Operating System Overview

操作系统概述

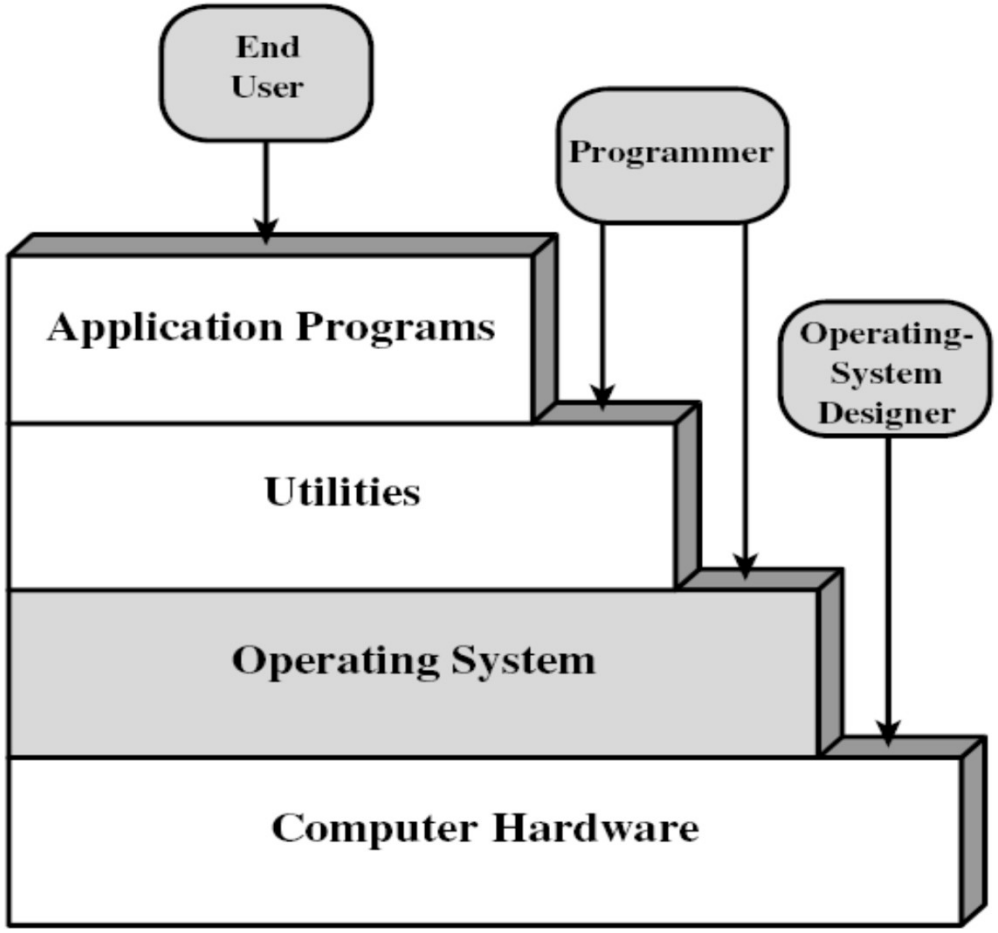
Chapter 2

Contents

- What Is an Operating System?
- The History of Operating Systems
- Operating System Concepts
- System Calls
- Operating System Structure

What Is an Operating System?

Layers of Computer System



What is an OS?

- **An operating system is a program that:**
  - manages the computer hardware,
  - controls the execution of application programs,
  - and acts as an interface between applications and hardware
- **Two basic functions of OS:**
  - OS as **Extended Machine (or Virtual Machine)**
  - OS as **Resource Manager**

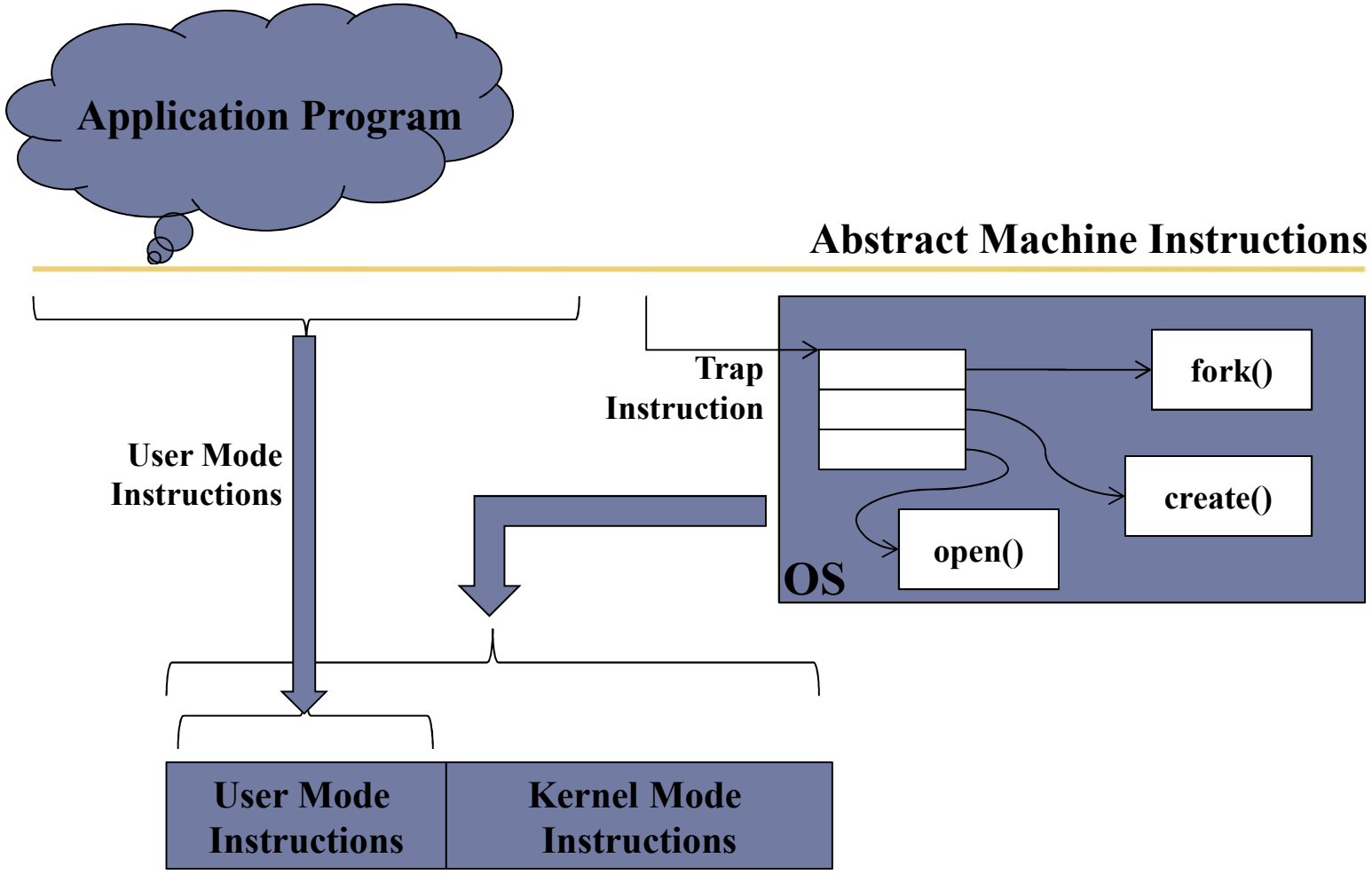
Two Basic Functions

- ***As an Extended Machine***
  - *It is a **top-down** view*
  - *Convenience: to make the computer more convenient to use*
  - *How? →**Resource Abstraction***
- ***As a Resource Manager***
  - *It is a **bottom-up** view*
  - *Efficiency: to allow computer system resources to be used in an efficient manner*
  - *How? →**Resource Sharing***

OS as an Extended Machine

- ***System software hides the details of how the underlying machinery operates.***
- ***Advantage:***
  - *abstractions are used to eliminate tedious details that a programmer otherwise would have to handle.*
- ***Disadvantage:***
  - *while it simplifies the way the application programmer controls the hardware, it can also limit the flexibility by which specific hardware can be manipulated*
  - *Generality (一般性) has its price in specificity (特异性).*

The Abstract Machine Interface



OS as a Resource Manager (1)

- **The job is to provide an orderly and controlled allocation of the processors, memories, and I/O devices among the various programs competing for them**
- **Why necessary?**
  - **An Example: multiple programs are competing for printing their outcomes.**

OS as a Resource Manager (2)
Resource Sharing

- **Concurrent Execution vs Parallel Execution**
  - **Concurrent**: when **either** multiple operations appears **or** they really are executing at the same time.
  - **Parallel**: when there are **true simultaneous** operations

OS as a Resource Manager (3)
Two Kinds of Sharing

- **Space-Multiplexed Sharing**
  - A resource is divided into two or more distinct units, and then the individual parts are allocated to processes.
  - For examples: the main memory and the disks
- **Time-Multiplexed Sharing**
  - One process can use the entire resource for a period of time, and then another process……
  - One example: only one processor
  - Another example: a taxi driver, a barber

Kernel

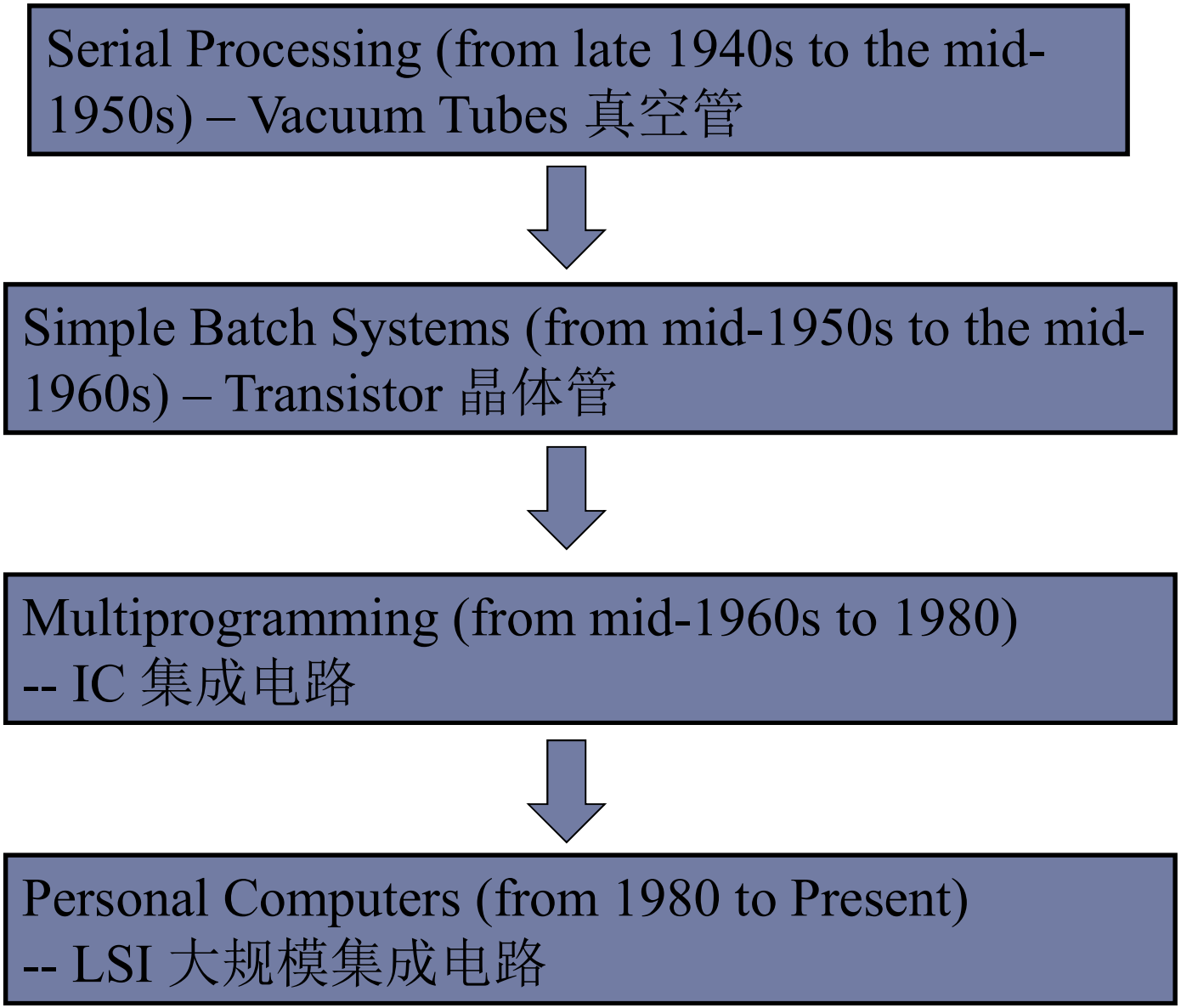
- **Portion of operating system that is in main memory:**
  - **The kernel,**
  - **Other portions of the operating system currently in use**
- **Kernel or Nucleus contains most-frequently used functions in the operating system.**
  - **Running at all times on a computer**

Ease of Evolution

- Why does an OS evolve?
  - Hardware upgrades and new types of hardware
  - New services
  - Fixes
- Design requirements to make evolution easier.
  - Modular in construction
  - Interfaces between modules defined clearly
  - Interfaces between modules well documented
  - ……

The History of Operating Systems

The history of operating systems is really a history of resource-driven choices


Serial Processing (from late 1940s to the mid-1950s) – Vacuum Tubes 真空管
Simple Batch Systems (from mid-1950s to the mid-1960s) – Transistor 晶体管
Multiprogramming (from mid-1960s to 1980)
-- IC 集成电路
Personal Computers (from 1980 to Present)
-- LSI 大规模集成电路

Serial Processing
串行处理

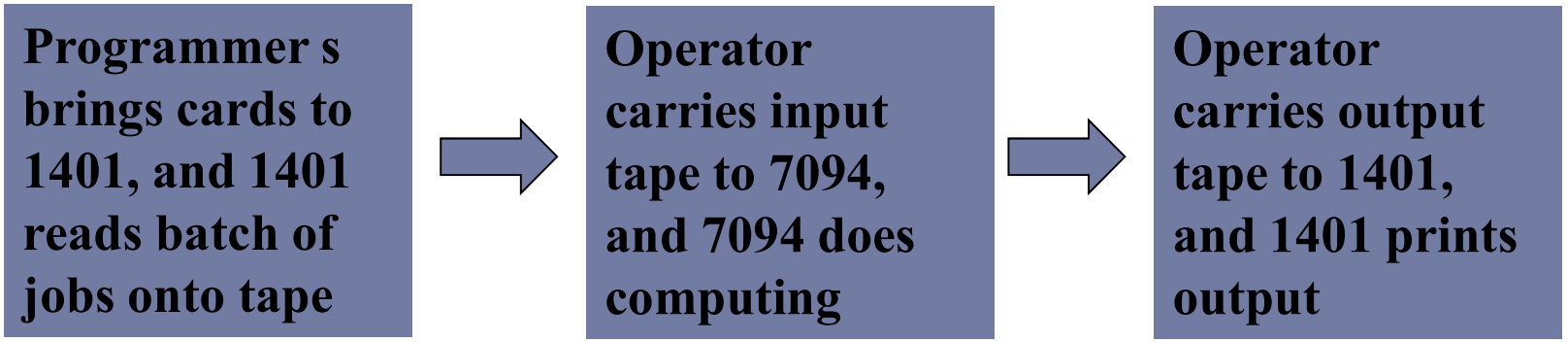
- ***Earliest computers (from late 1940s to the mid-1950s)***
  - *The first computers used **mechanical relays (机械继电器)**, very slow.*
  - *Later, relays were replaced by **vacuum tubes (真空管/电子管)***
  - *Input: initially **plugboards (插线板)**, replaced by **punched cards** (early 1950s)*
- ***A single program-in-execution at a time (essentially no operating system)***
  - *Machines run from a console (控制台) with display lights (显示灯) and toggle switches (拨动开关), input device, and printer*

Serial Processing Problems

- ***No need for resource sharing!!***
  - ***The only purpose of the system software: to simplify device programming through abstraction***
- ***Two main problems:***
  - ***Scheduling:*** *used a hardcopy sign-up sheet (预约表) to reserve machine time.*
    - *Problems? Too much time →Waste!! (It is almost impossible to reserve the exact quantity of time that you need).*
  - ***Setup time:*** *included loading the compiler, source program, saving compiled program, and loading and linking*

Simple Batch Systems
简单批处理系统

- ***Transistor (晶体管) was introduced in the mid-1950s.***
  - *The first batch operating system was developed in the **mid-1950s** by General Motors for use on an IBM701*



IBM 1401 is an inexpensive computer, while much more expensive computers such as IBM 7094 are used for real computing

Simple Batch Systems
简单批处理系统

- ***Central Idea: The Use of Monitor (监控程序)***
  - *Software that controls the running programs*
  - *The user no longer has the direct access to the machine*
    - *The user submits the job on cards or tape to a computer operator*
  - *The operator batches the jobs together sequentially and places the entire batch on an input device used by monitor.*
  - *Program branches back to monitor when finished. Then the monitor begins loading the next program.*

Job Control Language (JCL)
作业控制语言

- ***Job Control Language is a special type of programming language***
- ***It Provides everything about the job execution to the monitor (or the OS)***
  - ***what compiler to use***
  - ***what data to use***
  - ……

```
$JOB
$FTN
…
…  FORTRAN instructions
…
$LOAD
$RUN
…
…  Data
…
```

Hardware Features

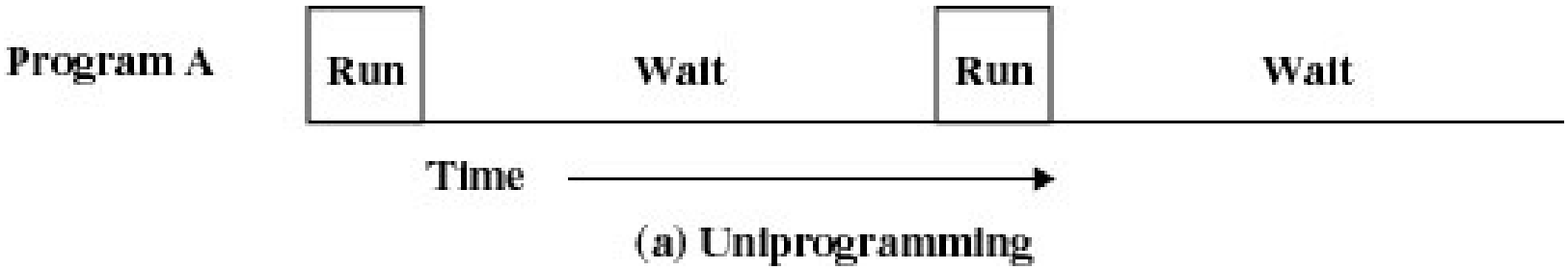
- *Memory protection*
  - *Do not allow the memory area containing the monitor to be altered*
- *Timer*
  - *Prevents a job from monopolizing (壟斷) the system (The timer is set at the beginning of each job)*
- *Privileged Instructions*
  - *The privileged instructions can be executed only by the monitor. For example, I/O instructions are privileged to prevent a user program from accidentally reading job control instructions from the next job*
- *Interrupts*
  - *Early computers did not have this capability. This feature gives the OS more flexibility in relinquishing control to and regain control from user programs*

Multiprogrammed Batch Systems
多道程序批处理系统

- ***There are multiple jobs resident in the main memory.***
  - ***They are all between the beginning point and ending point of execution.***
  - ***These jobs share the computer system resources.***

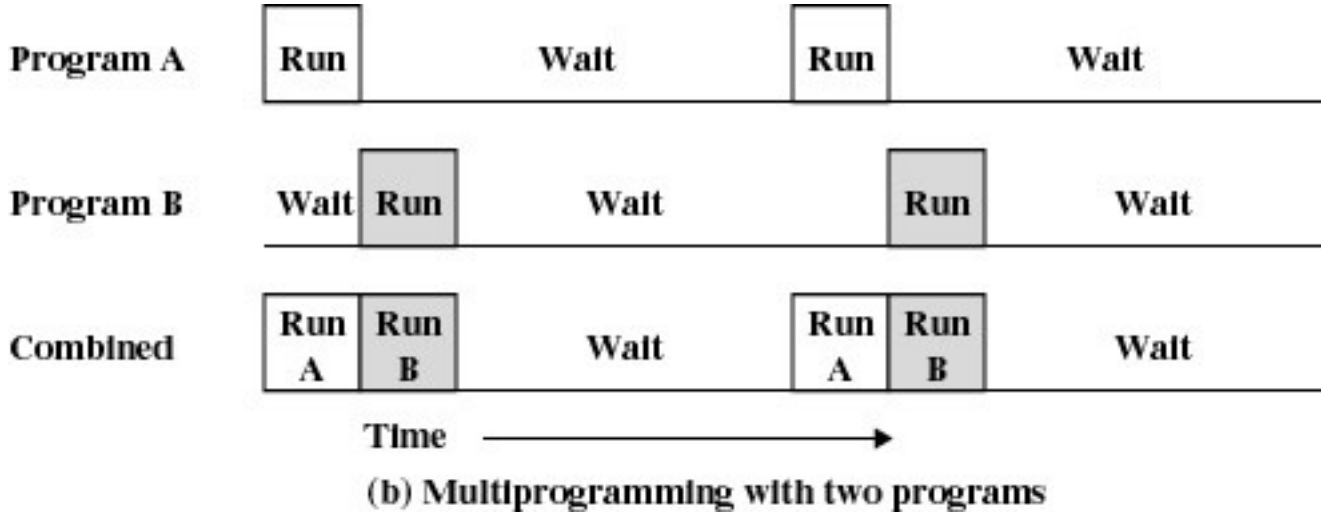
Uniprogramming
单道程序设计

- **Processor is often idle**
  - **Processor must wait for I/O instruction to complete before preceding**
    - **I/O devices are slow compared to the processor**



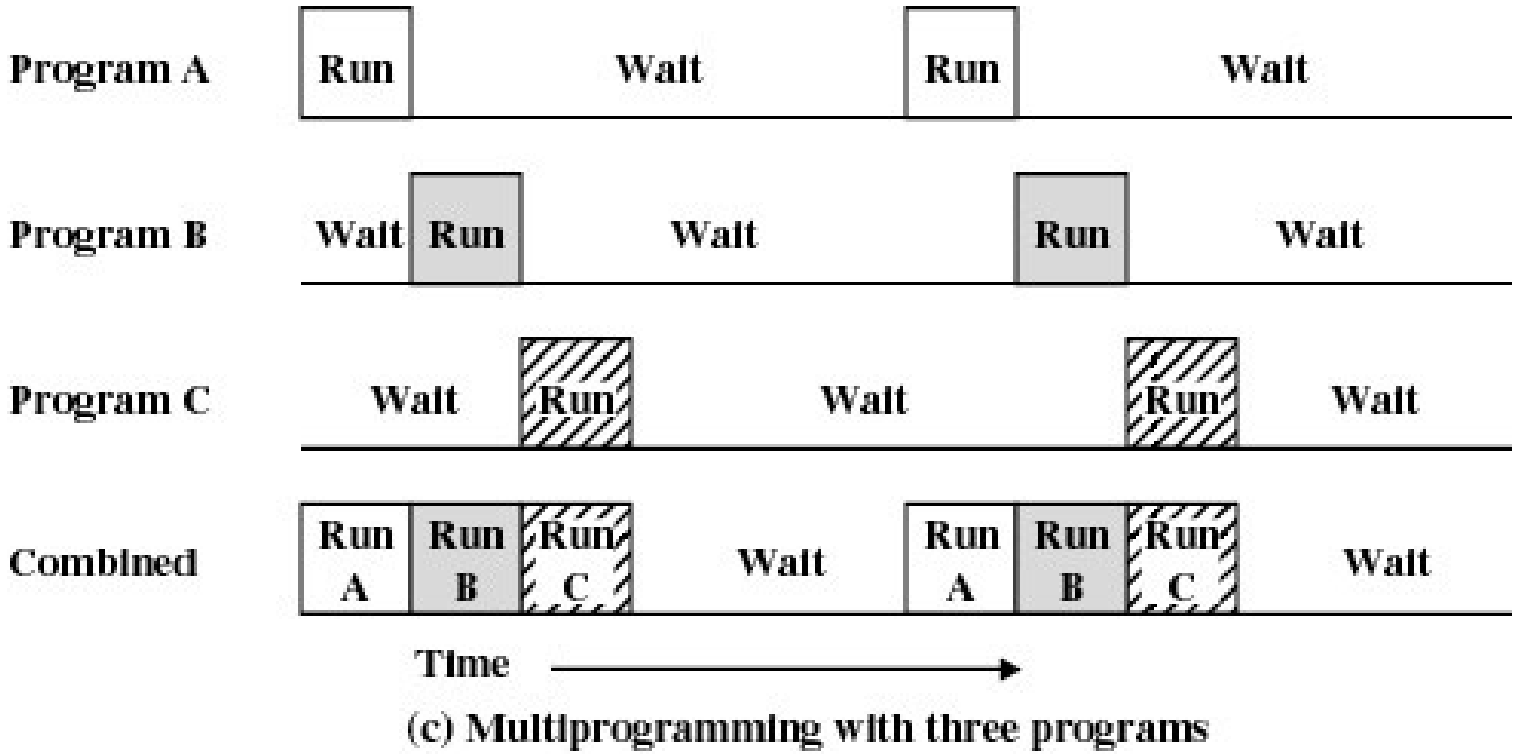
Multiprogramming
多道程序设计

- **When one job needs to wait for I/O, the processor can switch to the other job**



(b) Multiprogramming with two programs

Multiprogramming
多道程序设计



(c) Multiprogramming with three programs

Spooling
Simultaneous Peripheral Operation On-Line

- **As soon as jobs were brought to the computer room, they are read from cards onto the disk**
  - **Then, whenever a running job finished, the OS could load a new job from the disk into the now-empty partition and run it.**

What does a multiprogramming batch system rely on?

- **The *hardware* that supports *I/O interrupts and DMA* (direct memory access).**
- **Multiprogramming OSes are fairly sophisticated compared to uniprogramming systems**
  - **Some form of *memory management* is required to keep several jobs in main memory**
  - **Processor must decide which job to run (*Scheduling algorithm* is needed)**

A Simple Example (1)

- A computer with 256K words of available memory (not used by OS)
- Three programs (JOB1, JOB2, and JOB3)

A Simple Example (2)

	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min.	15 min.	10 min.
Memory required	50K	100 K	80 K
Need disk?	No	No	Yes
Need terminal	No	Yes	No
Need printer?	No	No	Yes

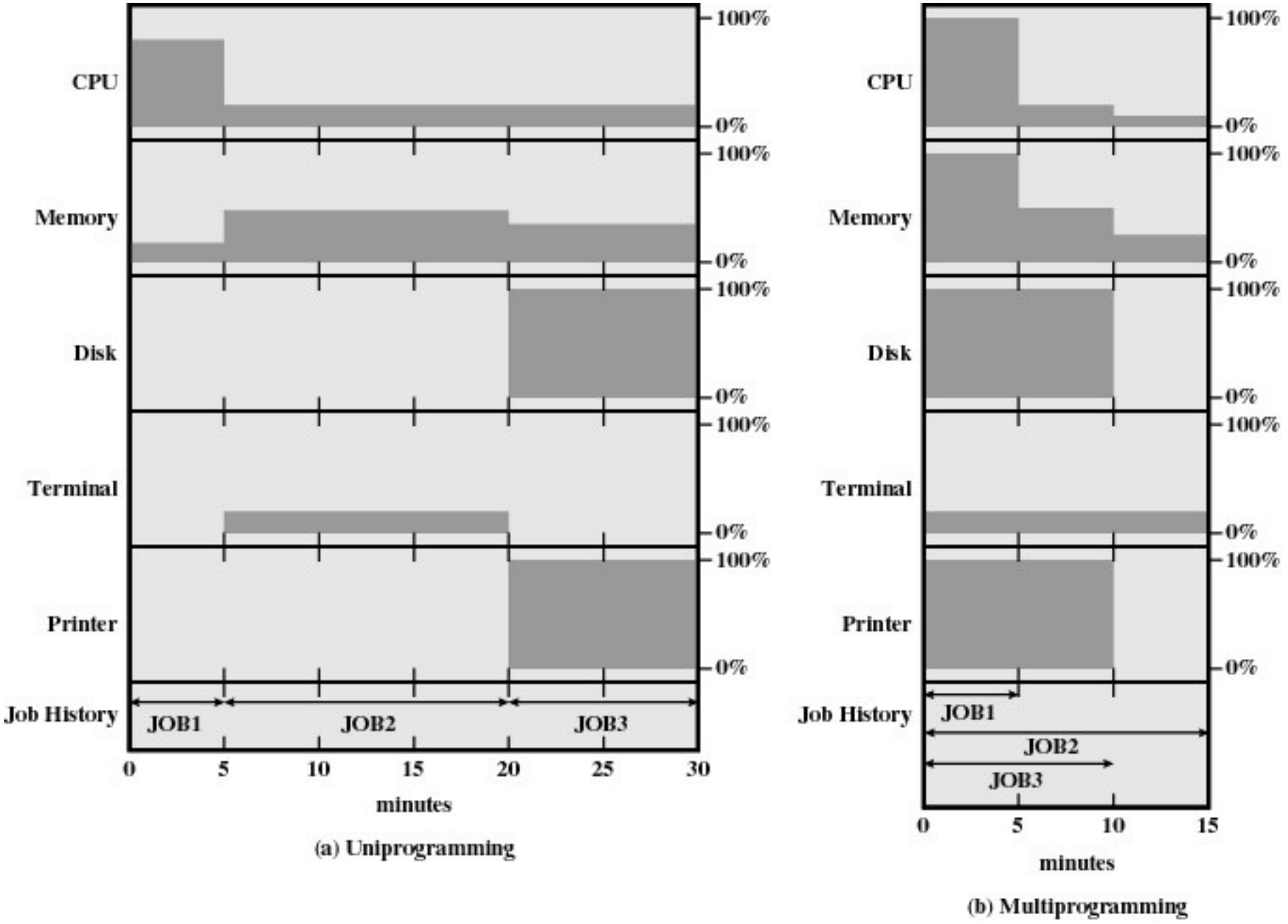


Figure 2.6 Utilization Histograms

A Simple Example (3)
Effects of Multiprogramming

	Uniprogramming	Multiprogramming
Processor use	22%	43%
Memory use	30%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min.	15 min.
Throughput rate	6 jobs/hr	12 jobs/hr
Mean response time	18 min.	10 min.

Problem about Process Use

- In what condition, can we get the results in table 2.2.
  - ?????

Time Sharing Systems
分时系统

- ***Time sharing system began to become popular in 1970s.***
  - **The desire for quick response time paved the way for timesharing**
- ***The objective of timesharing systems is to support multiple interactive users.***
  - ***Using multiprogramming to handle multiple interactive jobs***
  - ***Processor's time is shared among multiple users***
  - ***Multiple users simultaneously access the system through terminals***

Time Sharing Systems

- ***Several time-sharing systems:***
  - ***CTSS**: Compatible Time Sharing System*
    -- developed in the mid-1960s at MIT
  - ***Multics**: MULTiplexed Information and Computing Service.*
    -- It replaced CTSS.
  - ***UNIX**: It is still a leading operating system technology now.*

Batch Multiprogramming versus Time Sharing

Both batch processing and time-sharing operating systems use multiprogramming.

Key differences are listed as following:

	Batch Multiprogramming	**Time Sharing**
Principal objective	**Maximize processor use**	**Minimize response time**
Source of directives (指令源) to operating system	Job control language commands provided with the job	Commands entered at the terminal

Personal Computers and Workstations

- *Timestones of personal computers*
  - *In April 1977, the Apple II was launched*
  - *In August 1981, IBM Personal Computer was released*
- *In 1980s, personal computer system software was generally designed to allow one user to execute one program at a time.*
  - *The primary requirement for system software was to provide hardware abstraction*

Personal Computers and Workstations

- ***By 1995, personal computer hardware had become so sophisticated that these machines began to compete with workstations***
  - *Meanwhile, Microsoft upgraded its OS offerings with Windows NT and Windows 95.*
- ***Today, there is no distinction between a personal computer and a workstation***

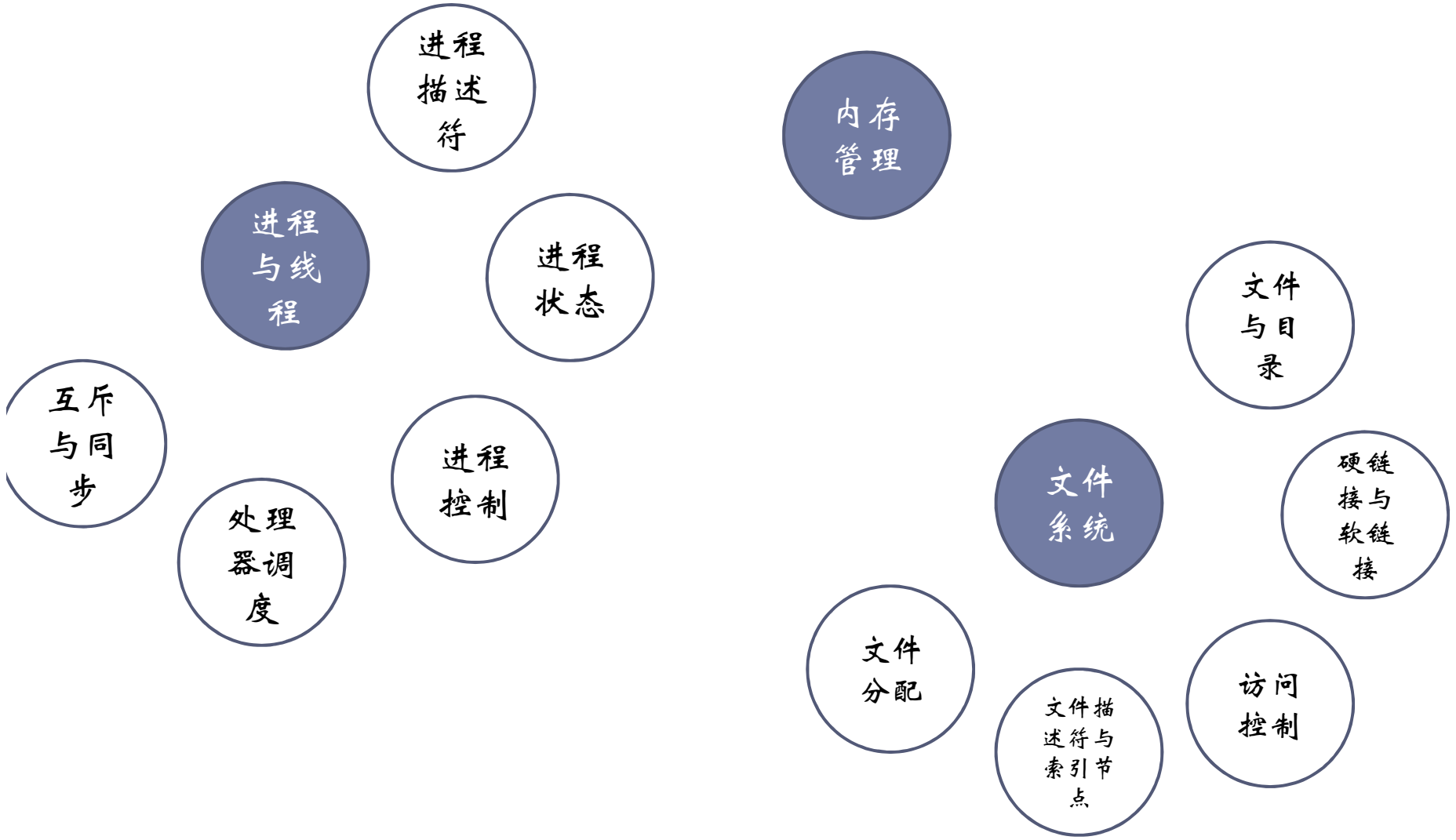
Personal Computers and Workstations

- *In 1982, Sun announced its first small computer (workstation)*
  - *Workstations were configured with enough resources that it made sense to use timesharing OS technology, especially multiprogramming, for their OS*
  - *Almost all workstations eventually used some form of UNIX*

Operating System Concepts

Basic Concepts in Operating Systems

- **Some basic concepts and abstractions provided by most operating systems:**
  - **Processes (进程)**
  - **Address Spaces(内存管理)**
  - **Files**
  - **Input/Output**
  - **Protection**


进程描述符
进程与线程
进程状态
互斥与同步
进程控制
处理器调度
内存管理
文件与目录
文件系统
硬链接与软链接
文件分配
文件描述符与索引节点
访问控制

Processes (1)
Definition

- ***Process is a key concept in all operating systems. It has many definitions:***
  - *A program in execution*
  - *An instance of a program running on a computer*
  - *The entity that can be assigned to and executed on a processor*
  - *A unit of activity characterized by a single sequential thread of execution, a current state, and an associated set of system resources*

Processes (2)
Why we need processes?

- **Multiprogramming batch operation**
  - In response to signals indicating the completion of I/O transactions, the processor is switched among the various programs residing in main memory.
- **Time-sharing**
  - Key Objective: to be responsive (及时响应) to the needs of the individual user
- **Real-time transaction systems**
  - A number of users are entering queries or updates against a database

Processes (3)

- **Process tree:**
  - A process can create one or more other processes (referred to as child processes), and these processes in turn can create child processes
- **Interprocess communication**
  - Related processes that are cooperating to get some job done often need to communicate with one another and synchronize their activities.

Address Spaces

- **For each process, it has an associated address space which the process can read and write.**
  - **The address space contains the executable program, the program's data, and its stack.**

Address Spaces

- **The address space is decoupled from the machine's physical memory**
  - **Allows programmers to address memory from a logical point of view, without regard to the amount of main memory physically available**
  - **The address space can be either larger or smaller than the physical memory.**
- **Management of address spaces and physical memory is also an important part of an OS.**

File System

- **File system is virtually supported by all operating systems.**
  - **It implements long-term store**
  - **Information stored in named objects called files**
- **Most operating systems have the concept of a directory as a way of grouping files together.**

Information Protection and Security

- **Access control**
  - regulate user access to the system
- **Information flow control**
  - regulate flow of data within the system and its delivery to users
- **Certification**
  - proving that access and flow control perform according to specifications

User Operating-System Interface

How Users Interface with OS?

- **Two Approaches:**
  - **Command Interpreter, or Command-Line Interface:**
  - **GUI (Graphical User Interface):**

Command Interpreter

- Main function: to get and execute the next user-specified command
- Two general ways for command implementation:
  - The command interpreter itself contains the code to execute the command
  - Commands an be implemented through system programs. The command interpreter merely uses the command to identify a file to be loaded into memory and executed.

Graphical User Interface

- A GUI provides a mouse-based window-and-menu system as an interface
  - Mouse is moved to position its pointer on icons which represent programs, files, directories, and system functions
  - Depending on the mouse pointer's position, clicking a button on the mouse can invoke a program, select a file or directory, or pull down a menu that contains commands.

System Calls

System Calls

- *The set of system calls is the interface between the operating system and the user programs.*
  - *The system calls available in the interface vary from operating system to operating system*
- *The actual mechanics of issuing a system call are highly machine dependent.*

Parameter Passing

- **Three general methods to pass parameters to the OS:**
  - *The simplest way →to pass the parameters in registers. (when there are more parameters than registers, how to handle?)*
  - *The second way →to store the parameters in a block or table in memory, and to pass the address of the block as a parameter in register (Linux uses this way)*
  - *The third way →to place or push the parameters onto the stack (and then the OS to pop off the parameters from the stack)*

Categories of System Calls

- ***System calls can be grouped into several categories:***
  - *Process Control*
  - *File Management*
  - *Device Management*
  - *Information Management*
  - *Communications*

System Calls
Process Control (1)

- A running process needs to be able to halt its execution either normally (`end`) or abnormally (`abort`)
- A process (or job) executing one program may want to `load` and `execute` another program
  - When to return control to the original process? Never, after loaded, or after termination?
- `create process` or `submit job`
  - If both programs continue concurrently, these system calls often exist

System Calls
Process Control (2)

- To determine and reset the attributes of a job or process
  - `get process attributes` and `set process attributes`)
- To terminate a process (`terminate process`)
- After creating new jobs or process, we may want to wait for a certain amount of time (`wait time`);
- we may want to wait for a specific event to occur (`wait event`).
- `allocate` and `free` memory

System Calls
File Management

- `create` and `delete` files
- `open` and `close` files
- `read`,`write`, and `reposition` files
- In addition, we may need the same set of operations for directories
  - If we have a directory structure for file organization
- `get file attribute` and `set file attribute`

System Calls
Device Management

- `request` and `release` devices
- `read` and `write` devices

- ***Many operating systems (including UNIX and DOS) merge the files and devices into a file-device structure***
  - ***Because the similarity between them are so great***
  - ***In this case, I/O devices are identified by special file names***

System Calls
Information Maintenance

- To transfer information between the user program and the operating system
  - To return the current time and date (`time` and `date`)
  - To access the information about all its processes (`get process attributes` and `set process attributes`)
  - To return information about the system (such as the version number of the operating system, the amount of free memory or disk space, and so on)

System Calls
Communication (1)

- There are two common models of communication
  - Message-Passing Model: A connection should be established before communication can take place.
  - Shared-Memory Model: Processes use `map memory` system calls to gain access to regions of memory owned by other processes.
    - OS tries to prevent one process from accessing another process' memory. Share memory requires that several processes agree to remove this restriction
    - The processes are responsible for mutual exclusion.

System Calls
Communication (2)

- System Calls for Message Passing Model
  - `open connection` and `close connection`
    - The recipient process usually must give its permission for communication to take place by `accept connection` call
  - The two processes exchange messages by `read message` and `send message`

Operating System Structure

Some Typical OS Structures

- Monolithic Kernels (单体内核)
- Layered Systems (分层系统)
- Microkernels (微内核)
- Virtual Machines (虚拟机)
- Exokernels (外核)

Monolithic Kernels

- **Monolithic kernel**
  - The structure here is no structure.
    - Each procedure is free to call any other one.
  - Typically, a monolithic kernel is implemented as a single process
    - All functions in OS share the same process space
  - Examples: MS-DOS and UNIX

Layered Systems (1)

- **Layered approach is one modular method to OS design and development**
  - **An operating system can be organized as a hierarchy of layers**
    - **The bottom layer – hardware**
    - **The top layer – user interface**
  - **Each layer is constructed upon the one below it**

Layered Systems (2)

- **Benefits**
  - Easy to construct and debug
- **Difficulties:**
  - How to appropriate define the various layers
  - Tend to be less efficient.

Microkernels

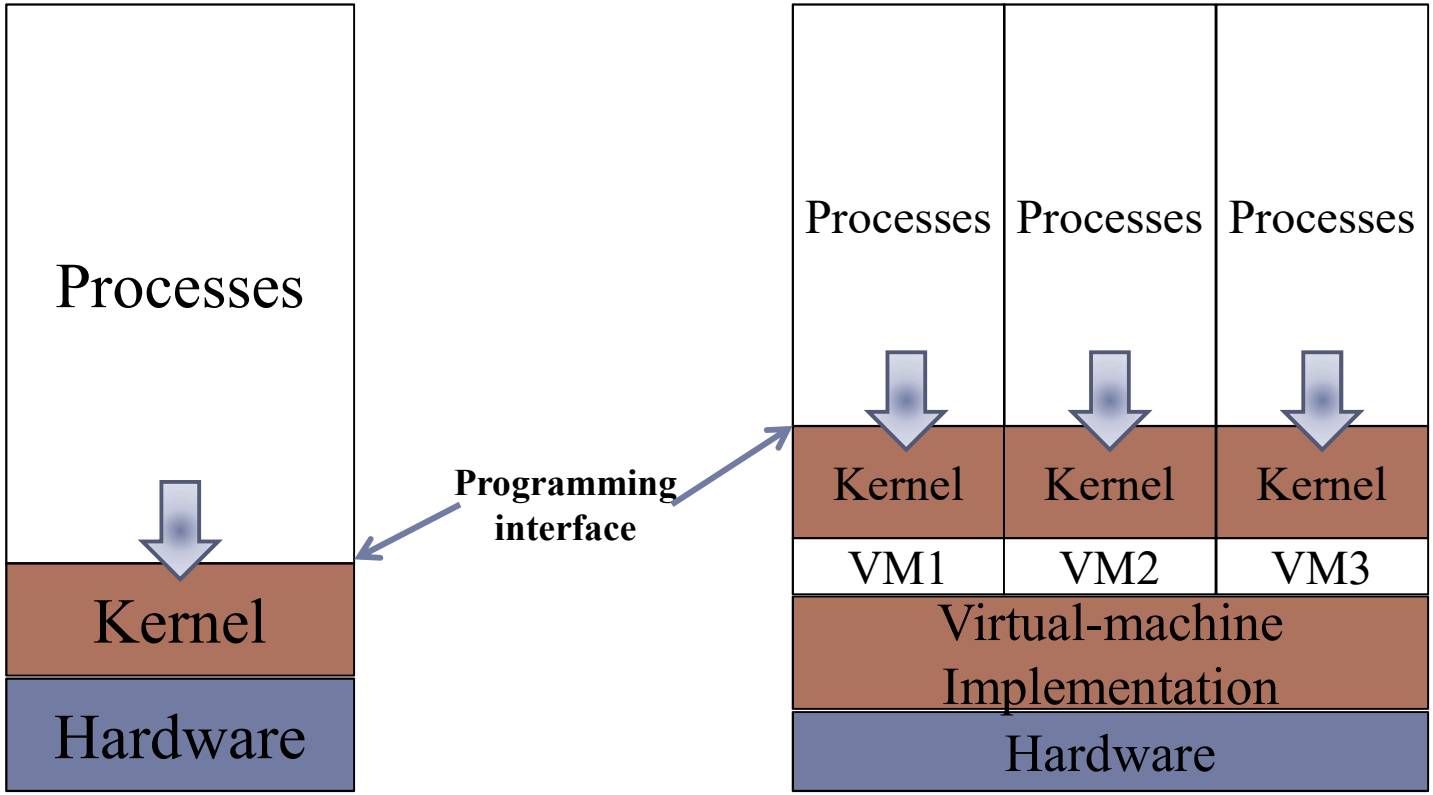
- **Microkernel architecture**
  - Assigns only a few essential functions to the kernel
    - Address space
    - Interprocess communication (IPC)
    - Basic scheduling
  - Other OS services are provided by processes (called servers)→in user mode
  - Examples: Mach

Microkernels

- Benefits:
  - Ease of extending the operating system
  - Provides more security and reliability
- Drawbacks:
  - Decreased performance

Virtual Machines

- **Fundamental Idea**
  - To abstract the **hardware of a single computer** into **several different execution environments**

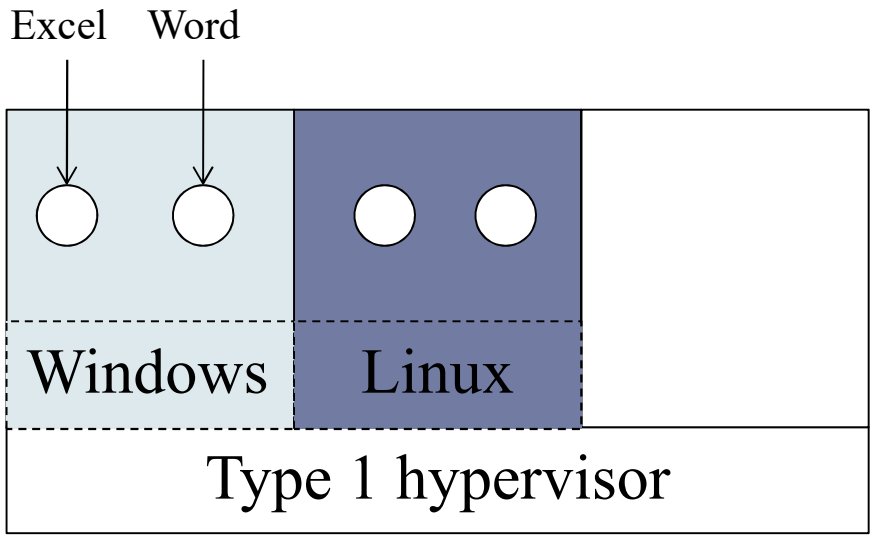


Hypervisor / Virtual Machine Manager
虚拟机管理器

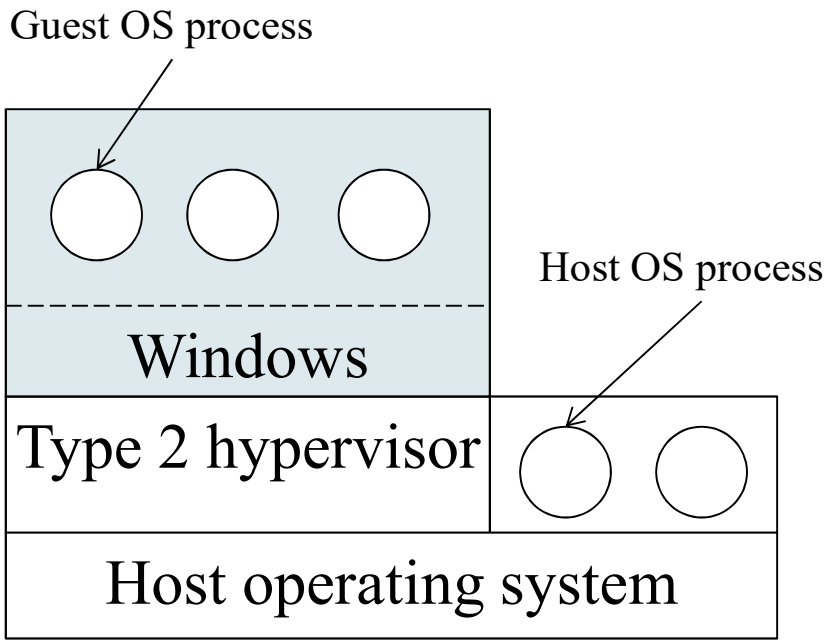
- A hypervisor is one kind of **hardware virtualization** techniques
  - Allow multiple operating systems (guests) to run **concurrently** on a host computer

Type 1 Hypervisor

Type 1 hypervisors run directly on the host's hardware to control the hardware and to manage guest operating systems



Type 2 Hypervisor



Type 2 Hypervisor

- Type 2 hypervisors run as application programs on top of host operating systems
  - After a type 2 hypervisor is started, it reads the installation CD-ROM for the chosen **guest operating system** and installs on a **virtual disk**

A virtual disk is just a big file in the host operating system's file system