操作系统原理与设计

第 3 章 Processes（进程）1

陈香兰

中国科学技术大学计算机学院

March 12, 2014

提纲

多道程序技术和程序并发执行的条件

- 多道程序技术的难点
- Seriel execution of programs (程序的顺序执行)
- Concurrent execution of programs (程序的并发执行)

Process Concept

- the Processes
- Process State
- Process Control Block (PCB)

小结和作业

Outline

Multiprogramming(多道程序) 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

some easily confused terms

- In our course:
  - **Program**(程序): passive entity, usually a file containing a list of instructions stored on disk (often called an **executable file**).
  - **Tasks**(任务): a general reference
  - **Jobs**(作业): in batch system, user programs (and data) waiting to be loaded and executed
  - **Processes**(进程): a program in execution
- Usually, the term **job** and **process** are used **interchangeably**.

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 **current** execution of programs based on precedence graph

Precedence Graph (前驱图) I

- **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 (前驱关系) "$\rightarrow$",
  $\rightarrow = \{(P_i, P_j) \mid P_i$必须在$P_j$开始执行前执行完$\}$

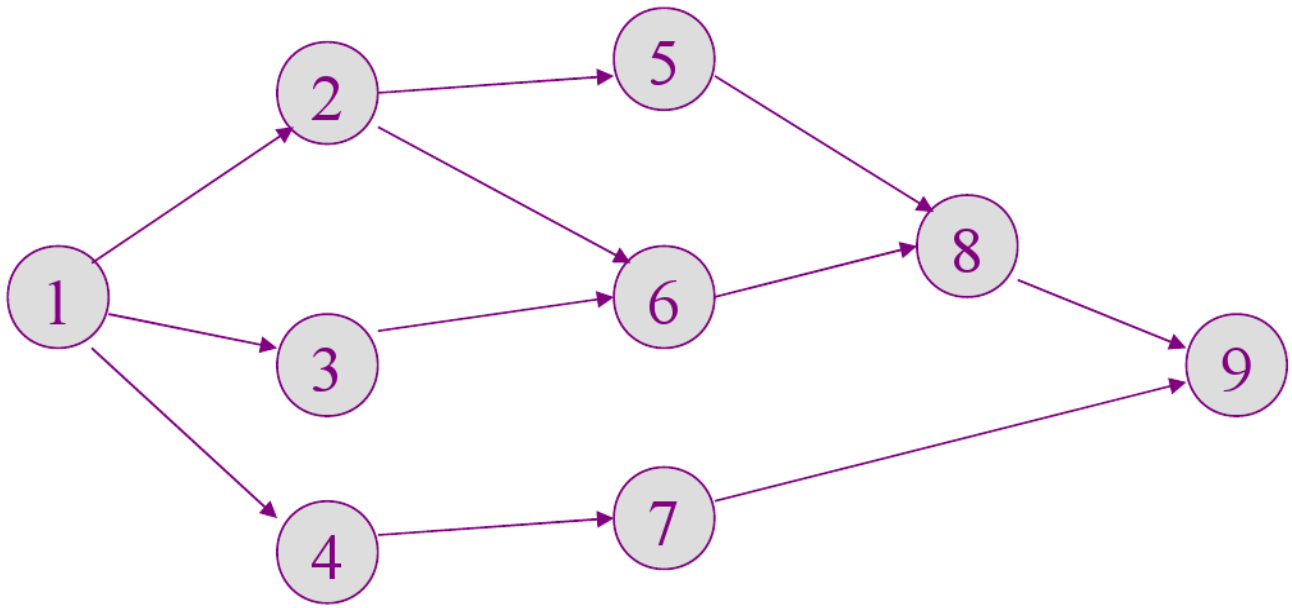
- If $(P_i, P_j) \in \rightarrow$, then $P_i \rightarrow P_j$
  Here, P_i is called the **predecessor(前趋)** of P_j, and P_j the **subsequent(后继)** of P_i

Precedence Graph (前驱图) II

- 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

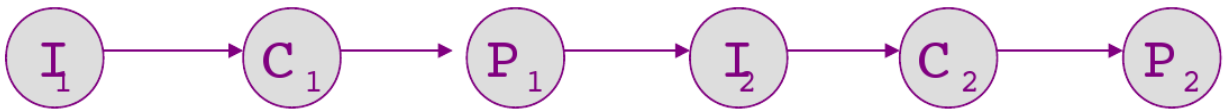
- Example:



Outline

Seriel execution of programs(程序的顺序执行)

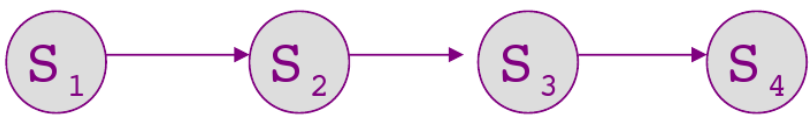
- 一个较大的程序通常包含若干个程序段。程序在执行时，必须按照某种先后顺序逐个执行，仅当前一个程序段执行完，后一个程序段才能执行。
  例如



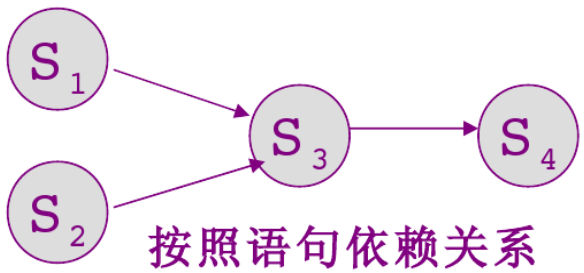
  - 其中
    - I 代表用户程序和数据的输入;
    - C 代表计算;
    - P 代表输出结果

Seriel execution of programs(程序的顺序执行)

- 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



若按指令地址顺序执行



按照语句依赖关系

程序顺序执行时的特征

1 顺序性

- 严格按照程序规定的顺序执行

2 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。

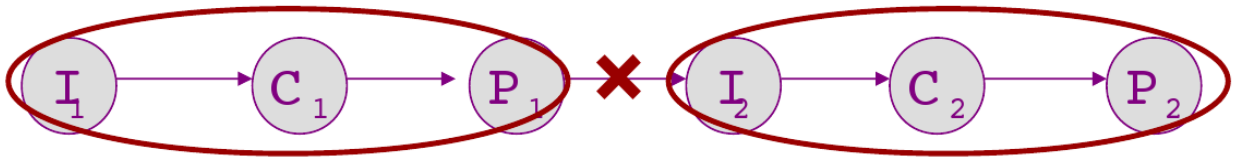
3 可再现性

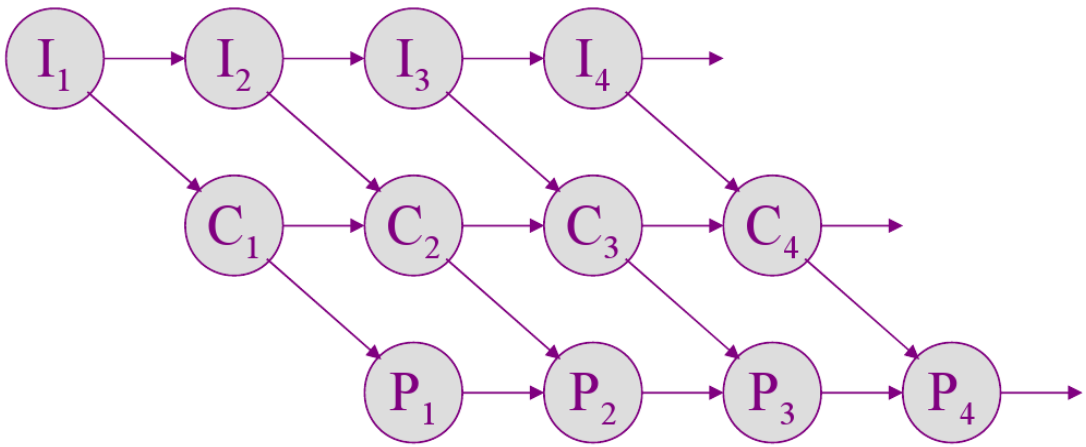
- 只要程序执行时的环境和初始条件相同，都将获得相同的结果。

Outline

Concurrent execution of programs (程序的并发执行)

- P_i 与 I_{i+1} 之间不存在内在的前趋关系





程序并发执行时的前趋图

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

程序并发执行的条件 (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** $R(p_i)$，表示程序 p_i 在执行时需要参考的所有变量的集合
    - **写集** $W(p_i)$，表示程序 p_i 在执行期间要改变的所有变量的集合
  - 1966, Bernstein: if programs p_1 and p_2 meet the following conditions, they can be executed **concurrently**, and have **reproducibility (可再现性)**
    1. If process p_i writes to a memory cell M_i, then no process p_j can read the cell M_i.
    2. If process p_i read from a memory cell M_i, then no process p_j can write to the cell M_i.
    3. If process p_i writes to a memory cell M_i, then no process p_j can write to the cell M_i.

$$R(p_1) \bigcap W(p_2) \bigcup R(p_2) \bigcap W(p_1) \bigcup W(p_1) \bigcap W(p_2) = \varnothing$$

Outline

the processes

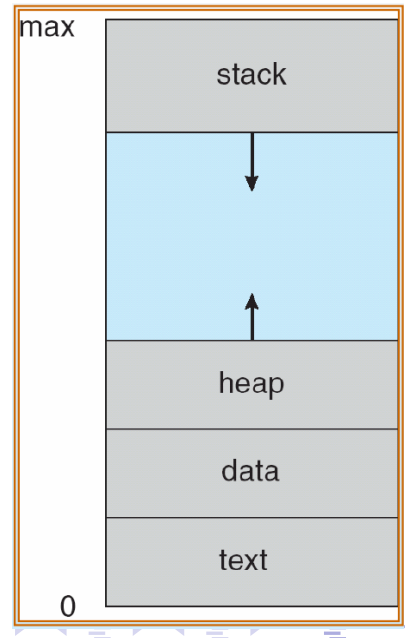
- 进程需要使用某种方法加以描述，原因
  - 进程运行的间断性，要求在进程暂停运行时记录该程序的现场，以便下次能正确的继续运行
  - 资源的共享，要求能够记录进程对资源的共享情况
  - 为保证程序“正确”的并发执行，必须将进程看成某种对象，对其进行描述并加以控制

Process Concept I

- An OS executes a variety of programs:
  - Batch system - **jobs**
  - Time-shared systems - **user programs or tasks**
  - PC - **several programs**: a word processor, a web browser, etc.
- we call all of them **process**
  - a program in execution;
  - process execution must progress in sequential fashion

A process includes:

- text section⇐program code
- program counter + other registers⇐current activity
- stack⇐temporary data
- data section⇐global variables
- heap



Process Concept II

COMPARE: Program vs. Process?

- Program: a passive entity (静态的)
- Process: a active entity (活动的)

进程的五大特征 I

1. **动态性**

   “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
   - **最基本的特性**

2. **并发性**

   - 多道
   - 既是进程也是 OS 的重要特征

3. **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。

4. **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。

5. **结构特征**

Outline

Process State

- As a process executes, it changes **state**.

State Models (状态模型)

- 最基本的“三状态”模型
- 引入“新”和“终止”态的“五状态”模型
- 引入“挂起”状态的“七状态”模型

“三状态”模型

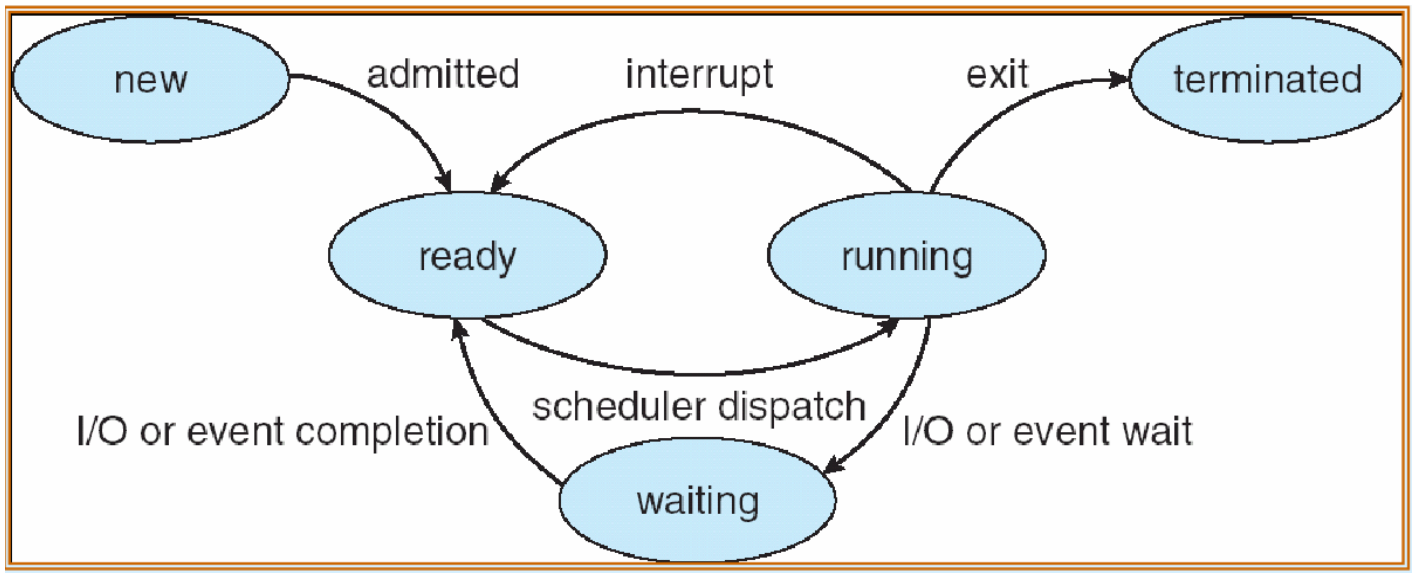
- 三种最基本的状态
  - **ready** (就绪): “万事具备，只欠 CPU”
    - ready queue
  - **running** (执行)
  - **waiting** (等待, also blocked(阻塞), sleeping(睡眠))
    The process is waiting for some event to occur:
    1. I/O completion
    2. reception of a signal
    3. resource allocation
    4. etc.

    - waiting queue

“五状态”模型

- Two more states is added to the “three state” model.
  - **new** (新状态): The process is beig created
    - initialization, resource preallocation, etc.
  - **terminated** (终止状态): The process has finished execution, normally or abnormally.
    - removed from ready queue, but still not destroyed.
    - other process may gather some information from the terminated processes

Diagram of Process State

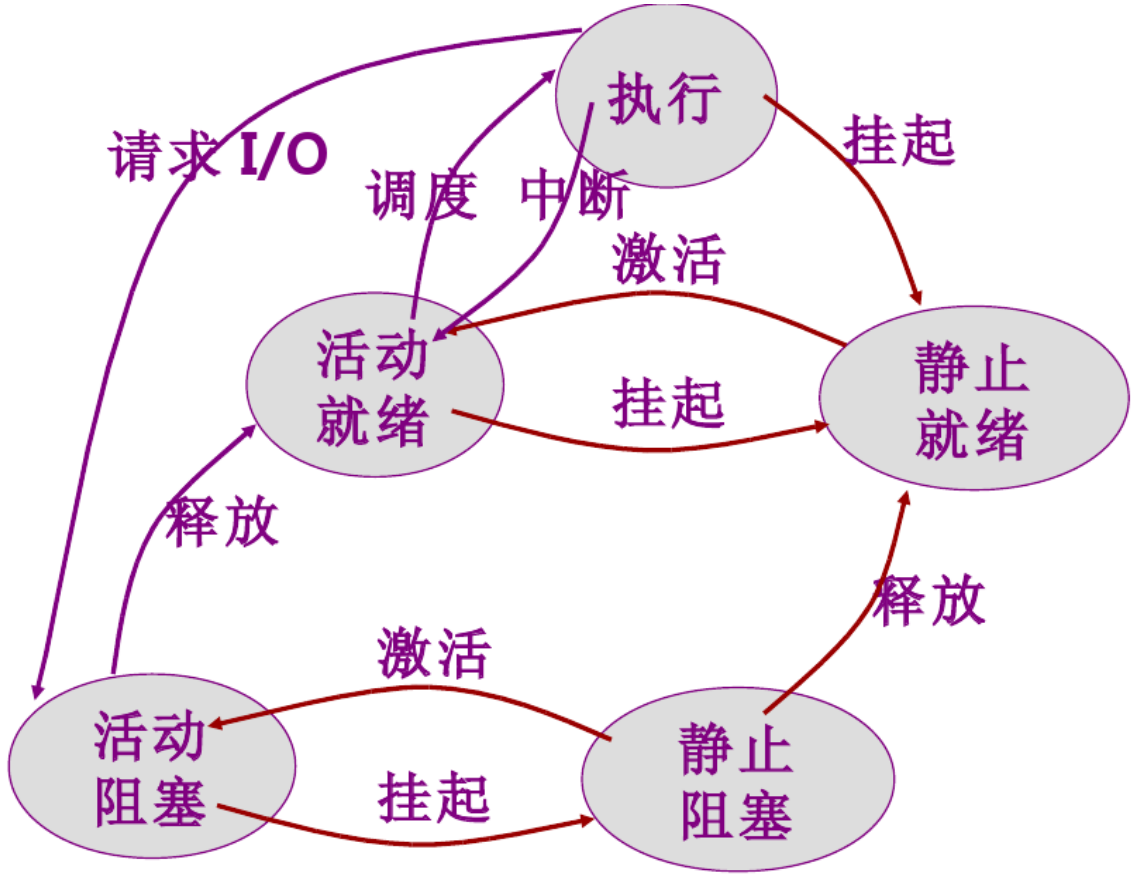


6 types of state transferring

“Seven state” model I

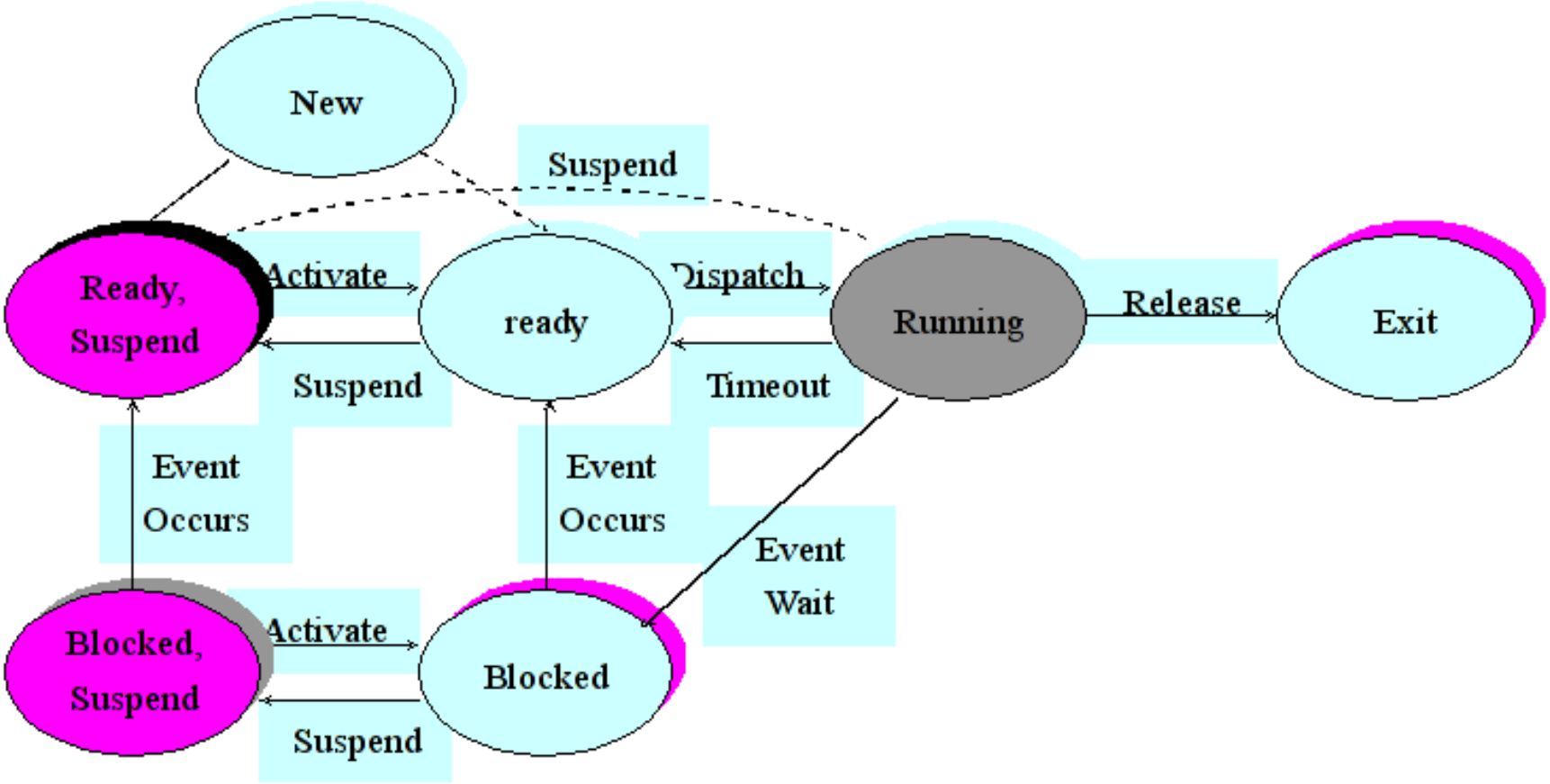
- 进程因自身内部的一些原因，无法继续运行时，暂时进入“等待”状态，当等待的原因消除后，就可以返回就绪状态；
- 但有时候会因为进程外部的一些原因，使得进程暂时不能继续运行。外部原因主要有
  - 终端用户的需要
  - 父进程的需求
  - 操作系统的需要
  - 对换的需要
  - 负载调节的需要
- 引入“挂起”状态
- “挂起”状态不是一种状态，而是**一类状态**
  - 挂起后处于静止状态：**静止就绪**，**静止阻塞**
  - 非挂起的活动状态：**活动就绪**，**活动阻塞**，还包括**执行态**
- 在状态转换中，增加了从活动状态与静止状态之间，以及静止状态内部之间的状态转换

“Seven state” model II



新增 6 种状态转换

“Seven state” model III



Outline

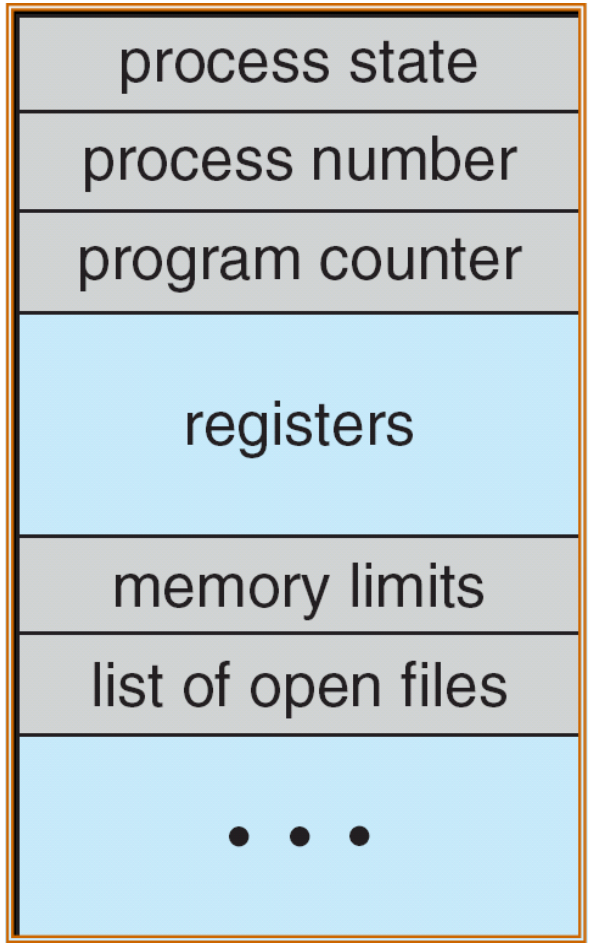
Process Control Block (进程控制块, PCB)

- 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

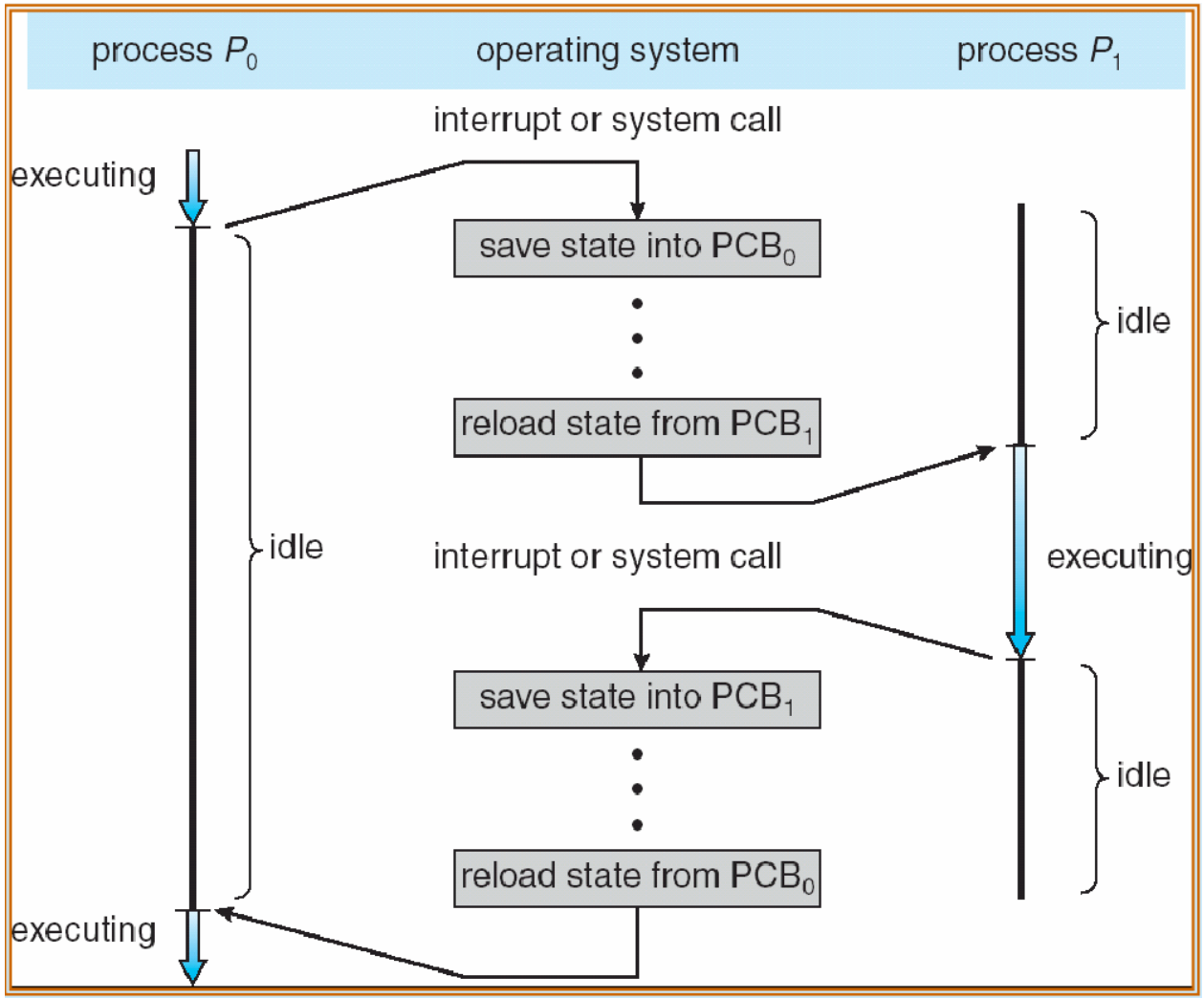
PCB 是进程存在的唯一标志

Process Control Block (进程控制块, PCB)

- Information associated with each process
  - Process **state** (...)
  - **Program counter**
  - CPU **registers**
  - CPU-scheduling information
  - Memory-management information
  - Accounting information: time used, time limit, ...
  - I/O status information



CPU Switch From Process to Process



Examples I

- 观察:
  - 数据结构
  - 状态
- struct task_struct in Linux0.11 & Linux 2.6.26
- struct OS_TCB in μC/OS-II

```
typedef struct os_tcb {
    OS_STK *OSTCBStkPtr; /* Pointer to current top of stack */
#if OS_TASK_CREATE_EXT_EN > 0
    void *OSTCBExtPtr; /* Pointer to user definable data for TCB extension */
    OS_STK *OSTCBStkBottom; /* Pointer to bottom of stack */
    INT32U OSTCBStkSize; /* Size of task stack (in number of stack elements) */
    INT16U OSTCBOpt; /* Task options as passed by OSTaskCreateExt() */
    INT16U OSTCBId; /* Task ID (0..65535) */
#endif
    struct os_tcb *OSTCBNext; /* Pointer to next TCB in the TCB list */
    struct os_tcb *OSTCBPrev; /* Pointer to previous TCB in the TCB list */
#if ((OS_Q_EN > 0) && (OS_MAX_QS > 0)) || (OS_MBOX_EN > 0) ||
(OS_SEM_EN > 0) || (OS_MUTEX_EN > 0)
    OS_EVENT *OSTCBEventPtr; /* Pointer to event control block */
#endif
```

Examples II

```
#if ((OS_Q_EN > 0) && (OS_MAX_QS > 0)) || (OS_MBOX_EN > 0)
    void *OSTCBMsg; /* Message received from OSMboxPost() or OSQPost() */
#endif
#if (OS_VERSION >= 251) && (OS_FLAG_EN > 0) && (OS_MAX_FLAGS > 0)
#if OS_TASK_DEL_EN > 0
    OS_FLAG_NODE *OSTCBFlagNode; /* Pointer to event flag node */
#endif
    OS_FLAGS OSTCBFlagsRdy; /* Event flags that made task ready to run */
#endif
    INT16U OSTCBDly; /* Nbr ticks to delay task or, timeout waiting for event */
    INT8U OSTCBStat; /* Task status */
    INT8U OSTCBPrio; /* Task priority (0 == highest, 63 == lowest) */
    INT8U OSTCBX; /* Bit position in group corresponding to task priority (0..7) */
    INT8U OSTCBY; /* Index into ready table corresponding to task priority */
    INT8U OSTCBBitX; /* Bit mask to access bit position in ready table */
    INT8U OSTCBBitY; /* Bit mask to access bit position in ready group */
#if OS_TASK_DEL_EN > 0
    BOOLEAN OSTCBDelReq; /* Indicates whether a task needs to delete itself */
#endif
} OS_TCB;
```

小结

多道程序技术和程序并发执行的条件

- 多道程序技术的难点
- Seriel execution of programs (程序的顺序执行)
- Concurrent execution of programs (程序的并发执行)

Process Concept

- the Processes
- Process State
- Process Control Block (PCB)

小结和作业

作业

- 程序的顺序执行和并发执行有什么异同之处?
- 什么是 Bernstein 条件?
- 对于下面的语句:

  S_1: $a = 5 - x$;
  S_2: $b = a \cdot x$;
  S_3: $c = 4 \cdot x$;
  S_4: $d = b + c$;
  S_5: $e = d + 3$

  1. 画出前趋图
  2. 证明 S_2 和 S_3 是可以并发执行的，而 S_3 和 S_4 是不能并发执行的。

- 阅读至少 2 本操作系统相关书籍，给出这些书中关于进程的定义，要列出出处。
- 阅读 linux-0.11 的内核代码，找到其进程数据结构加以分析。说明 linux-0.11 中进程的状态及其转换关系。

谢谢！