)

- If we take a step further from micro-kernel,

Virtual machine (1/4)

- If we take a step further from micro-kernel,
  - The low-level real hardware is "cloned" into several identical **virtual machines**.

Virtual machine (1/4)

- If we take a step further from micro-kernel,
  - The low-level real hardware is "cloned" into several identical **virtual machines**.
  - That is, virtual machine provides an interface identical to the underlying bare hardware.

Virtual machine (1/4)

- If we take a step further from micro-kernel,
  - The low-level real hardware is "cloned" into several identical **virtual machines**.
  - That is, virtual machine provides an interface identical to the underlying bare hardware.
- Then, the operating system functionality is built on top of the virtual machines.

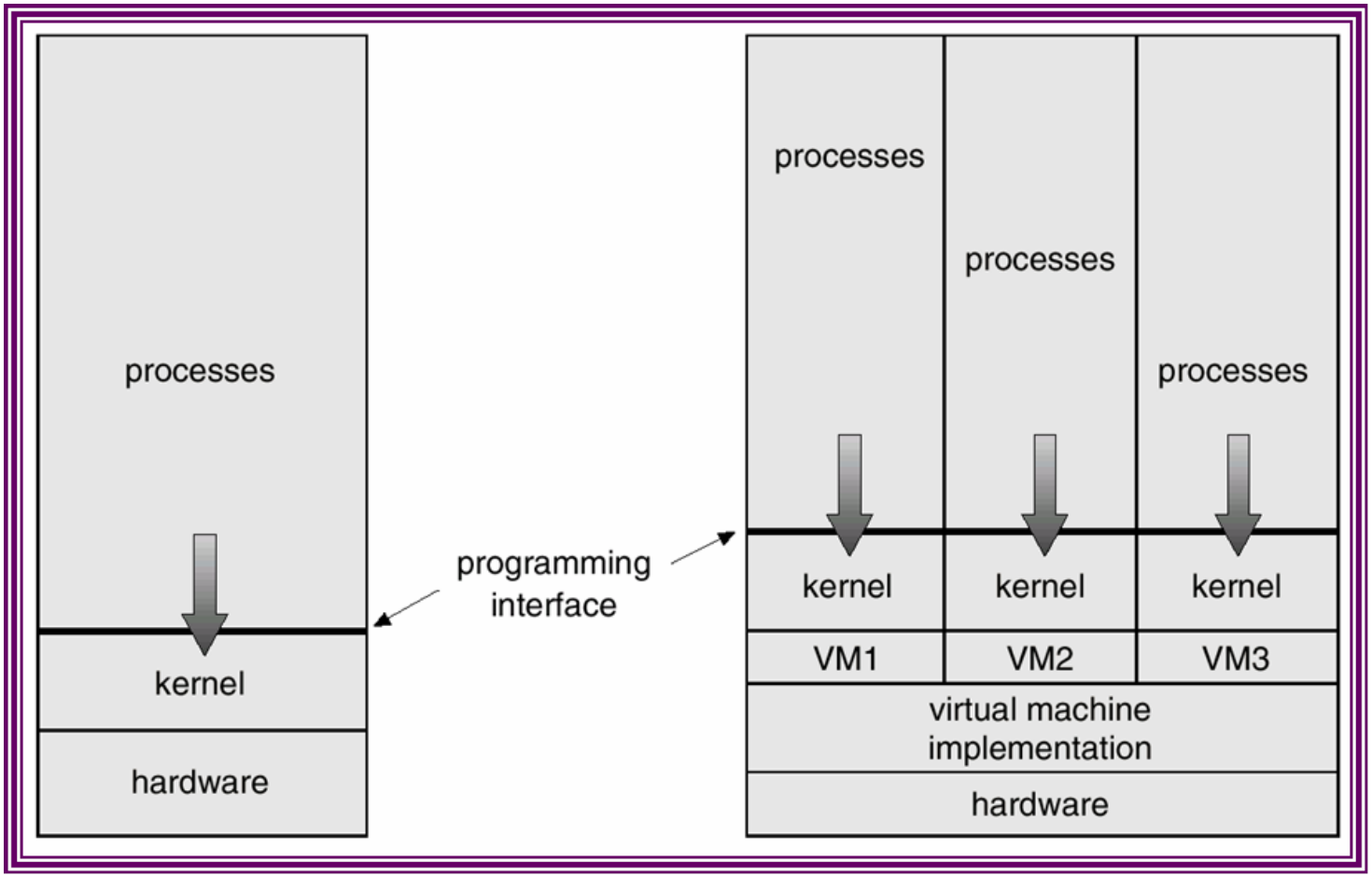
Virtual machine (2/4)

Virtual machine (2/4)

- Virtual machine structure

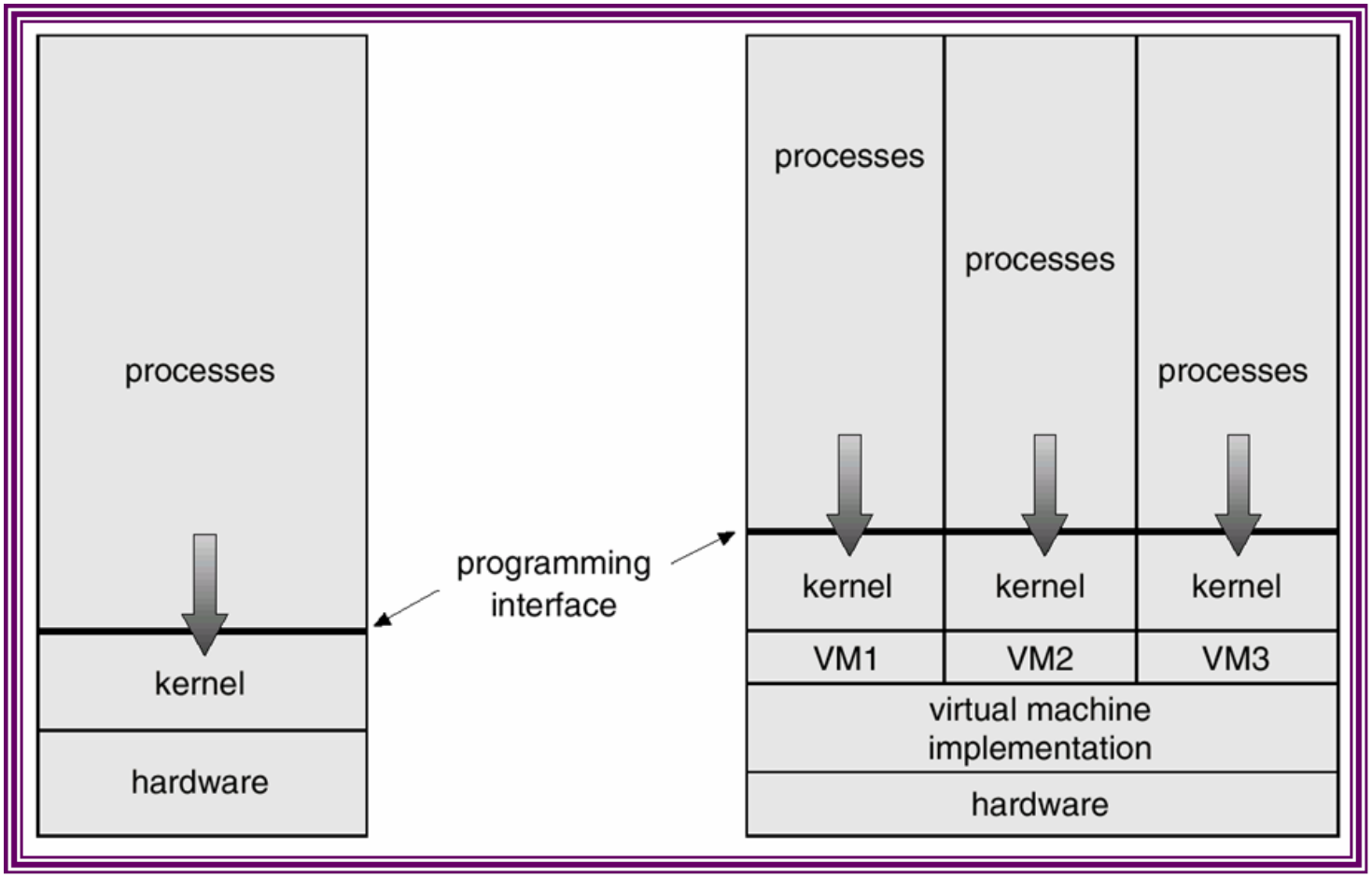
Virtual machine (2/4)

- Virtual machine structure



Virtual machine (2/4)

- Virtual machine structure



Virtual machine (3/4)

Virtual machine (3/4)

- Examples

Virtual machine (3/4)

- Examples
  - IBM VM/370

Virtual machine (3/4)

- Examples
  - IBM VM/370
  - VMware

Virtual machine (3/4)

- Examples
  - IBM VM/370
  - VMware
  - Microsoft Virtual PC

Virtual machine (3/4)

- Examples
  - IBM VM/370
  - VMware
  - Microsoft Virtual PC
  - Sun microsystem's Java Virtual Machine (JVM).

Virtual machine (4/4)

Virtual machine (4/4)

- Advantage and disadvantage of virtual machine

Virtual machine (4/4)

- Advantage and disadvantage of virtual machine
  - The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines.

Virtual machine (4/4)

- Advantage and disadvantage of virtual machine
  - The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines.
    - This isolation, however, permits no direct sharing of resources

Virtual machine (4/4)

- Advantage and disadvantage of virtual machine
  - The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines.
    - This isolation, however, permits no direct sharing of resources
  - A virtual-machine system is a perfect vehicle for operating-systems research and development.

Virtual machine (4/4)

- Advantage and disadvantage of virtual machine
  - The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines.
    - This isolation, however, permits no direct sharing of resources
  - A virtual-machine system is a perfect vehicle for operating-systems research and development.
  - The virtual machine concept is difficult to implement due to the effort required to provide an **exact** duplicate to the underlying machine.

Questions

- Any questions?

Outline

1. Operating system structures
   - Simple structure
   - Layered structure
   - Micro-kernel
   - Virtual machine
2. Operating system design
   - Policy and mechanism
   - Implementation

Operating System design

Operating System design

- Design goals

Operating System design

- Design goals

  User goals operating system should be convenient to use, easy to learn, reliable, safe, and fast.

Operating System design

- Design goals

  User goals operating system should be convenient to use, easy to learn, reliable, safe, and fast.

  System goals operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient.

Policy and mechanism

Policy and mechanism

- **Policies**

Policy and mechanism

- **Policies - what to do**

Policy and mechanism

- **Policies - what to do**
  - For example, users should not be able to read other users' files.

Policy and mechanism

- **Policies - what to do**
  - For example, users should not be able to read other users' files.
- **Mechanisms**

Policy and mechanism

- **Policies - what to do**
  - For example, users should not be able to read other users' files.
- **Mechanisms - how to do**

Policy and mechanism

- **Policies - what to do**
  - For example, users should not be able to read other users' files.
- **Mechanisms - how to do**
  - For example, file permissions are checked on *open* system call.

Policy and mechanism

- **Policies - what to do**
  - For example, users should not be able to read other users' files.
- **Mechanisms - how to do**
  - For example, file permissions are checked on *open* system call.
- The **separation** of policy from mechanism is a very important principle

Policy and mechanism

- **Policies - what to do**
  - For example, users should not be able to read other users' files.
- **Mechanisms - how to do**
  - For example, file permissions are checked on *open* system call.
- The **separation** of policy from mechanism is a very important principle
  - It allows maximum flexibility if policy decisions are to be changed later.

Policy and mechanism

- **Policies - what to do**
  - For example, users should not be able to read other users' files.
- **Mechanisms - how to do**
  - For example, file permissions are checked on *open* system call.
- The **separation** of policy from mechanism is a very important principle
  - It allows maximum flexibility if policy decisions are to be changed later.
- Two extremes:

Policy and mechanism

- **Policies - what to do**
  - For example, users should not be able to read other users' files.
- **Mechanisms - how to do**
  - For example, file permissions are checked on *open* system call.
- The **separation** of policy from mechanism is a very important principle
  - It allows maximum flexibility if policy decisions are to be changed later.
- Two extremes:
  - Micro-kernel

Policy and mechanism

- **Policies - what to do**
  - For example, users should not be able to read other users' files.
- **Mechanisms - how to do**
  - For example, file permissions are checked on *open* system call.
- The **separation** of policy from mechanism is a very important principle
  - It allows maximum flexibility if policy decisions are to be changed later.
- Two extremes:
  - Micro-kernel - all mechanisms, almost no policy

Policy and mechanism

- **Policies - what to do**
  - For example, users should not be able to read other users' files.
- **Mechanisms - how to do**
  - For example, file permissions are checked on *open* system call.
- The **separation** of policy from mechanism is a very important principle
  - It allows maximum flexibility if policy decisions are to be changed later.
- Two extremes:
  - Micro-kernel - all mechanisms, almost no policy
  - The Apple Macintosh

Policy and mechanism

- **Policies - what to do**
  - For example, users should not be able to read other users' files.
- **Mechanisms - how to do**
  - For example, file permissions are checked on *open* system call.
- The **separation** of policy from mechanism is a very important principle
  - It allows maximum flexibility if policy decisions are to be changed later.
- Two extremes:
  - Micro-kernel - all mechanisms, almost no policy
  - The Apple Macintosh - policy and mechanism are bound together

The operating system implementation

The operating system implementation

- Traditionally written in assembly language, operating systems can now be written in higher-level languages.

The operating system implementation

- Traditionally written in assembly language, operating systems can now be written in higher-level languages.
- Code written in a high-level language:

The operating system implementation

- Traditionally written in assembly language, operating systems can now be written in higher-level languages.
- Code written in a high-level language:
  - can be written faster.

The operating system implementation

- Traditionally written in assembly language, operating systems can now be written in higher-level languages.
- Code written in a high-level language:
  - can be written faster.
  - is more compact.

The operating system implementation

- Traditionally written in assembly language, operating systems can now be written in higher-level languages.
- Code written in a high-level language:
  - can be written faster.
  - is more compact.
  - is easier to understand and debug.

The operating system implementation

- Traditionally written in assembly language, operating systems can now be written in higher-level languages.
- Code written in a high-level language:
  - can be written faster.
  - is more compact.
  - is easier to understand and debug.
  - is easier to port to some other hardwares.

Questions

- Any questions?