北京邮电大学软件学院
School Of software Engineering Of BUPT

Operating Systems

Lecture 2 Processes

Jinpengchen
Email: jpchen@bupt.edu.cn

Catalog Description

- Multiprogramming techniques & Bernstein's conditions
- Process Concept
- Process Scheduling
- Operations on Processes
- Interprocess Communication

Multiprogramming techniques & Bernstein's conditions

- Difficulties of multiprogramming techniques
  - From Simple Batch system → Multiprogramming system
    - ✓ Memory must be shared by multiple programs
    - ✓ CPU must be multiplexing(复用) by multiple programs
    - ✓ 4 basic components:
      - Process management
      - Memory management
      - I/O system management
      - file management

Multiprogramming techniques & Bernstein's conditions

- Difficulties of multiprogramming techniques
  - 与单道相比，在多道系统中，进程之间的运行随着调度的发生而具有无序性，那么
    - ✓ How to ensure correct concurrent?
  - Related theory:
    - ✓ Conditions of the concurrent execution of program
    - ✓ Theoretical model: Precedence graph (前趋图)
    - ✓ Analysis on the serial execution of programs based on precedence graph
    - ✓ Analysis on the concurrent execution of programs based on precedence graph

Multiprogramming techniques & Bernstein's conditions

- Difficulties of multiprogramming techniques
  - Precedence Graph （前趋图）
    - ✓ Goal：准确的描述语句、程序段、进程之间的执行次序
  - Definition: Precedence graph（前趋图）is a Directed Acyclic Graph（有向无环图，DAG）.
    - ✓ Node(结点)：一个执行单元（如一条语句、一个程序段或进程）
    - ✓ Edge(边，directed edge(有向边))：The precedence relation（前趋关系）“→”，
    - ✓ → ={(Pi，Pj)｜Pi必须在Pj开始执行前执行完}

Multiprogramming techniques & Bernstein's conditions

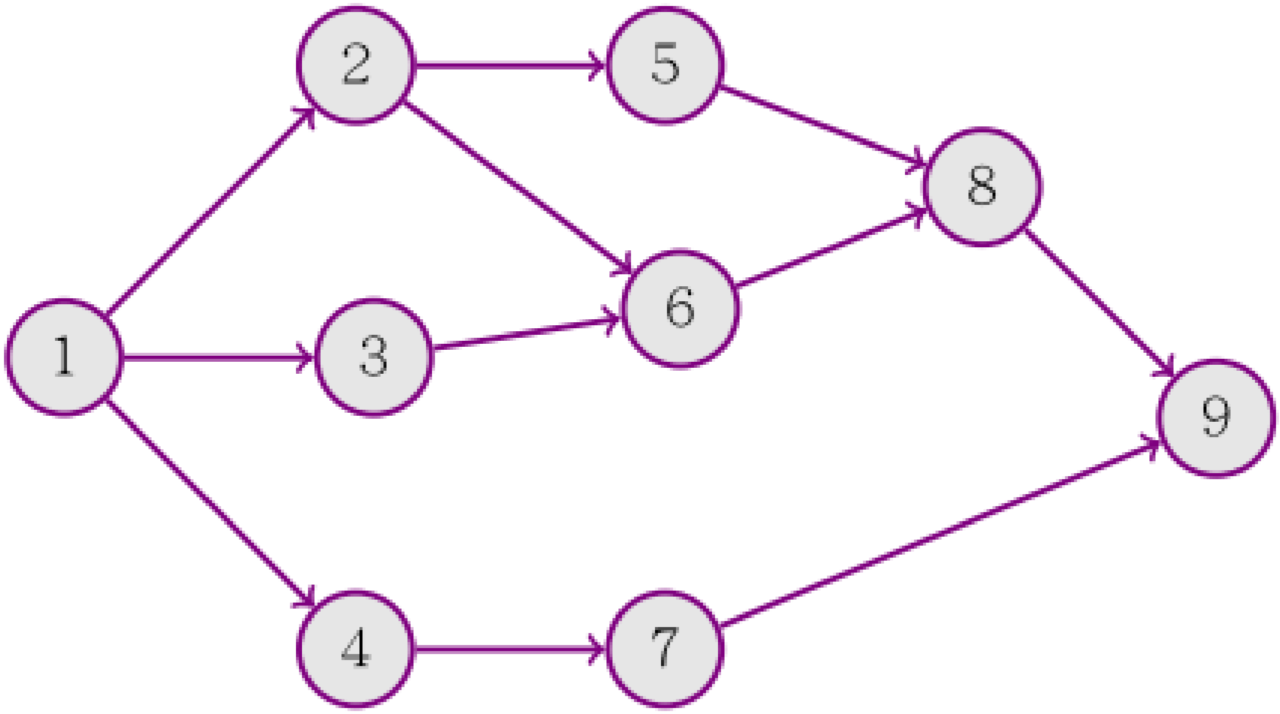
- If （Pi，Pj）∈→， then Pi → Pj

  Here,

  Pi is called the predecessor(前趋) of Pj, and Pj is the subsequent(后继) of Pi
- 没有前趋的结点称为初始结点（initial node）
- 没有后继的结点称为终止结点（final node）
- 结点上使用一个权值（weight）表示该结点所含的程序量或结点的执行时间

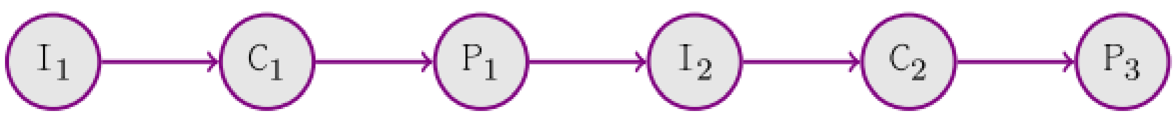
Multiprogramming techniques & Bernstein's conditions

✓ Example



Multiprogramming techniques & Bernstein's conditions

- Serial execution of programs(程序的顺序执行)
  - 一个较大的程序通常包含若干个程序段。程序在执行时，必须按照某种先后顺序逐个执行，仅当前一个程序段执行完，后一个程序段才能执行。
  - 例如



其中，

I代表用户程序和数据的输入；

C代表计算；

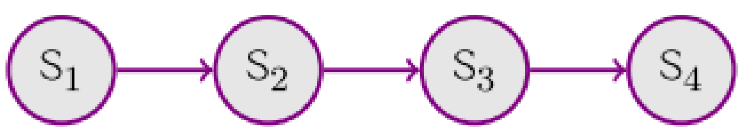
P代表输出结果。

Multiprogramming techniques & Bernstein's conditions

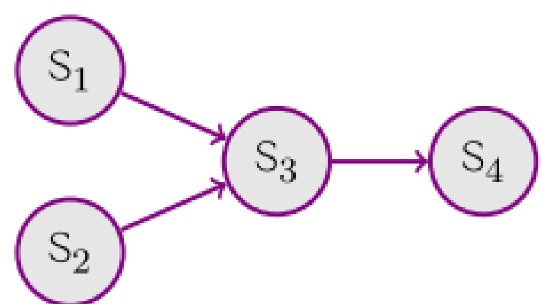
- 在一个程序段中，多条语句也存在执行顺序的问题。

  在下面的例子中，S1和S2必须在S3执行前执行完。类似的，S4必须在S3执行完才能执行。

  S1：a＝x＋3
  S2：b＝y＋4
  S3：c＝a＋b
  S4：d＝a＋c



按指令地址顺序执行



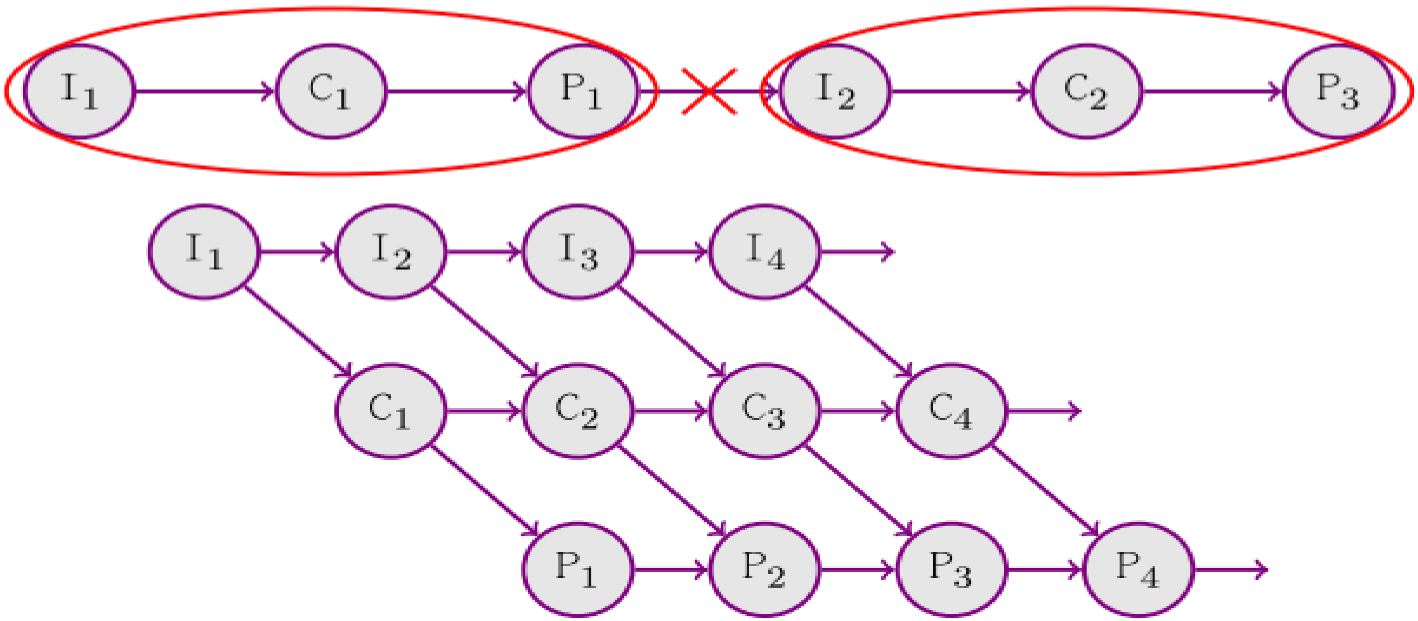
按照语句依赖关系

Multiprogramming techniques & Bernstein's conditions

- 程序顺序执行时的特征
  - 顺序性
    - ✓ 严格按照程序规定的顺序执行
  - 封闭性
    - ✓ 程序是在封闭的环境下运行的。独占全机资源。一旦开始运行，结果不受外界因素的影响。
  - 可再现性
    - ✓ 只要程序执行时的环境和初始条件相同，都将获得相同的结果。

Multiprogramming techniques & Bernstein's conditions

- Concurrent execution of programs (程序的并发执行)
  - Pi与Ii+1 之间不存在内在的前趋关系



程序并发执行时的前趋图

Multiprogramming techniques & Bernstein's conditions

- 程序并发执行时的特征
  - 间断性
    - ✓ 并发程序“执行——暂停执行——执行”
  - 失去封闭性
    - ✓ 由于资源共享，程序之间可能出现相互影响的现象
  - 不可再现性
    - ✓ 原因同上
    - ✓ 举例：变量N的共享，设某时刻N＝n，则若执行顺序为：

      1.N:＝N＋1；print(N)； N:=0； N的值依次为n＋1；n＋1；0

      2.print(N)； N:=0； N:＝N＋1；   N的值依次为n；0；1

      3.print(N)； N:＝N＋1；N:=0； N的值依次为n；n＋1；0

Multiprogramming techniques & Bernstein's conditions

- Bernstein's conditions
  - 在上述3个特性中，必须防止“不可再现性”
  - 为使并发程序的执行保持“可再现性”，引入并发执行的条件
    - ✓ 思路：分析程序或语句的输入信息和输出信息，考察它们的相关性
    - ✓ Definitions, notation and terminology:
      读集R (pi)，表示程序pi 在执行时需要参考的所有变量的集合
      写集W (pi)，表示程序pi 在执行期间要改变的所有变量的集合
    - ✓ 1966, Bernstein: if programs p1 and p2 meet the following conditions, they can be executed concurrently, and have reproducibility (可再现性)

Multiprogramming techniques & Bernstein's conditions

- Bernstein's conditions

If process pi writes to a memory cell Mi, then no process pj can read the cell Mi.

If process pi read from a memory cell Mi, then no process pj can write to the cell Mi.

If process pi writes to a memory cell Mi, then no process pj can write to the cell Mi.

$$R(p1) \cap W(p2) \cup R(p2) \cap W(p1) \cup W(p1) \cap W(p2) = \varnothing$$

Catalog Description

- Multiprogramming techniques & Bernstein's conditions
- Process Concept
- Process Scheduling
- Operations on Processes
- Interprocess Communication

Why Processes?

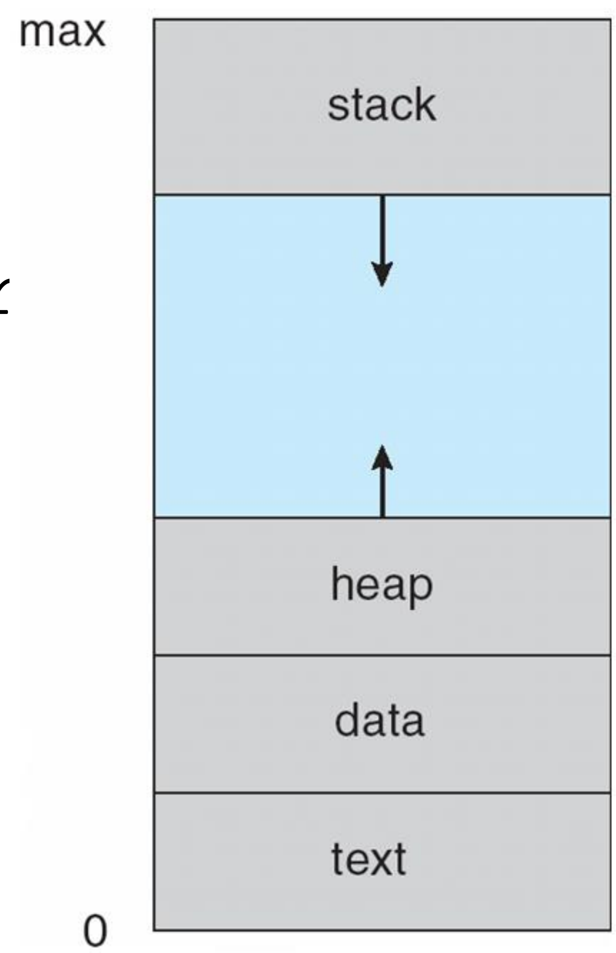
- 为了提高计算机系统中各种资源的利用率，现代操作系统广泛采用多道程序技术（multiprogramming），使多个程序在系统中存在并运行。
- 在多道程序系统中，各个程序之间是并发执行的，共享系统资源。CPU需要在各个运行的程序之间来回切换，这样的话，要想描述这些多道的并发活动过程就变得很困难。

Process Concept

- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – user programs or tasks
  - Textbook uses the terms job and process almost interchangeably
- we call all of them process
  - a program in execution;
  - process execution must progress in sequential fashion

Process Concept

- A process includes:
  - text section ⇐ program code
  - program counter + other
  - Registers ⇐ current activity
  - Stack ⇐ temporary data
  - data section ⇐ global variables
  - Heap



Process Concept

- COMPARE: Program vs. Process?
  (Program ! = Process)
  - **A program is a passive entity (C statements or commands 静态的)**
  - **Process is a active entity (program + running context活动的)**

Main(){
…
}
A(){
…}

Program

Main(){
…
}
A(){
…}

Process

Heap

Stack

A
Main

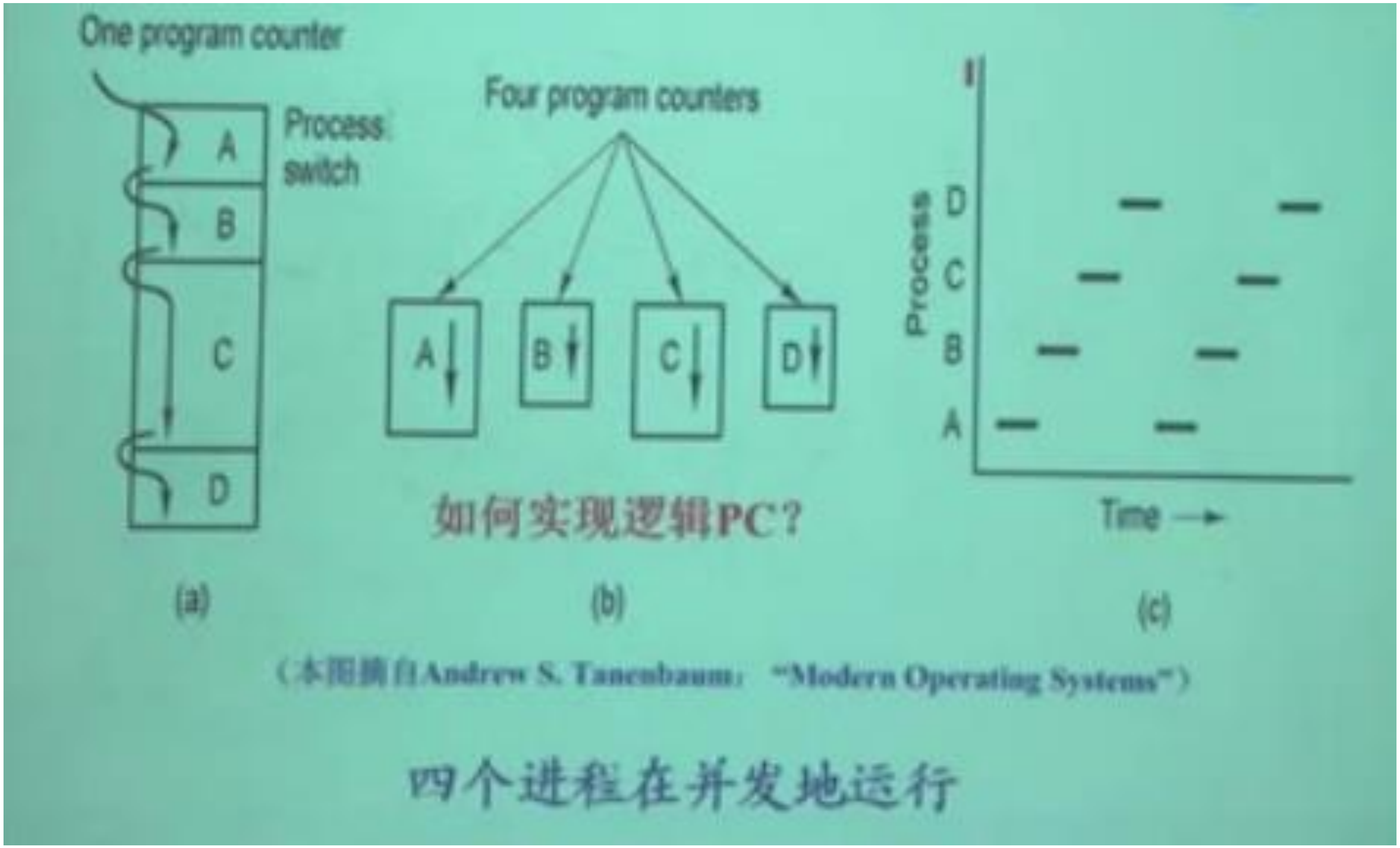
Process Concept

- 进程的五大特征
  - 动态性：最基本的特性
    - ✓ “它由创建而产生，由调度而执行，因得不到资源而暂停执行，以及由撤销而消亡”
    - ✓ 具有生命期
  - 并发性
    - ✓ 多道
    - ✓ 既是进程也是OS的重要特征
  - 独立性
    - ✓ 进程是一个能独立运行的基本单位，也是系统中独立获得资源和独立调度的基本单位。

Process Concept

- 进程的五大特征
  - 异步性
    - ✓ 进程按各自独立的、不可预知的速度向前推进。
    - ✓ 导致“不可再现性”
    - ✓ OS必须采取某种措施来保证各程序之间能协调运行。
  - 结构特征
    - ✓ 从结构上看，进程实体是由程序段、数据段及进程控制块三部分组成
    - ✓ 进程映像 = 程序段 + 数据段 + 进程控制块

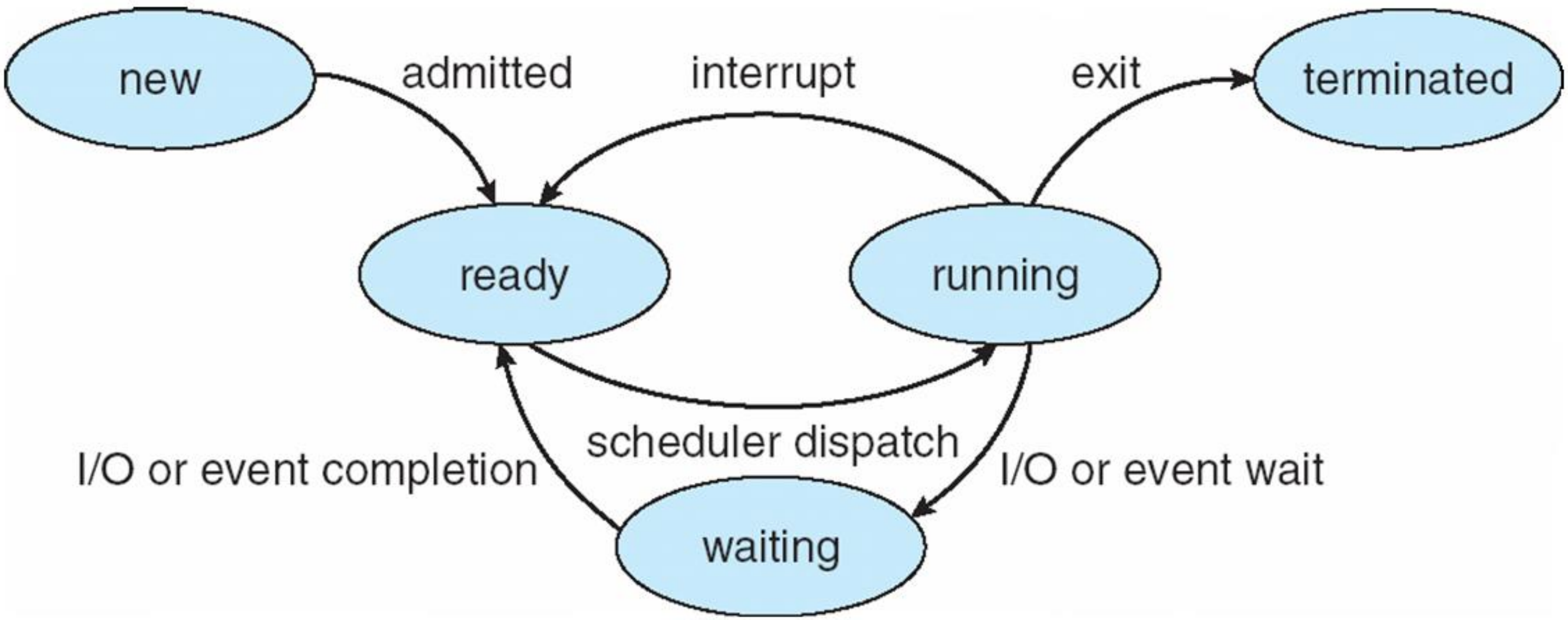
Process Concept



Process Concept

- Process State
  - As a process executes, it changes state
    - new: The process is being created
    - running: Instructions are being executed
    - waiting: The process is waiting for some event to occur
    - ready: The process is waiting to be assigned to a processor
    - terminated: The process has finished execution

Process Concept



1. 进程正常运行（未阻塞）时处于什么状态？
2. 此PPT处于什么状态？
3. 是否有其他的状态转换？

Process Concept

- Process Control Block
  - Program = data structure + algorithm
  - Each process is represented in the OS by a PCB, also called Task Control Block, TCB
    - ✓ 是操作系统中的一种关键数据结构
    - ✓ 由操作系统进程管理模块维护
    - ✓ 常驻内存
  - 操作系统根据PCB来控制和管理并发执行的进程

Process Concept

- Process Control Block
  - Process state
  - Program counter
  - CPU registers
  - CPU scheduling information
  - Memory-management information
  - Accounting information
  - I/O status information

Process Concept

- Process Control Block

Process Management

Memory Management
File Management

Process Concept

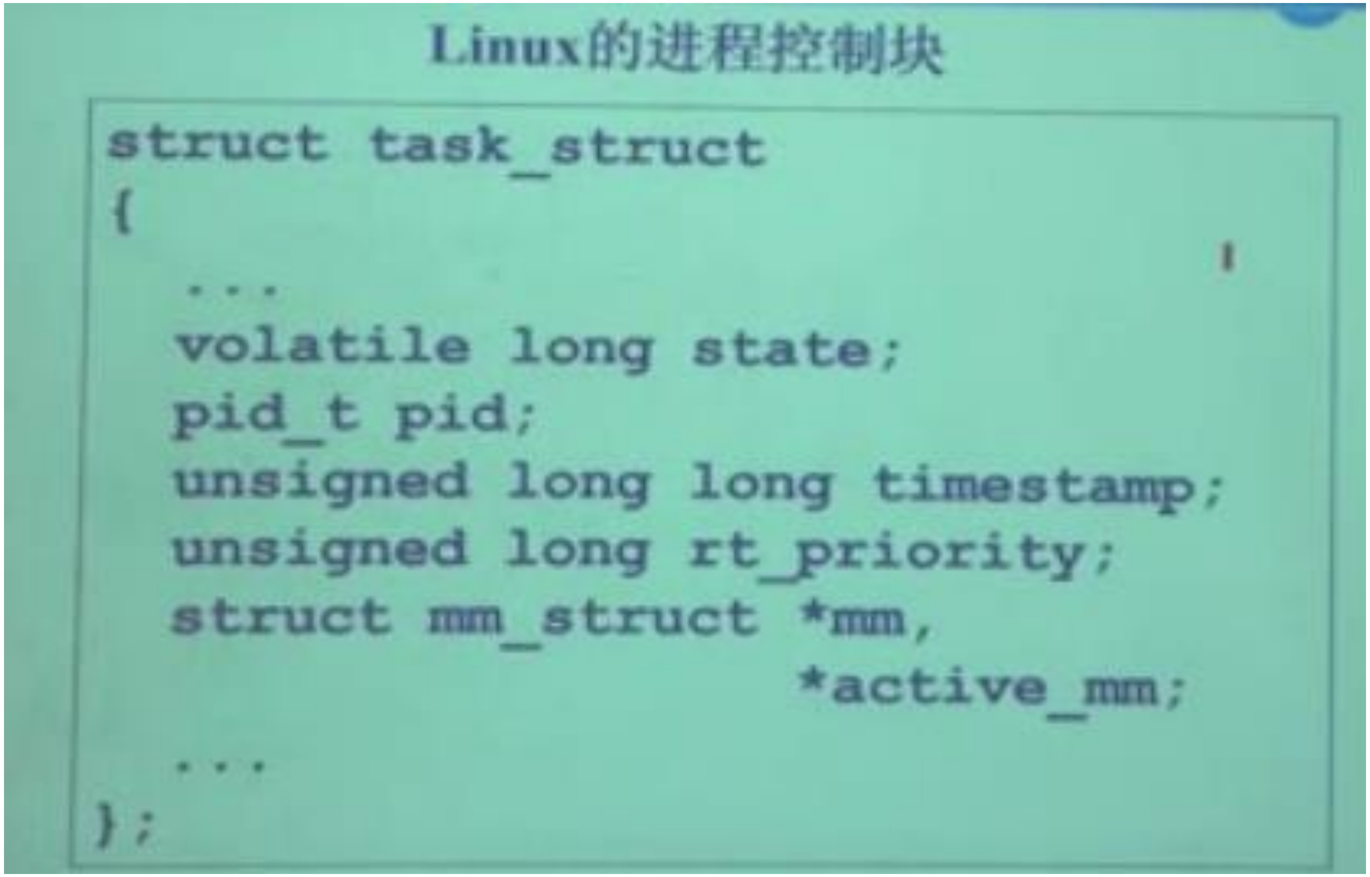
- Process Control Block

Process management	Memory management	File management
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

逻辑寄存器

Process Concept

- Process Control Block

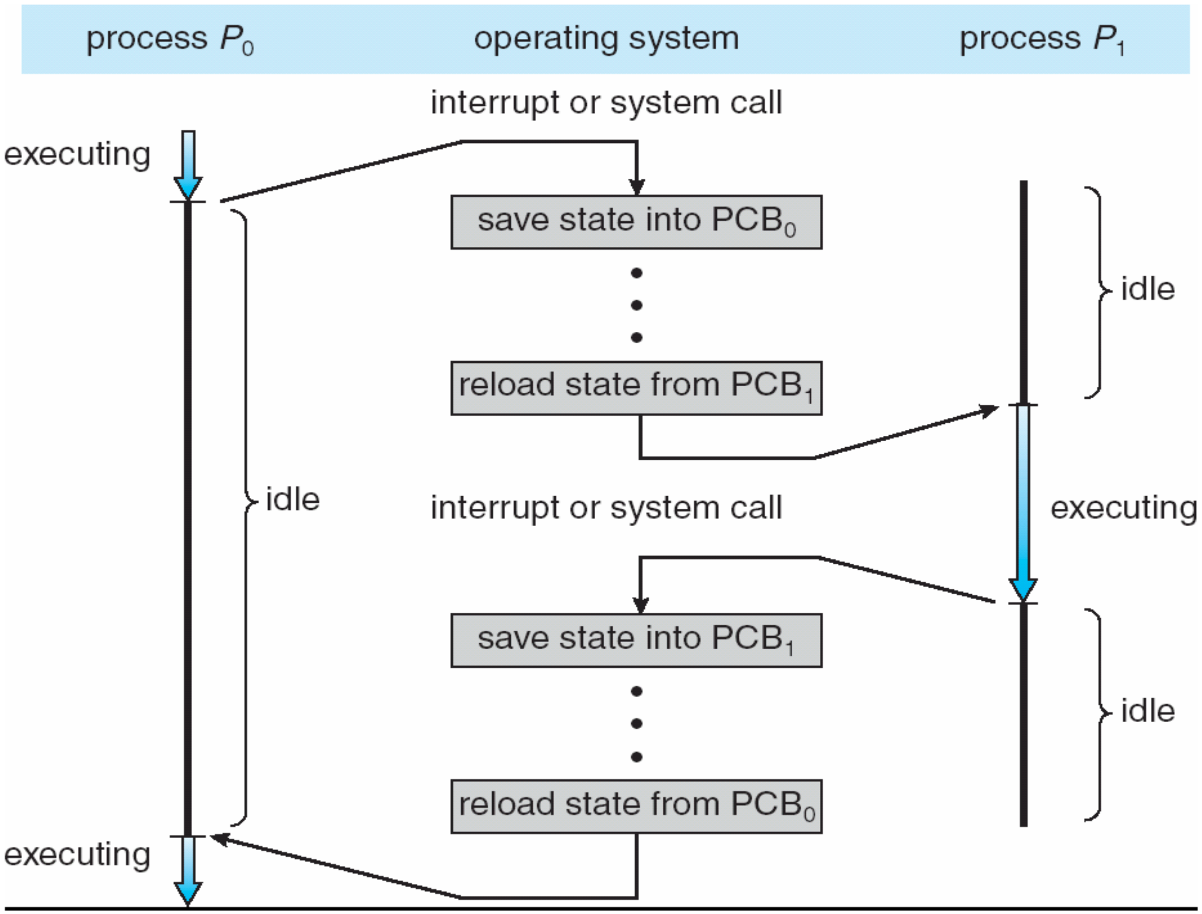


Linux的进程控制块

```
struct task_struct
{
    ...
    volatile long state;
    pid_t pid;
    unsigned long long timestamp;
    unsigned long rt_priority;
    struct mm_struct *mm,
                     *active_mm;
    ...
};
```

Process Concept

- CPU Switch From Process to Process



Catalog Description

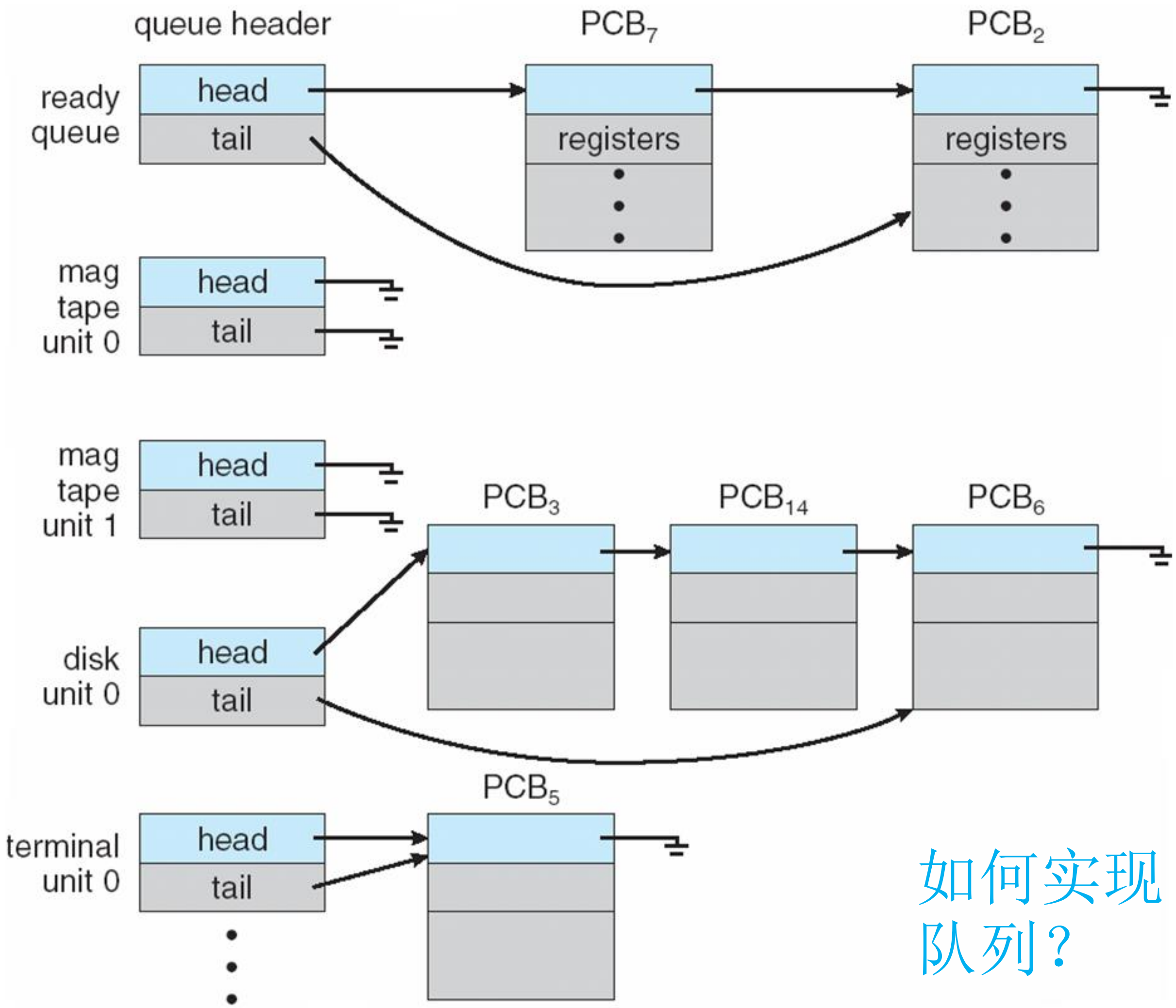
- Multiprogramming techniques & Bernstein's conditions
- Process Concept
- Process Scheduling
- Operations on Processes
- Interprocess Communication

Process Scheduling

- Process Scheduler selects as available process to execute.
- Process Scheduling Queues
  - Job queue – set of all processes in the system
  - Ready queue – set of all processes residing in main memory, ready and waiting to execute
  - Device queues – set of processes waiting for an I/O device
- Processes migrate among the various queues

Process Scheduling

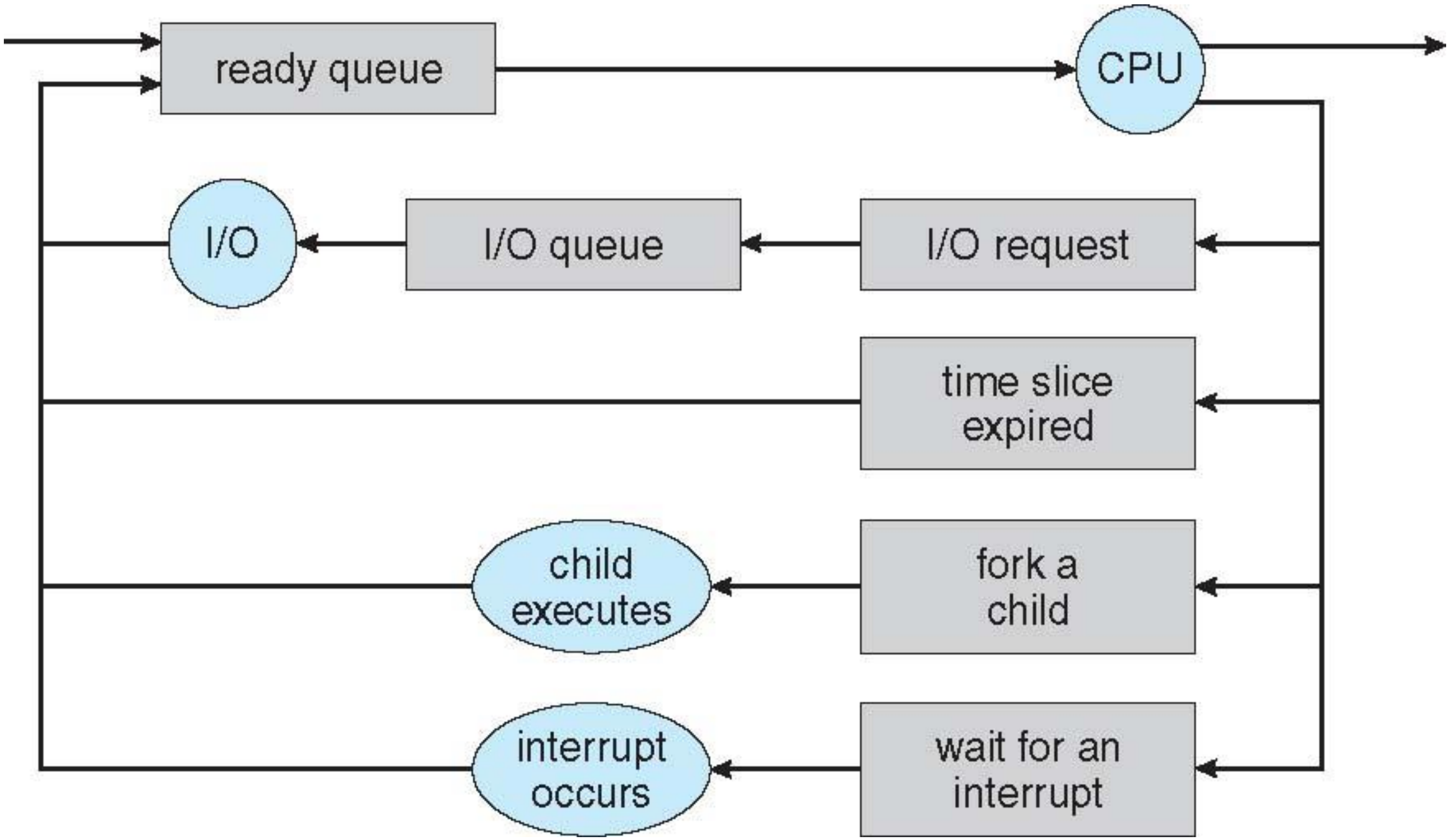
- Ready Queue And Various I/O Device Queues



如何实现队列？

Process Scheduling

- Representation of Process Scheduling



Process Scheduling

- Schedulers
  - Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue
  - Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU

Process Scheduling

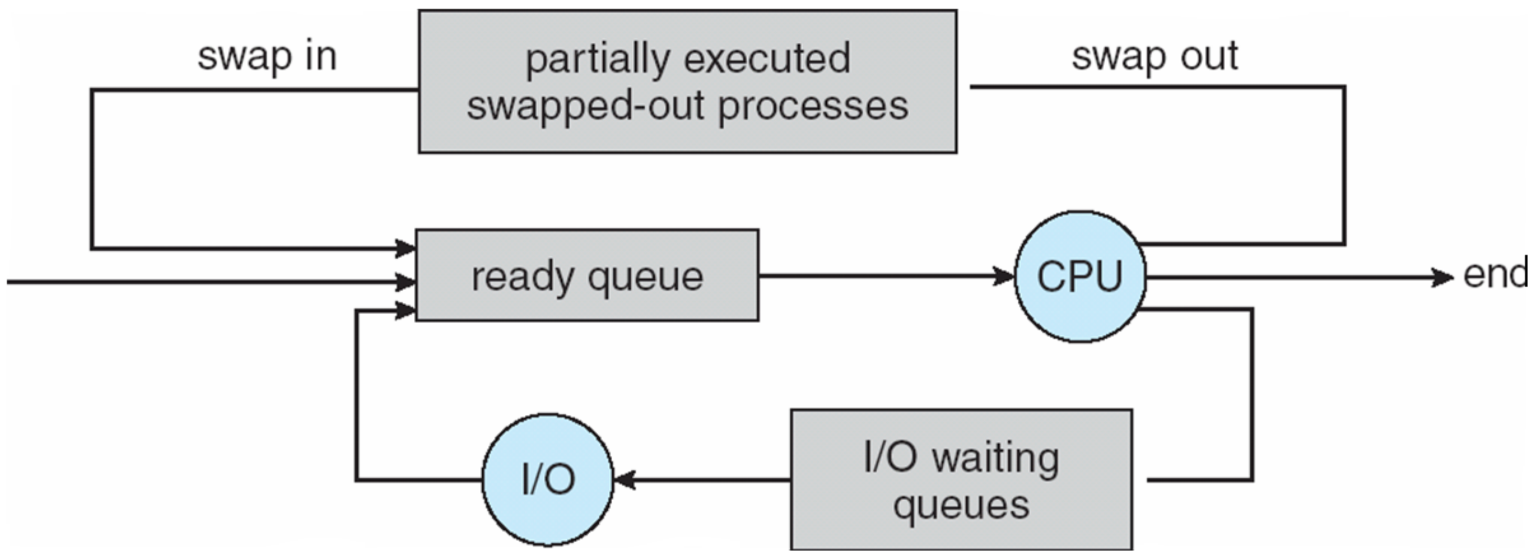
- Schedulers
  - The prilmary distinction between long-term & short-term schedulers lies in frequency of execution
    - ✓ Short-term scheduler is invoked very frequently(milliseconds)(must be fast)
    - ✓ Long-term scheduler is invoked very infrequently(seconds, minutes)(may be slow)
  - The long-term scheduler controls the degree of multiprogramming
    - ✓ the number of processes in memory.

Process Scheduling

- Schedulers
  - Processes can be described as either:
    - ✓ I/O-bound process – spends more time doing I/O than computations, many short CPU bursts
    - ✓ CPU-bound process – spends more time doing computations; few very long CPU bursts
  - IMPORTANT for long-term scheduler:
    - ✓ A good process mix of I/O-bound and CPU-bound processes.

Process Scheduling

- Addition of Medium Term Scheduling
  - ✓ can reduce the degree of multiprogramming
  - ✓ the scheme is called swapping (交换): swap in VS. swap out



Process Scheduling

- Context Switch
  - Context(上下文)
    - ✓ when an interrupt occurs; When scheduling occurs
  - the context is represented in the PCB of the process
    - ✓ CPU registers
    - ✓ process state
    - ✓ memory-management info
    - ✓ ...

Process Scheduling

- Context Switch
  - When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a context switch
  - Context-switch time is overhead; the system does not do useful work while switching
  - Time dependent on hardware support (typical: nμs)
    - ✓ CPU & memory speed
    - ✓ N of registers
    - ✓ the existence special instructions

Catalog Description

- Multiprogramming techniques & Bernstein's conditions
- Process Concept
- Process Scheduling
- Operations on Processes
- Interprocess Communication

Operations on Processes

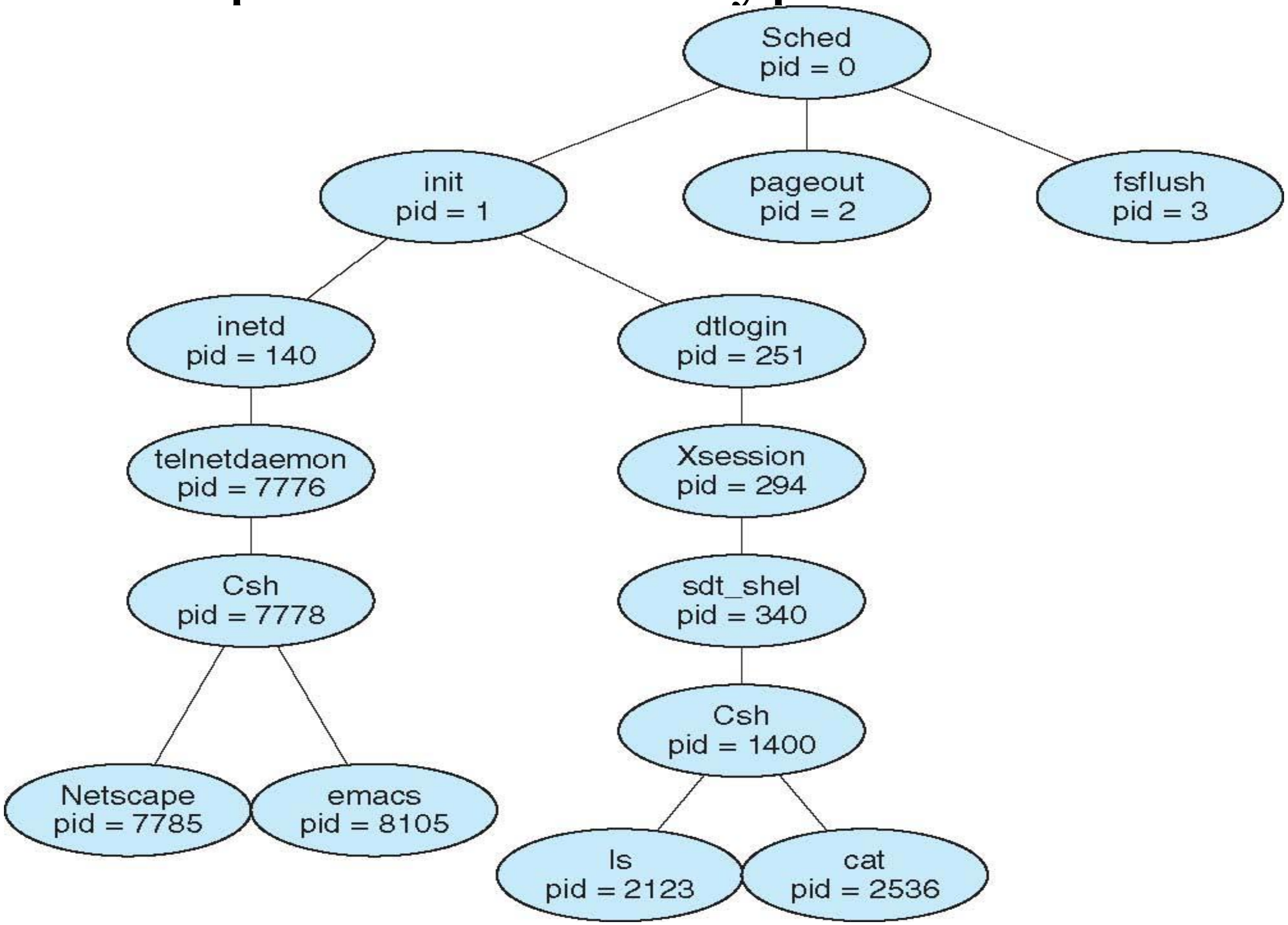
- Process Creation
  - The processes in most systems can execute concurrently, and they may be created and deleted dynamically.
  - The OS must provide a mechanism for
    - ✓ process creation
    - ✓ process termination

Operations on Processes

- Process Creation
  - Parent process create children processes, which in turn create other processes, forming a tree of processes
  - Generally, process identified and managed via a process identifier (pid)
    - ✓ typically an integer number

Operations on Processes

- A tree of processes on a typical Solaris

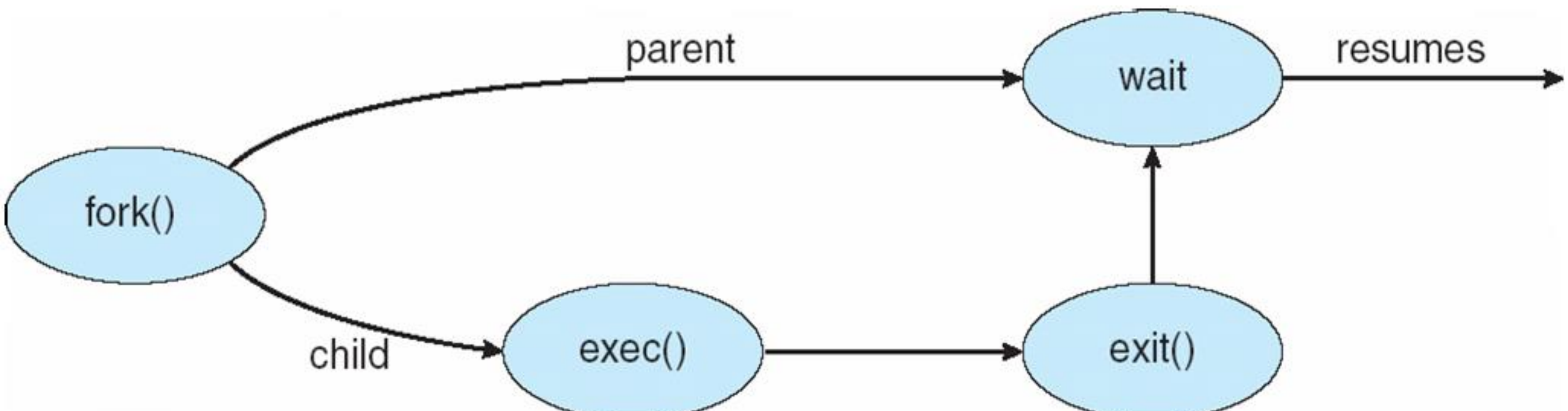


Operations on Processes

- Process Creation
  - Resource sharing
    - ✓ In general, a process will need certain resources (CPU time, memory, files, I/O devices) to accomplish its task.
    - ✓ When a process creates a subprocesses
      - –Parent and children share all resources
      - –Children share subset of parent's resources
      - –Parent and child share no resources
  - Execution
    - ✓ Parent and children execute concurrently
    - ✓ Parent waits until children terminate

Operations on Processes

- Process Creation
  - Address space
    - ✓ Child duplicates parents'
    - ✓ Child has a program loaded into it
  - UNIX examples
    - ✓ fork system call creates new process
    - ✓ exec system call used after a fork to replace the process' memory space with a new program



Operations on Processes

- Process Creation
  - C Program Forking Separate Process

```
int main()
{
        pid_t  pid;
         /* fork another process */
         pid = fork();//创建进程，返回一个ID
         if (pid < 0) { /* error occurred */
                          fprintf(stderr, "Fork Failed");
                          exit(-1);
         }
         else if (pid == 0) { /* child process */
                          execlp("/bin/ls", "ls", NULL);
         }
         else { /* parent process */
                          /* parent will wait for the child to complete */
                          wait (NULL);
                          printf ("Child Complete");
                          exit(0);
         }
}
```

Operations on Processes

- Process Termination
  - [Self] Process executes last statement and asks the OS to delete it by using the exit() system call.
    - ✓ Output data (a status value, typically an integer) from child to parent (via wait())
    - ✓ Process' resources are deallocated by the OS
  - [Other] Termination can be caused by another process
    - ✓ Example: TerminateProcess() in Win32
  - [User] Users could kill some jobs.

Operations on Processes

- Process Termination
  - [Parent] Parent may terminate execution of children processes (abort)
    - ✓ Child has exceeded allocated resources
    - ✓ Task assigned to child is no longer required
    - ✓ If parent is exiting（被撤销）

      Some operating system do not allow child to continue if its parent terminates

      All children terminated - cascading termination

Catalog Description

- Multiprogramming techniques & Bernstein's conditions
- Process Concept
- Process Scheduling
- Operations on Processes
- Interprocess Communication

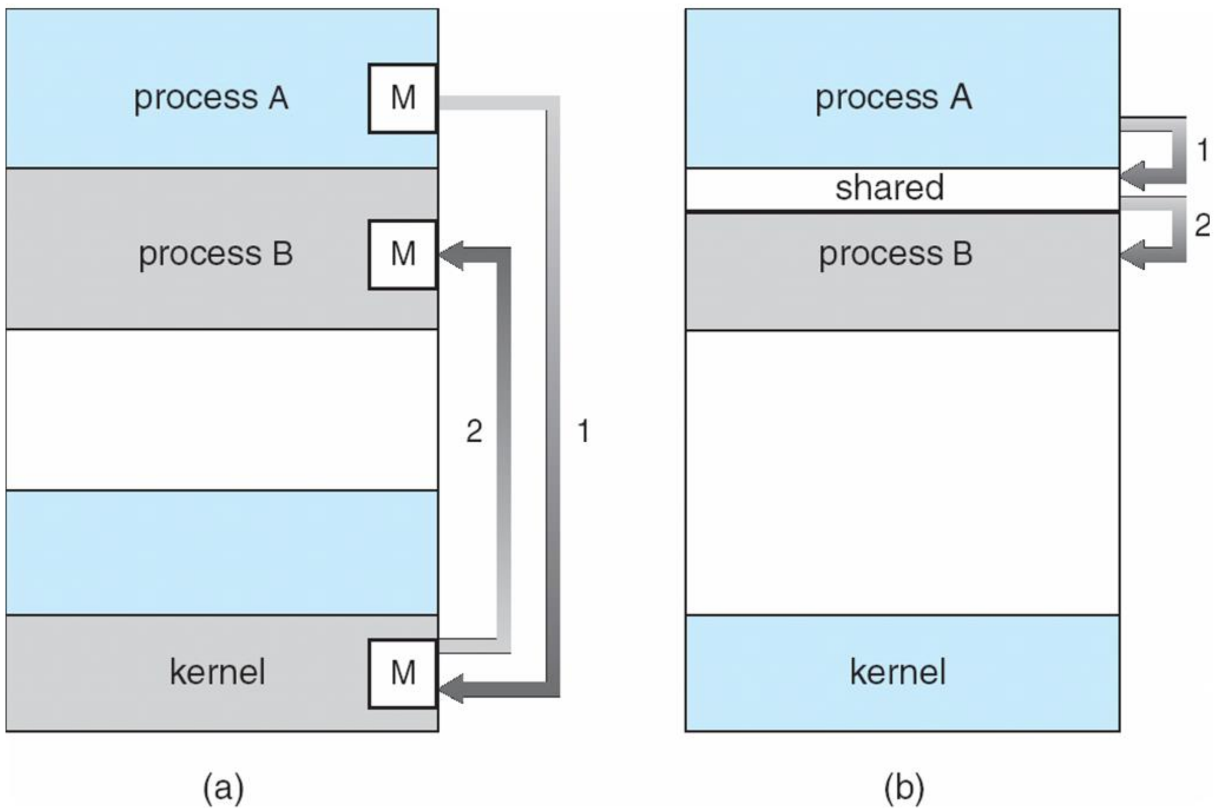
Interprocess Communication

- Interprocess Communication (进程间通信，IPC)
  - Processes executing concurrently in the OS may be either independent processes or cooperating processes
    - ✓ Independent process cannot affect or be affected by the execution of other processes
    - ✓ Cooperating process can affect or be affected by the execution of other processes
  - Advantages of allowing process cooperation
    - ✓ Information sharing: a shared file VS. several users
    - ✓ Computation speed-up: 1 task VS. several subtasks in parallel with multiple processing elements (such as CPUs or I/O channels)
    - ✓ Modularity
    - ✓ Convenience: 1 user VS. several tasks
  - Cooperating processes require an IPC mechanism that will allow them to exchange data and information.

Interprocess Communication

- **Two fundamental models of IPC:**
  - Message-passing (消息传递) model
    - ✓ useful for exchange smaller amount of data, because no conflicts need be avoided.
    - ✓ easier to implement
    - ✓ exchange information via system calls such as send(), receive()
  - Shared-memory (共享内存) model
    - ✓ faster at memory speed via memory accesses.
    - ✓ system calls only used to establish shared memory regions

Interprocess Communication

- Two fundamental models of IPC:
  - 

Interprocess Communication

- Shared-Memory systems
  - Normally, the OS tries to prevent one process from accessing another process's memory.
  - Shared memory requires that two or more processes agree to remove this restriction.
    - ✓ exchange information by R/W data in the shared areas.
    - ✓ The form of data and the location are determined by these processes and not under the OS’s control.
    - ✓ The processes are responsible for ensuring that they are not writing to the same location simultaneously.

Interprocess Communication

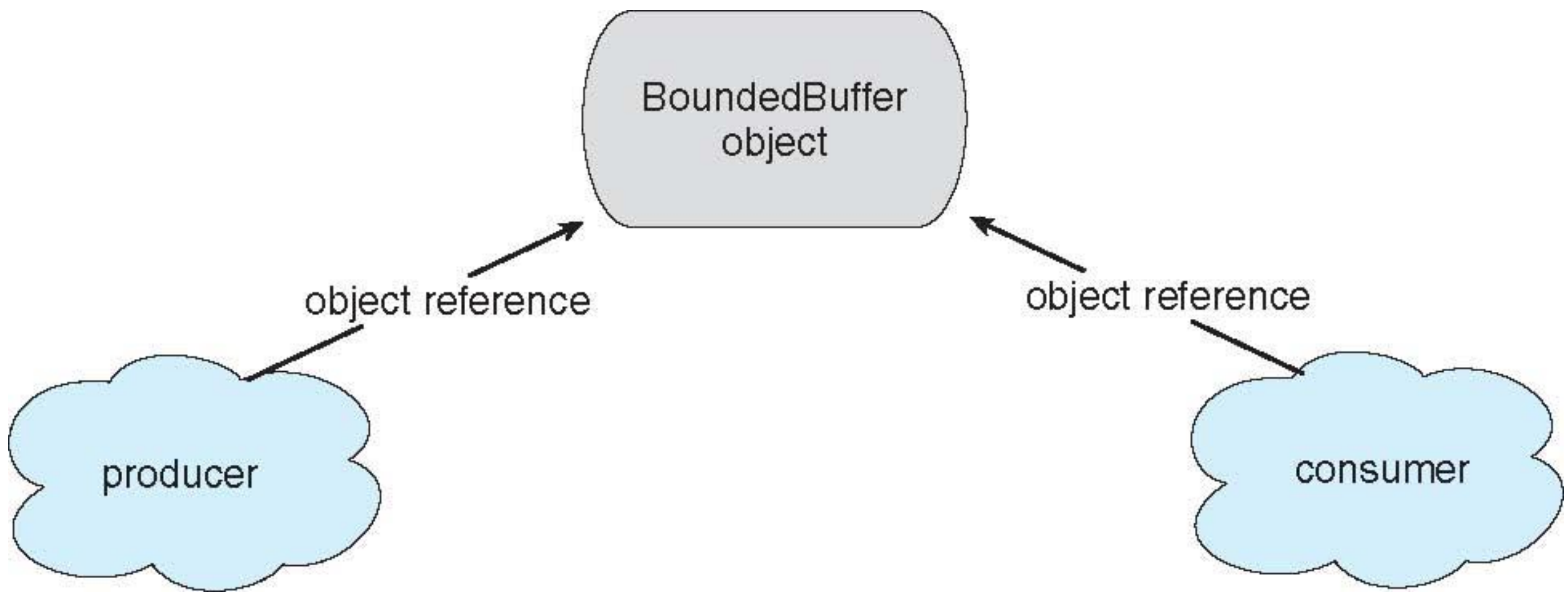
- Shared-Memory systems
  - Producer-Consumer Problem (生产者-消费者问题，PC问题):
    - ✓ Paradigm for cooperating processes, producer process produces information that is consumed by a consumer process

      - unbounded-buffer places no practical limit on the size of the buffer

      - bounded-buffer assumes that there is a fixed buffer size

Interprocess Communication

- Shared-Memory systems
  - Producer-Consumer Problem (生产者-消费者问题，PC问题):



Interprocess Communication

- Shared-Memory systems
  - Producer-Consumer Problem (生产者-消费者问题，PC问题):
    - ✓ Shared data

```
#define BUFFER_SIZE 10
typedef struct {
 . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

    - ✓ Solution is correct, but can only use BUFFER_SIZE-1 elements

Interprocess Communication

- Shared-Memory systems
  - Producer-Consumer Problem (生产者-消费者问题，PC问题):
    - ✓ Bounded-Buffer – Producer

```
while (true) {
   /* Produce an item */
        while (((in + 1) % BUFFER SIZE)  == out)
     ;   /* do nothing -- no free buffers全满 */
      buffer[in] = item;
      in = (in + 1) % BUFFER SIZE;
      }
```

Interprocess Communication

- Shared-Memory systems
  - Producer-Consumer Problem (生产者-消费者问题，PC问题):
    - ✓ Bounded-Buffer – Consumer

```
while (true) {
        while (in == out)
           ; // do nothing -- nothing to consume
     // remove an item from the buffer
     item = buffer[out];
     out = (out + 1) % BUFFER SIZE;
 return item;
    }
```

    - ✓ all empty? all full? ⇒ Solution is “correct “, but can only use BUFFER_SIZE-1 elements or busy waiting

Interprocess Communication

- Message passing (消息传递)
  - ✓ provides a mechanism for processes to communicate and to synchronize their actions without sharing the same address space
  - ✓ processes communicate with each other without resorting to shared variables
  - ✓ particularly useful in a distributed environment.
- IPC facility provides at least two operations:
  - ✓ send(message) – message size fixed or variable
  - ✓ receive(message)
- If process P and Q wish to communicate, they need to:
  - ✓ establish a communication link between them
  - ✓ exchange messages via send/receive
- Implementation of communication link
  - ✓ physical (e.g., shared memory, hardware bus)
  - ✓ logical (e.g., logical properties)

Interprocess Communication

- Implementation Questions
  - How are links established?
  - Can a link be associated with more than two processes?
  - How many links can there be between every pair of communicating processes?
  - What is the capacity of a link?
  - Is the size of a message that the link can accommodate fixed or variable?
  - Is a link unidirectional or bi-directional?

Interprocess Communication

- Direct Communication
  - Processes must name each other explicitly:
    - ✓ send(P, message)- send a message to process P
    - ✓ receive(Q, message)-receive a message from process Q
  - Properties of communication link in this scheme
    - ✓ Links are established automatically
    - ✓ A link is associated with exactly one pair of communicating processes
    - ✓ Between each pair there exists exactly one link
    - ✓ The link may be unidirectional, but is usually bi-directional
  - Symmetry VS asymmetry
    - ✓ send(P, message)
    - ✓ receive(id, message)-receive a message from any process

Interprocess Communication

- Indirect Communication
  - Messages are directed and received from mailboxes (also referred to as ports)
    - ✓ Each mailbox has a unique id (such as POSIX message queues)
    - ✓ Processes can communicate only if they share a mailbox
  - Properties of communication link in this scheme
    - ✓ Link established only if processes share a common mailbox
    - ✓ A link may be associated with more than two processes
    - ✓ Each pair of processes may share several communication links
    - ✓ Link may be unidirectional or bi-directional

Interprocess Communication

- Indirect Communication
  - Operations
    - ✓ create a new mailbox
    - ✓ send and receive messages through mailbox
    - ✓ destroy a mailbox
  - Primitives are defined as:
    - ✓ send(A, message) – send a message to mailbox A
    - ✓ receive(A, message) – receive a message from mailbox A

Interprocess Communication

- Indirect Communication
  - Mailbox sharing
    - ✓ P1, P2, and P3 share mailbox A
    - ✓ P1, sends; P2 and P3 receive
    - ✓ Who gets the message?
  - Solutions
    - ✓ Allow a link to be associated with at most two processes
    - ✓ Allow only one process at a time to execute a receive operation
    - ✓ Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

Interprocess Communication

- Synchronization
  - Message passing may be either blocking or non-blocking
    - ✓ Blocking is considered synchronous
    - ✓ Blocking send has the sender block until the message is received
    - ✓ Blocking receive has the receiver block until a message is available
  - Non-blocking is considered asynchronous
    - ✓ Non-blocking send has the sender sending the message and continue
    - ✓ Non-blocking receive has the receiver receiving a valid message or null

Interprocess Communication

- Buffering
  - ✓ Zero capacity – 0 messages
    Sender must wait for receiver (rendezvous)
  - ✓ Bounded capacity – finite length of n messages
    Sender must wait if link full
  - ✓ Unbounded capacity – infinite length
    Sender never waits