Operating Systems

Introduction to Computer

Yu-Ting Wu

(with some slides borrowed from Prof. Tian-Li Yu)

Outline

- What is an operating system
- The history of operating systems
- Operating system architecture
- Coordinating the machine's activities
- Handling competition among processes
- Security

Outline

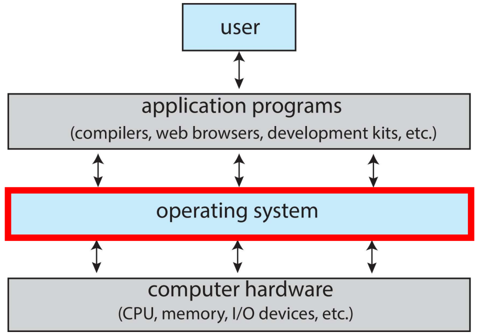
- What is an operating system
- The history of operating systems
- Operating system architecture
- Coordinating the machine's activities
- Handling competition among processes
- Security

What is an Operating System

- An operating system (OS) is a **software program** that acts as an **intermediary** between a user and the computer hardware
  - Execute user programs
  - Make the computer system convenient to use
    - Such that users can focus on their problems
  - Use the computer hardware in an efficient manner

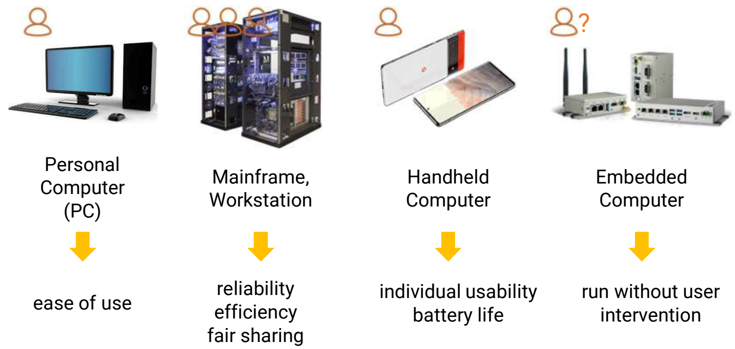
What is an Operating System (cont.)

- An operating system (OS) can be considered as a government or environment provider



Features of Operating Systems

- User view: varies by the types of the computers



Features of Operating Systems (cont.)

- System view: a resource allocator and control program

- **Resource allocator**
  - CPU time
  - Memory space
  - File storage
  - I/O devices

- **Control program**
  - Control execution of user programs
  - Prevent errors and misuse

Examples of Operating Systems

- Windows
- UNIX
- Mac OS
- Solaris
- Linux
- Apple iOS
- Windows phone
- BlackBerry OS
- Nokia Symbian OS
- Google Android

Free and Open-Source OSes

- OS with available source
  - Otherwise: closed-source OS. E.g., MS Windows, iOS
- Examples: GNU/Linux, BSD, UNIX, etc.
- Arguably issues on bugs, security, support

9

Outline

- What is an operating system
- The history of operating systems
- Operating system architecture
- Coordinating the machine's activities
- Handling competition among processes
- Security

10

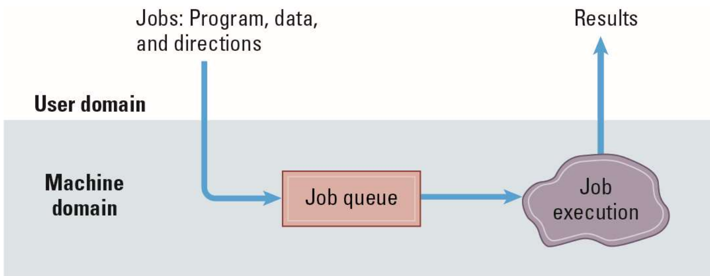
History of Operating Systems

- Batch processing (job queue)
- Interactive and (real-time) processing
- Multi-tasking and time-sharing and
- Multiprocessor machines
- Embedded Systems (specific devices)

11

Batching Process

- Each program is called a "job"
  - Feed by computer operators
- First-in, first-out (FIFO)



12

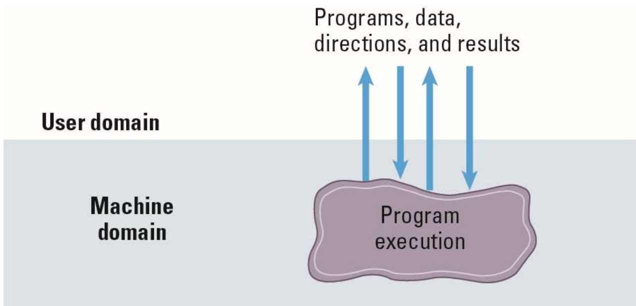
Batching Process (cont.)

Punch card operator

13

Interactive Processing

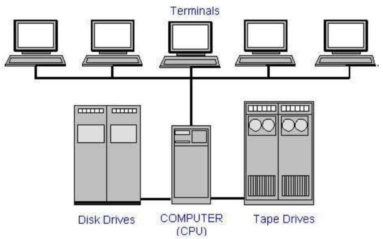
- OS with remote terminals



14

Interactive Processing (cont.)

- Terminals



15

Real-Time Processing

- Real-time OS has well-defined fixed time constraints
  - **Hard real-time system**
    - Processing **must** be done within the constraint
    - Correct operation only if constraints met
  - **Soft real-time system**
    - Missing a timing is serious but does not necessarily result in failure (ex: multimedia)

- Real-time means **on time**! (not fast)

16

Multi-Tasking

- Before multi-tasking, one job at a time
- Example: MS DOS

17

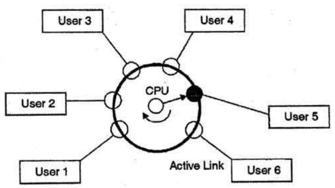
Multi-Tasking (cont.)

- A single user cannot always keep CPU and I/O devices busy
  - E.g., humans and disk I/O are too slow compared to CPU and memory
- Put multiple programs in memory
- OS organizes jobs so that the CPU always has one to execute
  - When a job has to wait (e.g., for I/O), OS **switches to another job**
  - ➡ Increase CPU utilization
  - ➡ Need job and CPU scheduling

18

Multi-Tasking with Time-Sharing

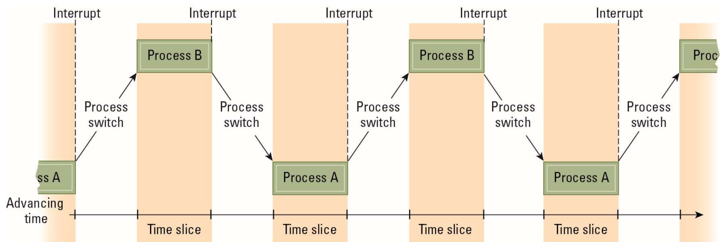
- CPU switches jobs frequently so that users can interact with each job while it is running
  - Only **one** (per core) task is being executed at any given time
  - A logical extension of multi-tasking
  - **Interactivity**!
    - Response time should be less than 1 sec.



19

Context Switch

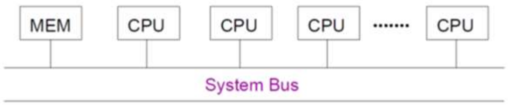
- Kernel saves the state of the old process and loads the saved state for the new process
- Context switch time is **purely overhead**
- Switch time (about 1 ~ 1000 ms) depends on hardware



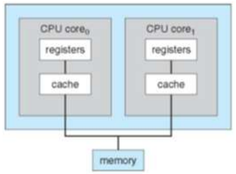
20

Multiprocessor

- More than one processor in close communication sharing bus, memory, and peripheral devices



- The recent trend: from a fast single processor to lots of processors
  - **Multiple cores** over a single chip



21

Outline

- What is an operating system
- The history of operating systems
- Operating system architecture
- Coordinating the machine's activities
- Handling competition among processes
- Security

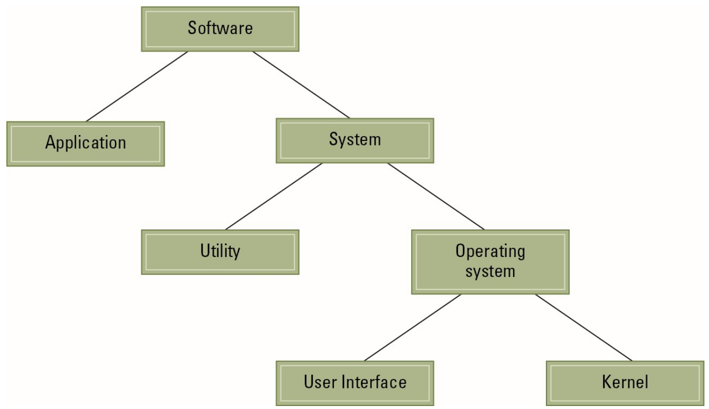
22

Software Classification

- **Application software**
  - Performs specific tasks for users (productivity, games, software development)
- **System software**
  - Provides infrastructure for application software
  - Consists of operating system and utility software

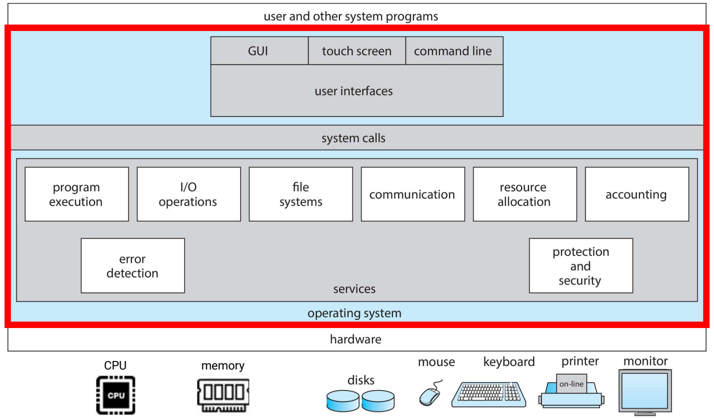
23

Software Classification (cont.)



24

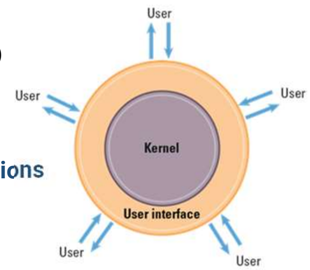
Operating System Components



25

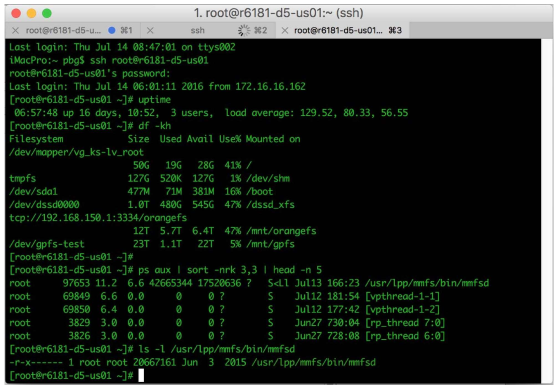
Operating System Components (cont.)

- **User interface:**
  - **Communicates with users**
    - Text-based (Shell)
    - Graphical user interface (GUI)
- **Kernel:**
  - **Performs basic required functions**
    - File manager
    - Device drivers
    - Memory manager
    - Scheduler
    - Dispatcher



26

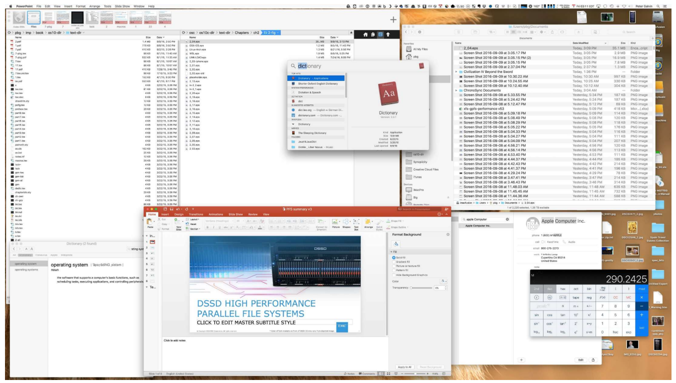
User Interface: Shell



Bourne Shell (default shell of UNIX ver. 7)

27

User Interface: GUI

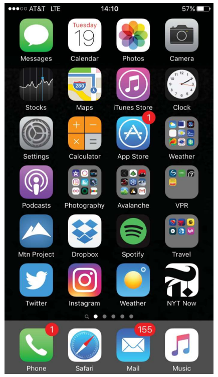


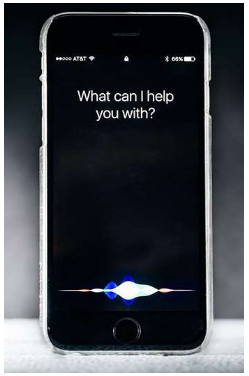
Mac OS X GUI

28

User Interface: Others

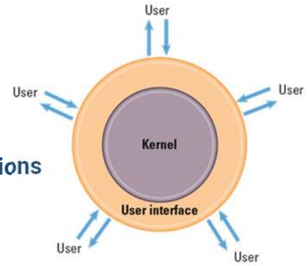
- Touch-screen
- Voice control





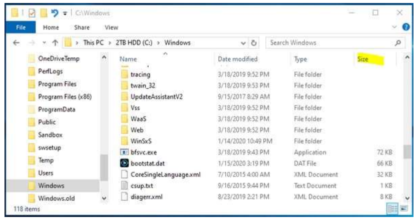
Operating System Components

- **User interface:**
  - **Communicates with users**
    - Text-based (Shell)
    - Graphical user interface (GUI)
- **Kernel:**
  - **Performs basic required functions**
    - File manager
    - Device drivers
    - Memory manager
    - Scheduler
    - Dispatcher

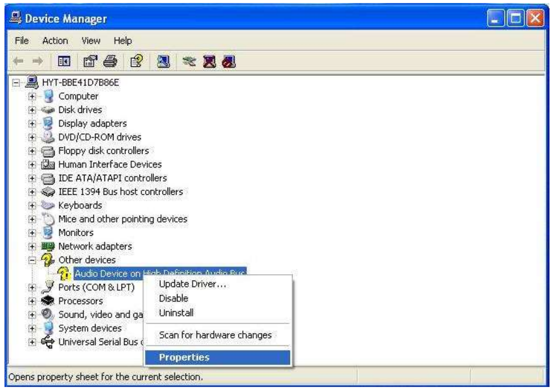


Kernel: File Manager

- Directory (or folder)
  - A user-created bundle of files and other directories (subdirectories)
- Directory path
  - A sequence of directories within directories

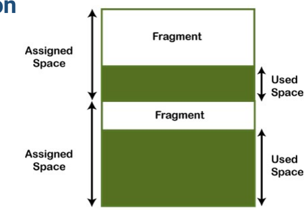


Kernel: Device Drivers



Kernel: Memory Manager

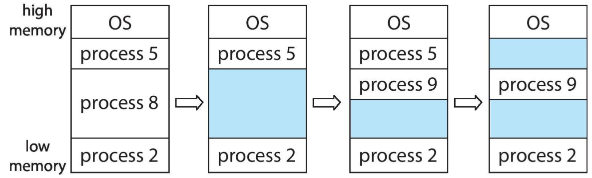
- Allocating space in the main memory
- **Contiguous allocation: fixed-partition allocation**
  - Each process loads into one partition of fixed-size
  - **Degree of multi-programming** is bounded by the number of partitions
  - Result in **internal fragmentation**
    - Memory that is internal to a partition but is not being used



33

Kernel: Memory Manager (cont.)

- Allocating space in the main memory
- **Contiguous allocation: variable-size partition**

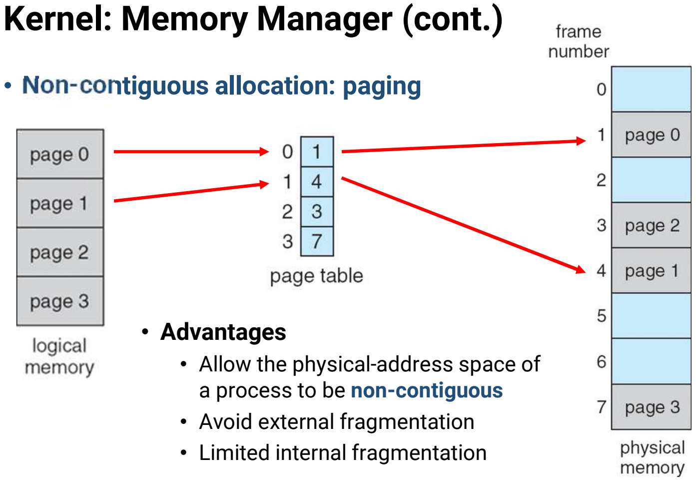


  - When a process arrives, it is allocated a hole **large enough** to accommodate it
  - Result in **external fragmentation**

34

Kernel: Memory Manager (cont.)

- **Non-contiguous allocation: paging**



- **Advantages**
  - Allow the physical-address space of a process to be **non-contiguous**
  - Avoid external fragmentation
  - Limited internal fragmentation

35

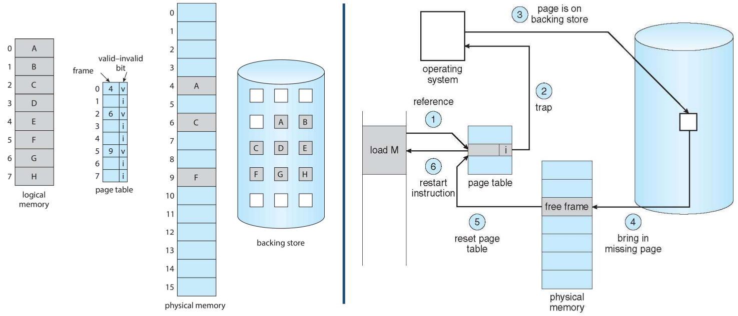
Kernel: Memory Manager (cont.)

- **Paging**
  - Divide **physical memory** into fixed-size blocks called **frames**
  - Divide **logical address** space into blocks of the **same size** called **pages**
  - To run a program of *n* pages, need to find *n* free frames and load the program
  - **Must keep track of free frames**
  - Set up a **page table** to translate logical to physical addresses

36

Kernel: Memory Manager

- **Virtual memory**
  - A process can be swapped out of memory to a **backing store**, and later brought back into memory for continuous execution



37

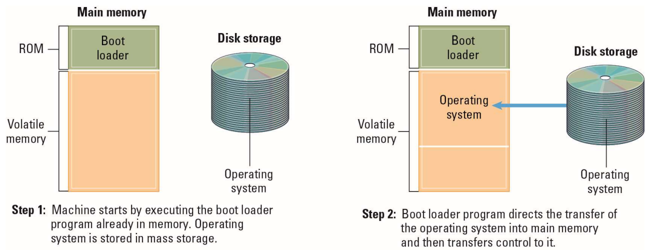
Kernel: Memory Manager

- **Virtual memory**
  - To run an **extremely large process**
    - Logical address space can be much larger than physical address space
  - To increase **CPU/resource utilization**
    - Higher degree of multi-tasking
    - Avoid putting rarely used data and codes in memory
  - To **launch** programs **faster**
    - Less I/O would be needed to load or swap

38

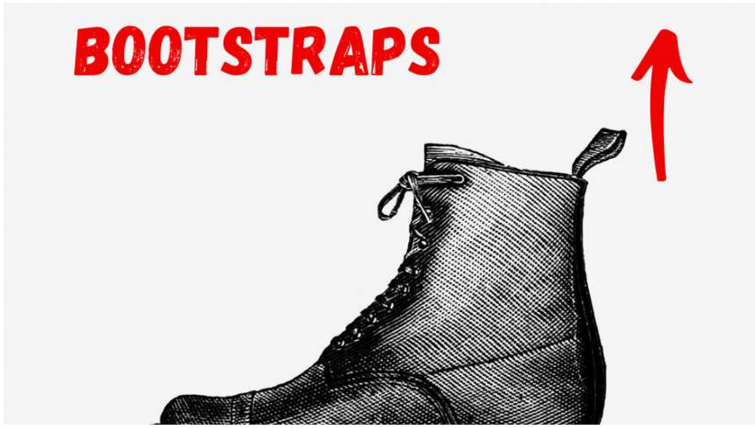
Bootstrapping / Booting

- **Boot loader:** program in ROM (read-only memory)
  - Run by the CPU when power is turned on
  - Transfers operating system from mass storage to main memory
  - Executes jump to the operating system



39

Bootstrapping / Booting (cont.)



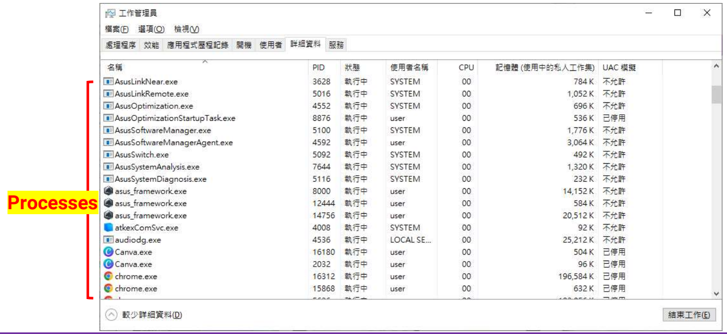
40

Outline

- What is an operating system
- The history of operating systems
- Operating system architecture
- Coordinating the machine's activities
- Handling competition among processes
- Security

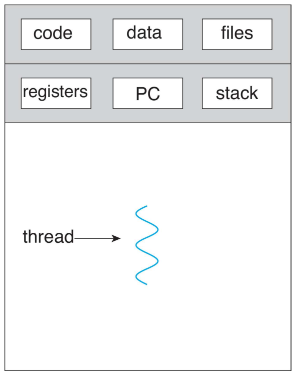
Coordinating the Machine's Activities

- An operating system coordinates the execution of application software, utility software, and units within the operating system itself



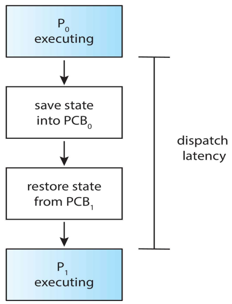
The Concept of a Process

- **Process**
  - The activity of executing a program
- **Process state**
  - Current status of the activity
    - Program counter
    - General purpose registers
    - Related portion of main memory
  - Managed by a process table (**Process Control Block, PCB**)
  - Save/load during a **context switch**

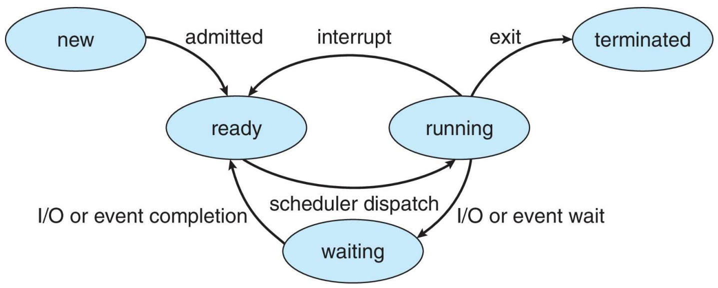


Process Administration

- **Scheduler**
  - Maintain the process table
    - Introduce new processes
    - Remove completed processes
    - Decide whether a process is ready or waiting
- **Dispatcher**
  - Really execute the program
    - Control the allocation of time slices to the processes
    - Switch processes (**context switch**)



Process State



Only one process is running on any processor at any instant

However, many processes may be ready or waiting (put into a queue)

Scheduling Criteria

- **CPU utilization**
  - Theoretically 0% ~ 100%
  - Real systems: 40% (light) ~ 90% (heavy)
- **Throughput** — system view
  - Number of completed processes per time unit
- **Turnaround time**
  - Submission ~ completion
- **Waiting time**
  - Total waiting time in the ready queue
- **Response time**
  - Submission ~ the first response is produced

single job view

Scheduling Criteria (cont.)

- **Max** CPU utilization
- **Max** Throughput
- **Min** Turnaround time
- **Min** Waiting time
- **Min** Response time

Scheduling Algorithms

- First-Come, First-Served (FCFS) scheduling
- Shortest-Job-First (SJF) scheduling
- Priority scheduling
- Round-Robin scheduling
- Multi-level queue scheduling
- Multi-level feedback queue scheduling

Starvation

- Process cannot get the resources needed for a long time because the resources are being allocated to other processes
- **Aging**
  - Add an aging factor to the priority of each request

Outline

- What is an operating system
- The history of operating systems
- Operating system architecture
- Coordinating the machine's activities
- Handling competition among processes
- Security

Data Consistency

- **Concurrent access** to **shared data** may result in **data inconsistency**
- Maintaining data consistency requires a mechanism to ensure the **orderly execution** of cooperating processes

Example: Consumer & Producer Problem

- **Producer** process produces information that is consumed by a **Consumer** process, both operating on a fixed-size buffer

```
/* Producer */
while (true) {
    // produce an item in next produced.
    while (counter == BUFFER_SIZE);
        // do nothing.
    buffer[in] = next_produced;
    in = (in + 1) % BUFFER_SIZE;
    counter++;
}
```

```
/* Consumer */
while (true) {
    while (counter == 0);
        // do nothing.
    next_consumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
    counter--;
    // consume the item in next consumed.
}
```

Concurrent Operations on Counter

- The statement "counter++" may be implemented in machine language as
  - *move R1, counter*
  - *add R1, 1*
  - *move counter, R1*
- The statement "counter--" may be implemented as
  - *move R2, counter*
  - *sub R2, 1*
  - *move counter, R2*

Instruction Interleaving

- Assume the counter is initially 5. One interleaving of statement is

producer: move R1, counter	→ R1 = 5
producer: add R1, 1	→ R1 = 6
context switch	
consumer: move R2, counter	→ R2 = 5
consumer: sub R2, 1	→ R2 = 4
context switch	
producer: move counter, R1	→ counter = 6
context switch	
consumer: move counter, R2	→ counter = 4

Handling Competition among Processes

- **Critical Region**
  - A **protocol** for processes to cooperate
  - A group of instructions that should be executed by only one process at a time
- **Mutual exclusion**
  - Requirement that **only one** process at a time be allowed to execute a critical region

```
do {
    entry section      → get entry permission
    critical section   → modified shared data
    exit section       → release entry permission
    remainder section
} while (1);
```


Semaphore

- A tool to generalize the synchronization problem
  - Can be achieved by hardware or software solutions
- Hardware support: **atomic instructions** (uninterruptible)

```
bool TestAndSet (bool &lock) {
    bool value = lock;
    lock = true;
    return value;
}
```

execute atomically: **return the value** of "lock" and **set** "lock" to true

```
shared data: bool lock; // initially lock = false
```

```
// P0                              // P1
do {                               do {
    while (TestAndSet (lock));         while (TestAndSet (lock));
    critical section                   critical section
    lock = false;                      lock = false;
    remainder section                  remainder section
} while (1);                       } while (1);
```

Deadlock

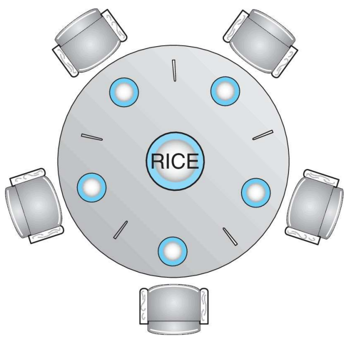
- Processes block each other from continuing because each is waiting for a resource that is allocated to another
- Example
  - 2 processes
    - P_1 holds resource B and waits for resource A
    - P_2 holds resource A and waits for resource B

Deadlock (cont.)

- Conditions required for deadlock
  - Competition for non-sharable resources (**mutual exclusion**)
    - Only one process at a time can use a resource
  - Resources requested on a partial basis (**hold and wait**)
    - A process holding some resources and is waiting for another resource
  - An allocated resource can not be forcibly retrieved (**no preemption**)
    - A resource can be only released by a process **voluntarily**
  - **Circular wait**
    - There exists a set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes such that $P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow \ldots \rightarrow P_n \rightarrow P_0$

Deadlock (cont.)

- Dining-philosophers problem



Handling Deadlocks

- Ensure the system will **never** enter a deadlock state
  - **Deadlock prevention**: ensure that at least one of the **four necessary conditions** cannot hold
  - **Deadlock avoidance**: **dynamically** examines the resource-allocation state before allocation
- Allow to **enter a deadlock state** and then **recover**
  - **Deadlock detection**
  - **Deadlock recovery**
- **Ignore the problem** and pretend that deadlocks never occur in the system
  - **Used by most operating systems, including UNIX**

Outline

- What is an operating system
- The history of operating systems
- Operating system architecture
- Coordinating the machine's activities
- Handling competition among processes
- Security

61

Security

- **Goals**
  - Prevent error and misuse
  - Resources are only allowed to be accessed by authorized processes
- **Attacks from outside**
  - Problems
    - Insecure passwords and **bad habits**
    - Sniffing software
    - Virus, worms, Trojan horses
  - Counter measures
    - Auditing software (record and analyze activities)
    - Antivirus software

62

Security (cont.)

- **Attacks from within**
  - Problem
    - Process that gains access to memory outside its designated area
  - Counter measures
    - Control process activities via **privilege levels** and **privileged instructions**

63

Any Questions?

64