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

2009年09月01日

提纲

Outline

多道程序技术

- 从单道→多道
  - 内存必须被多个“程序”共享
  - CPU必须被多个“程序”复用
  - 操作系统必须具备4大基本功能：
    - 处理器管理
    - 内存管理
    - I/O管理
    - 文件管理

容易混淆的几个基本术语

- Program，程序；Tasks，任务
  Jobs，作业；Processes，进程
- 本课程中关于上述几个术语的界定
  - 程序：静态的代码序列，通常以文件为存储介质。代码可以是高级语言的、汇编语言的、指令序列等等。
  - 任务：泛指。
  - 作业：批处理系统中，等待装入内存执行的用户程序和数据。
  - 进程：已经被装载到内存中运行的程序及其外延

多道程序技术的难点

- 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
  - **如何保证正确的并发？**

- 相关理论：
  - 程序并发执行的条件
  - 理论模型：**前趋图**

  - 基于前趋图的程序顺序执行分析
  - 基于前趋图的程序并发执行分析

Precedence Graph 前驱图 I

- **目的：**
  准确的描述语句、程序段、进程之间的执行次序

Definition

前趋图是一个**有向无环图DAG**（Directed Acyclic Graph）

- **结点：**
  一个执行单元（如一条语句、一个程序段或进程）
- **（有向）边：**
  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$\}$

Precedence Graph 前驱图 II

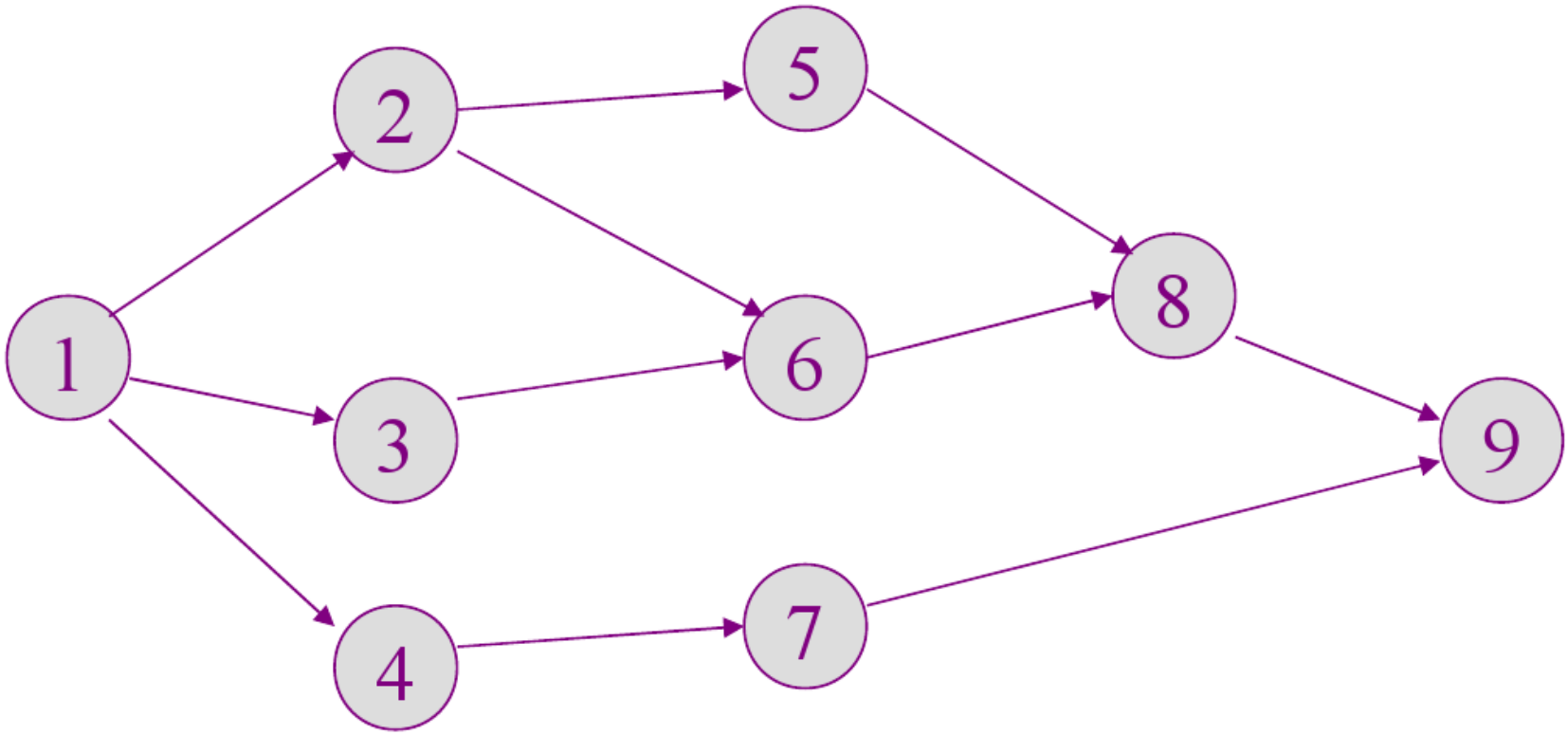
- 若$(P_i, P_j) \in \rightarrow$，可写成$P_i \rightarrow P_j$
  其中，称P_i是P_j的前驱，而P_j是P_i的后继

- 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

- 前趋图举例：

Precedence Graph 前驱图 III



Outline

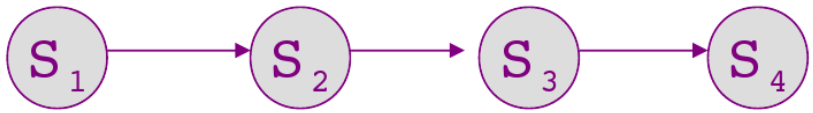
程序的顺序执行 I

- 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

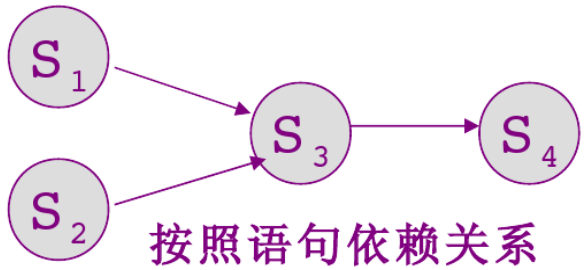
  - 其中
    - I代表用户程序和数据的输入；
    - C代表计算；
    - P代表输出结果

程序的顺序执行 II

- 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

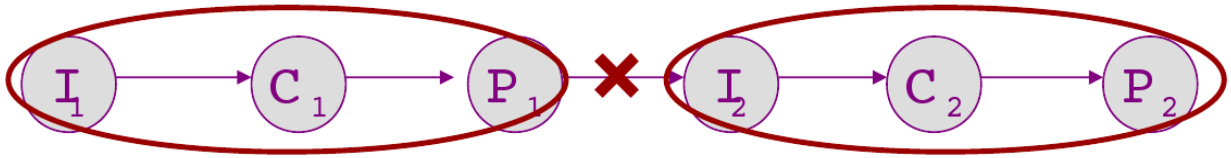
程序顺序执行时的特征

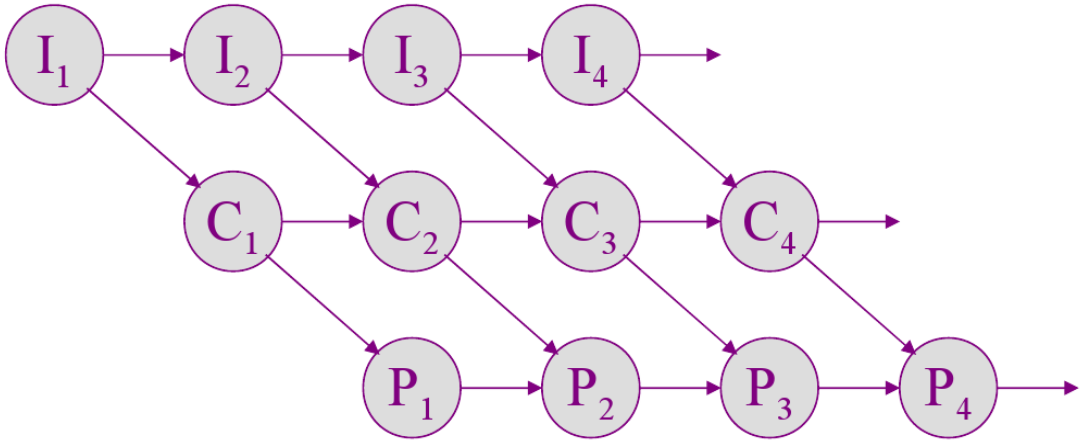
- 顺序性
- 封闭性
- 可再现性

Outline

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条件)

- 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
  - 符号和术语定义：
    - **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：
    若两个程序p_1和p_2满足下列条件，则它们就能并发执行，且具有可再现性

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

Outline

- 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
- Process
  - a program in execution;
  - process execution must progress in sequential fashion

Process Concept II

A process includes:

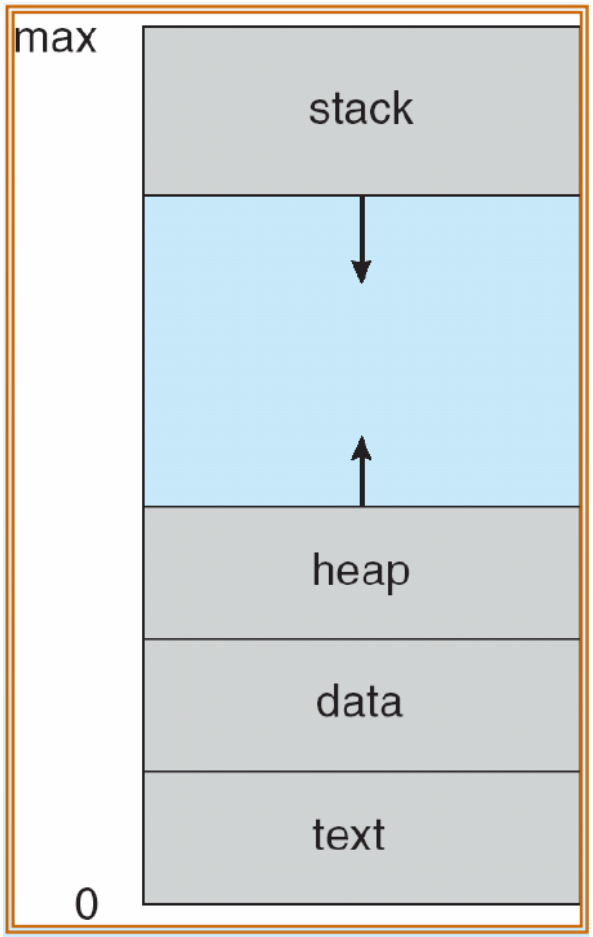
- text section
- program counter + other registers
- stack ???
- data section ???
- heap ???

COMPARE: Program vs. Process?

- 静态的 vs 活动的

Process Concept III

- Process in Memory



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
   - 即是进程也是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。

进程的五大特征 II

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 I

- 根据进程的动态特性，进程在整个生命期中将会处于不同的状态。

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
  - 就绪状态（**ready**）
    - “万事具备，只欠CPU”
    - 就绪队列
  - 执行状态（**running**）
  - 阻塞（等待、睡眠）（**waiting**）状态。
    等待原因：
    1. 发出I/O指令之后，等待I/O操作的完成
    2. 等待某个事件发生
    3. 等待分配到资源

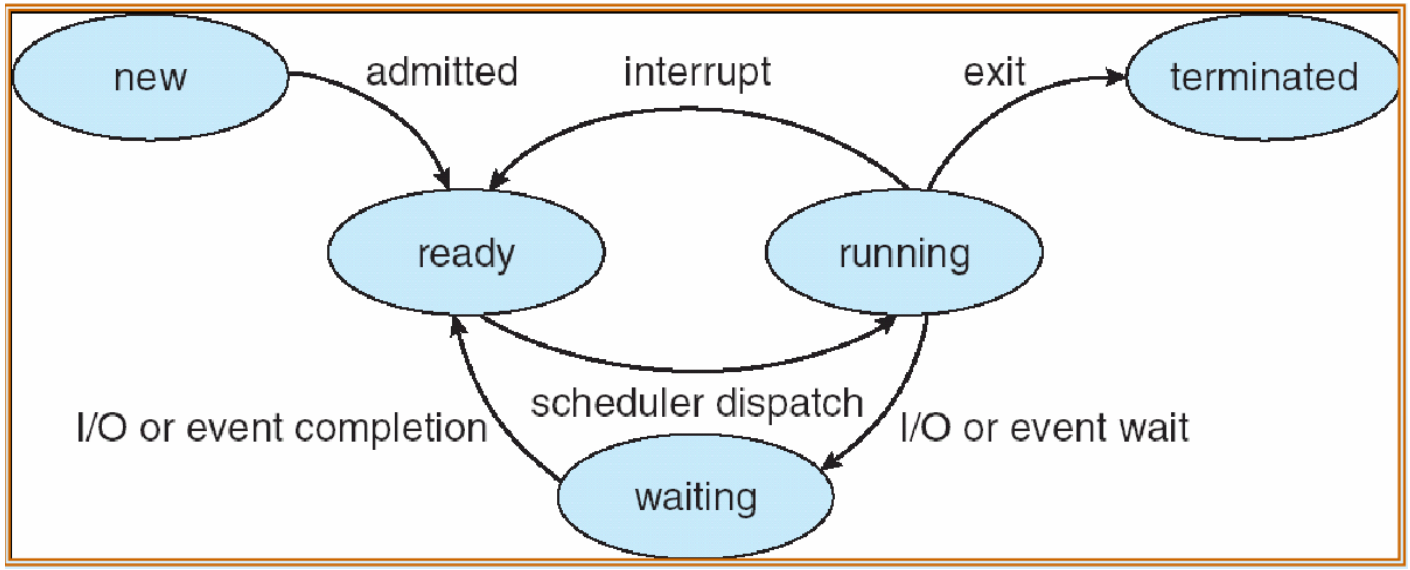
    等等
    - 阻塞（等待）队列，通常按照原因，有多个

“五状态”模型

- 在三种基本状态中，引入
  - 新状态（**new**）：进程刚创建，但还没有进入就绪队列
    - 需要进程初始化，分配一些预设资源等等
  - 终止状态（**terminated**）：进程已经（正常/异常）结束，已经从就绪队列中移出，但尚未撤销
    - 需要将进程暂时留在系统中，以便其他进程去收集该进程的相关信息

Diagram of Process State

- 进程的状态转换图



  - 6种转换关系

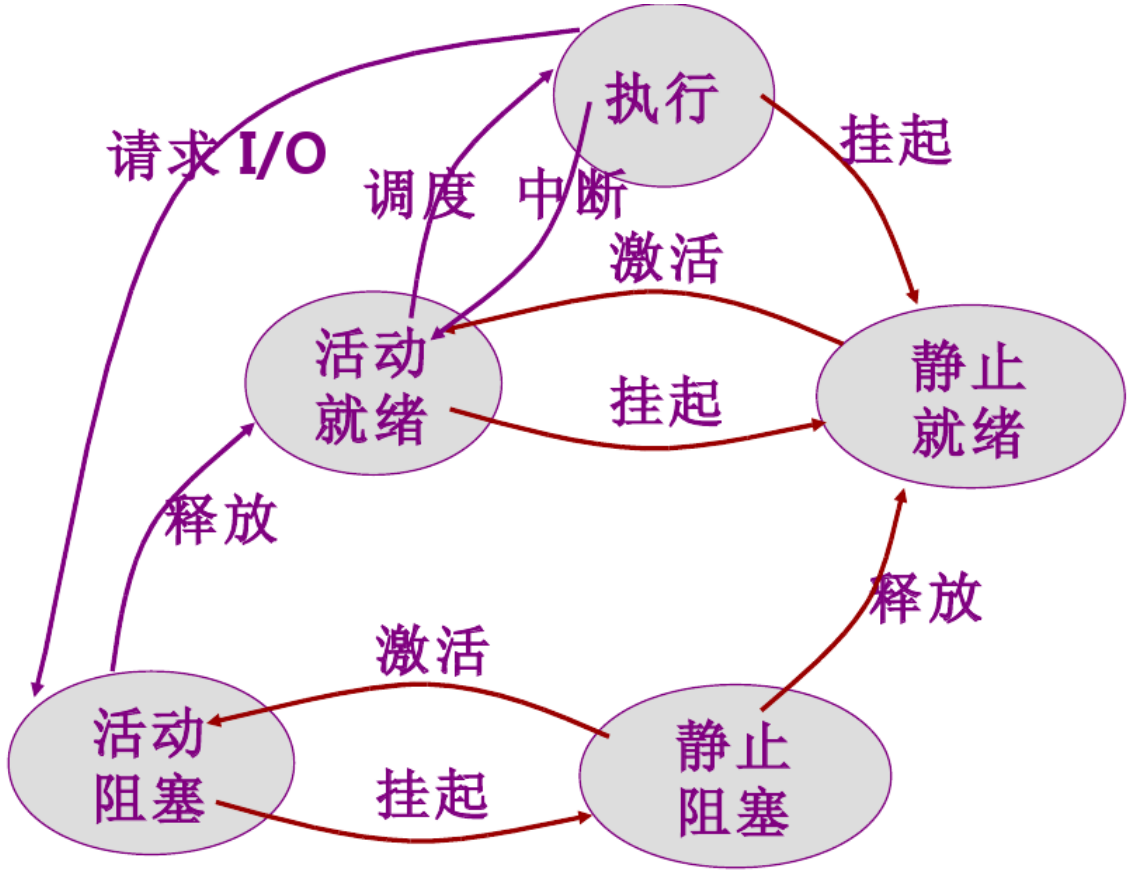
“七状态”模型 I

- 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

“七状态”模型 II

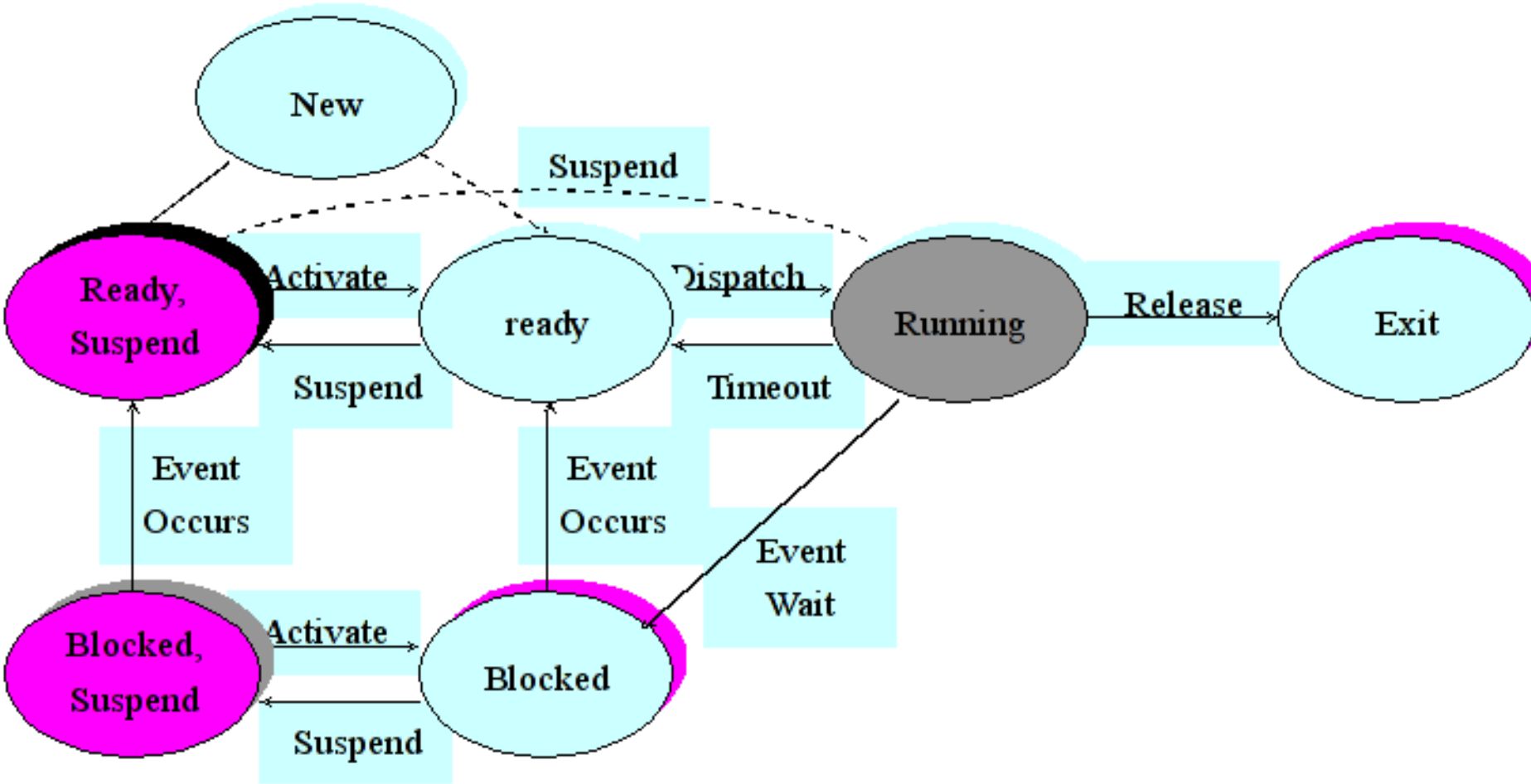
- “挂起”状态不是一种状态，而是**一类状态**
  - 挂起后处于静止状态：**静止就绪**，**静止阻塞**
  - 非挂起的活动状态：**活动就绪**，**活动阻塞**，还包括**执行态**
- 在状态转换中，增加了从活动状态与静止状态之间，以及静止状态内部之间的状态转换

“七状态”模型 III



新增 6 种状态转换

“七状态”模型 IV


New
Suspend
Ready, Suspend
Activate
ready
Dispatch
Running
Release
Exit
Suspend
Timeout
Event Occurs
Event Occurs
Event Wait
Blocked, Suspend
Activate
Blocked
Suspend

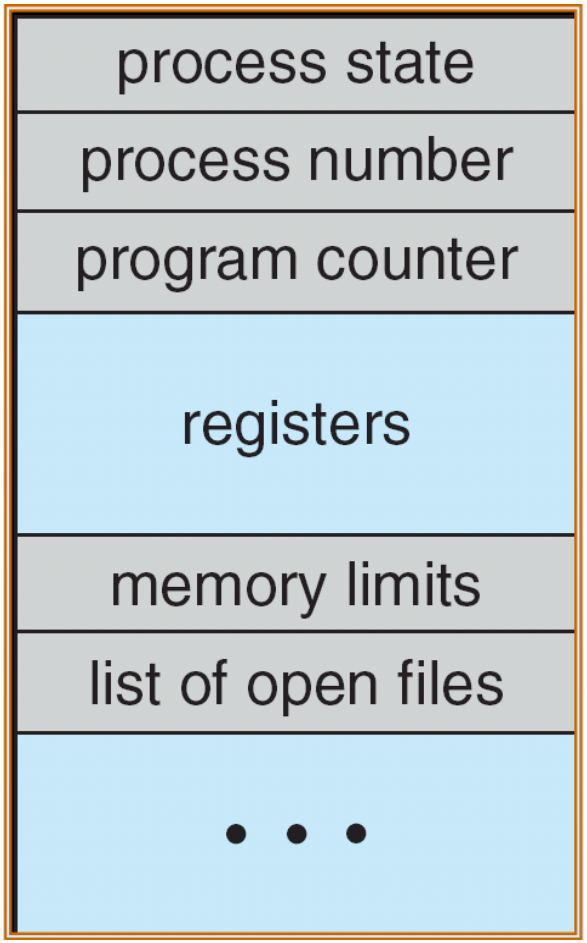
Outline

Process Control Block (PCB) I

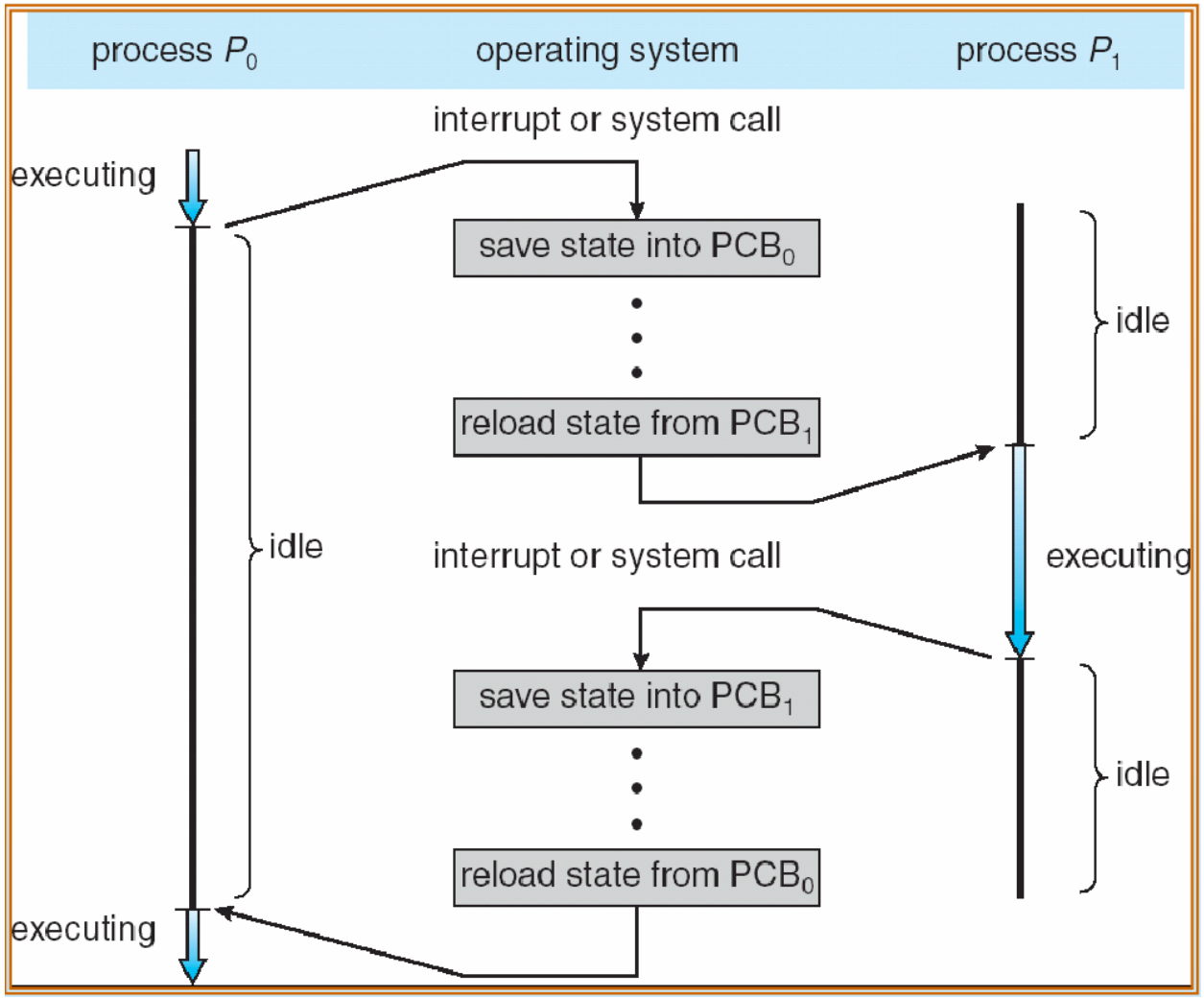
- 进程控制块（Process Control Block，PCB）是操作系统中的一种关键数据结构
  - 由操作系统进程管理模块维护
  - 常驻内存
- 操作系统根据PCB来控制和管理并发执行的进程

PCB是进程存在的唯一标志

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



CPU Switch From Process to Process



Examples

- 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

小结

1. 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
   - 程序的顺序执行
   - 程序的并发执行
2. Process Concept
   - the Process
   - Process State
   - Process Control Block (PCB)
3. 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！