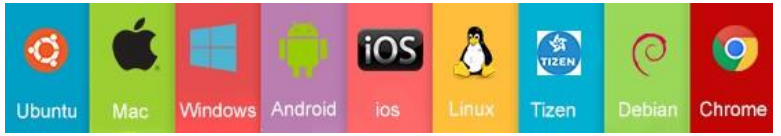
操作系统
Operating System

汤臣薇

tangchenwei@scu.edu.cn

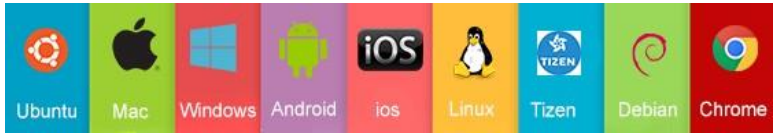
四川大学计算机学院（软件学院）

数据智能与计算艺术实验室



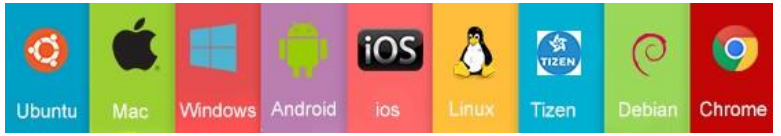
操作系统定义

计算机系统中一组**控制**和**管理**计算机硬件资源和软件**资源**，
合理地对各类作业进行**调度**，以**方便用户**使用的**程序的集合**



操作系统发展历史

- 未配置OS的计算机系统
- 单道批处理系统(simple batch processing)
- 多道批处理系统(multiprogramming system)
- 分时系统(time-sharing system)
- 实时系统(real-time system)
- 微机操作系统(microcomputer operating system)



操作系统基本特性

并发性：进程 ⇨ 两个或两个以上的事件在同一时间段内发生

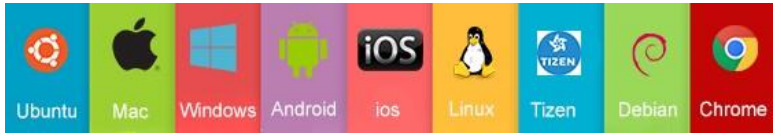
共享性：资源 ⇨ 计算机系统中的资源能够被并发执行的多个进程共同使用

虚拟性：存储 ⇨ 系统通过某种技术将一个实际存在的实体变成多个逻辑上的对应体

异步性：进程 ⇨ 也称为随机性，是指多道程序环境中多个进程的执行、推进和完成时间都是随机、交替、不可预测的

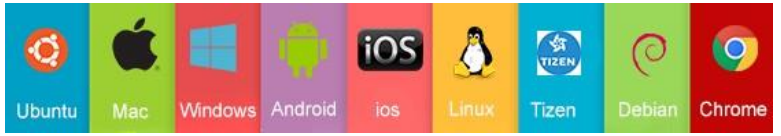
并发和共享是操作系统的两个最基本特性，它们互为存在的条件

不确定性


Ubuntu
Mac
Windows
Android
iOS
Linux
Tizen
Debian
Chrome

操作系统主要功能


处理器管理
存储器管理
设备管理
文件管理
用户接口
OS
功能

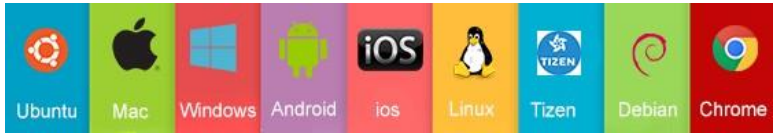


操作系统主要功能

- 处理器管理：
  - 进程和线程的描述与控制
  - 处理器调度
  - 进程或线程的同步与互斥
  - 死锁的检测和预防
  - 进程之间及线程之间的通信

文件管理

设备管理



操作系统主要功能

- 处理器管理：
  - 进程和线程的描述与控制
  - 处理器调度
  - 进程或线程的
  - 死锁的检测和
  - 进程之间及线
- 存储器管理：
  - 内存规划、分配及地址映射
  - 内存保护
  - 内存扩充

文件管理

设备管理

操作系统主要功能

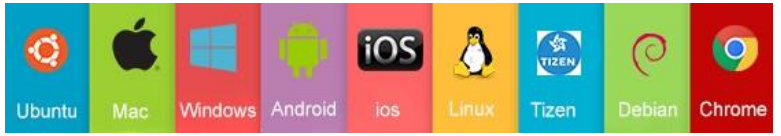
- 处理器管理：
  - 进程和线程的描述与控制
  - 处理器调度
  - 进程或线程的
  - 死锁的检测和
  - 进程之间及线

用

存储器管理：
- 内存规划、分配及地址映射
- 内存保护
- 内存扩充

文件管理

设备管理：
- 输入/输出设备控制
- 设备独立性
- 虚拟设备
- 缓冲管理
- 设备分配
- 磁盘存储器管理



操作系统主要功能

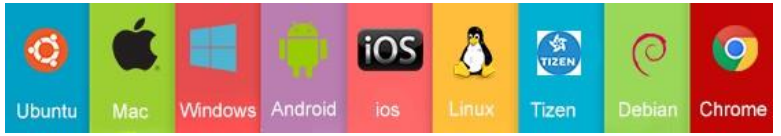
- 处理器管理：
  - 进程和线程的描述与控制
  - 处理器调度
  - 进程或线程的
  - 死锁的检测
  - 进程之间及

用

存储器管理：
- 内存规划、分配及地址映射
- 内存保护
- 内存扩充

文件管理：
- 对文件结构进行组织和目录管理
- 提供文件的存取访问
- 实现文件的存储空间管理
- 实现文件的共享和保护

设备管理：
- 输入/输出设备控制
- 设备独立性
- 虚拟设备
- 缓冲管理
- 设备分配
- 磁盘存储器管理



操作系统主要功能

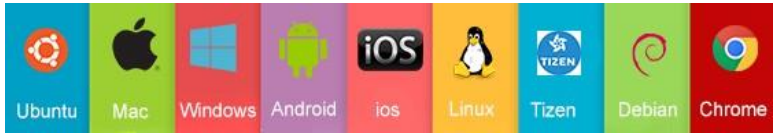
- 处理器管理：
  - 进程和线程的描述与控制
  - 处理器调度

- 用户接口：
  - 程序接口
  - 命令接口
  - 图形接口

- 存储器管理：
  - 内存规划、分配及地址映射
  - 内存保护
  - 内存扩充

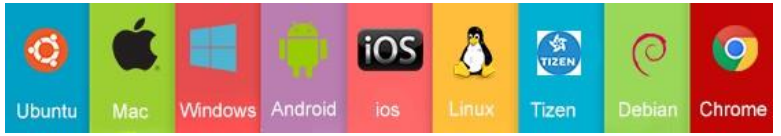
- 文件管理：
  - 对文件结构进行组织和目录管理
  - 提供文件的存取访问
  - 实现文件的存储空间管理
  - 实现文件的共享和保护

- 设备管理：
  - 输入/输出设备控制
  - 设备独立性
  - 虚拟设备
  - 缓冲管理
  - 设备分配
  - 磁盘存储器管理

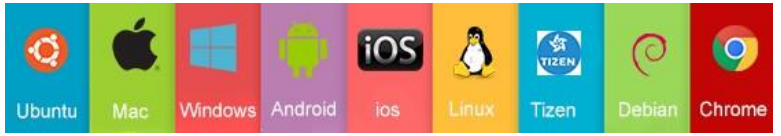


进程的描述和控制

- 前趋图和程序执行
- 进程的描述
- 进程控制
- 进程同步 ★
- 经典进程的同步问题 ★
- 进程通信
- 线程的基本概念
- 线程的实现

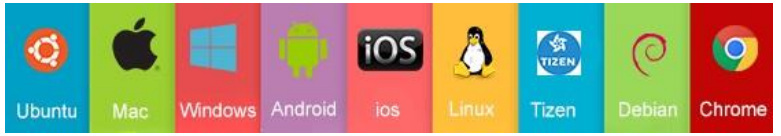


- 前趋图和程序执行
- 进程的描述
- 进程控制
- 进程同步 ★
- 经典进程的同步问题 ★
- 进程通信
- 线程的基本概念
- 线程的实现



前趋图和程序执行

1. 前趋图
2. 程序顺序执行
3. 程序并发执行



2.1 前趋图和程序执行

程序的顺序执行：

- 在内存中仅装入一道用户程序，由它独占系统中的所有资源，只有在一个用户程序执行完成后，才允许装入另一个程序并执行
- 浪费资源、系统运行效率低

单道批处理

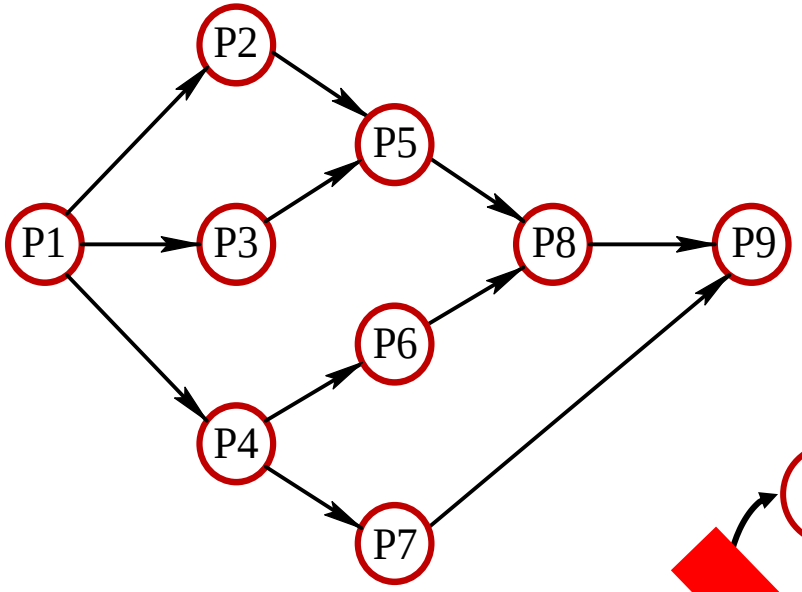
程序的并发执行：

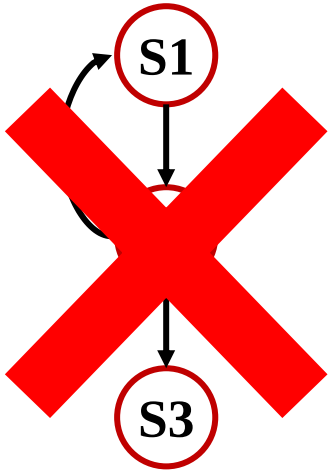
- 内存中同时装入多个程序，共享系统资源，**并发执行**
- 提高资源利用率和运行效率
- 程序并发执行的特征 → **进程**的引入
- **前趋图**：描述程序的顺序执行与并发执行

多道批处理

◆ 前趋图（Precedence Graph）：描述程序执行先后顺序

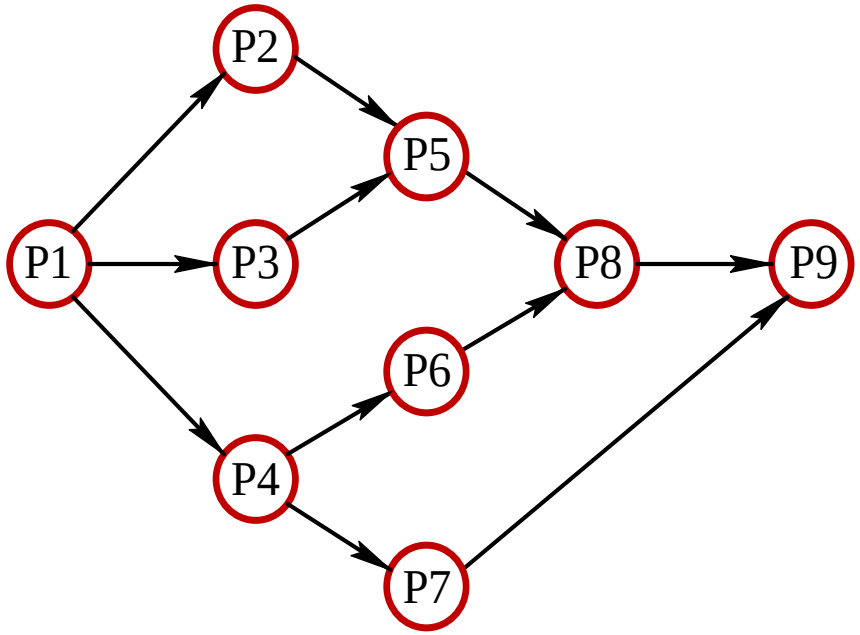
- 描述多个进程之间的关系
- 有向无循环图（DAG）
- 结点表示一个进程或一段程序
- 结点之间用一个有方向的线段相连
- 方向表示所连接的结点之间的前趋和后继关系
- 被指向的结点为后继结点，离开箭头的结点是前趋结点





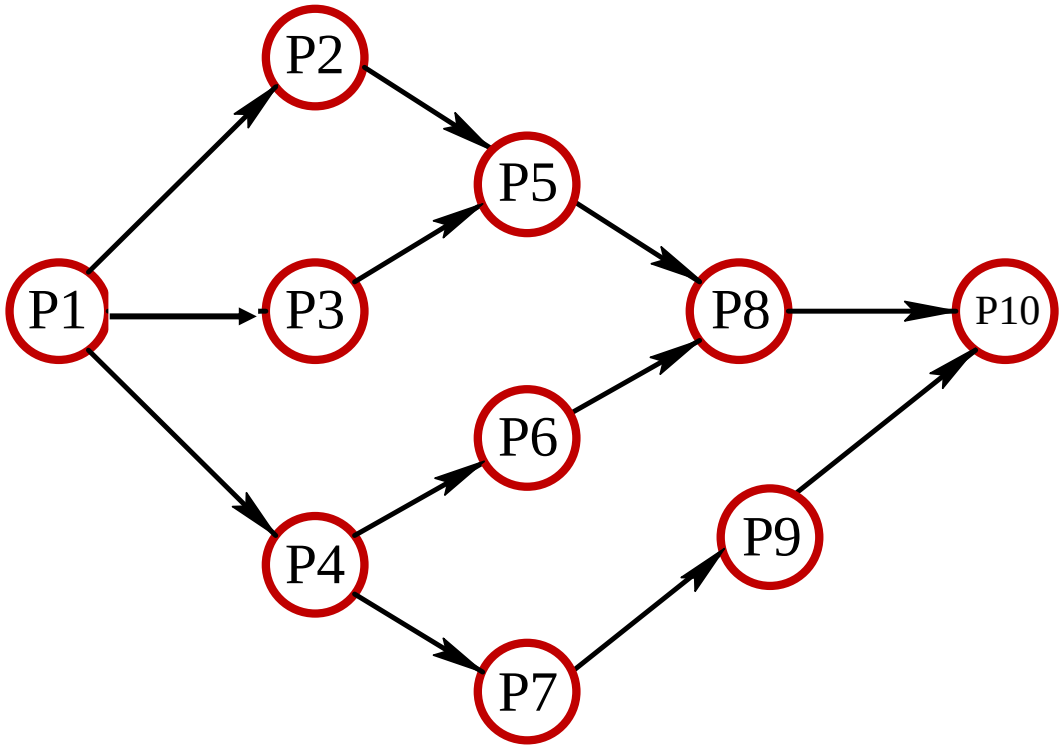
◆ 前趋图（Precedence Graph）

- →={(Pi, Pj)|Pi must complete before Pj may start}, 如果(Pi, Pj)∈→,可写成Pi→Pj， Pi是Pj的直接前趋，Pj是Pi的直接后继
- 在前趋图中，把没有前趋的结点称为初始结点(Initial Node)，把没有后继的结点称为终止结点(Final Node)
- 每个结点还具有一个重量(Weight)，用于表示该结点所含有的程序量或结点的执行时间



◆ 前趋图（Precedence Graph）

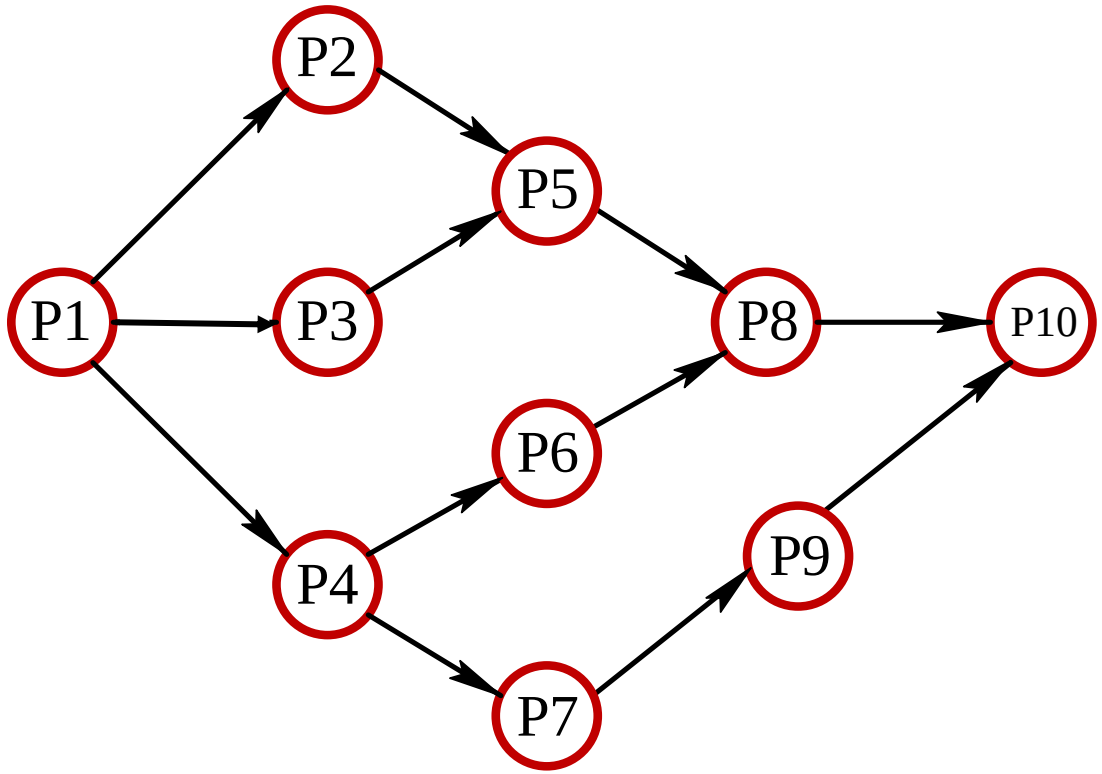
➢请根据前趋图写出偏序关系节点的集合。



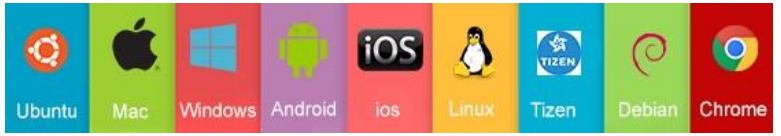
- **G={P, → }**
- **P={P1, P2, P3, P4, P5, P6, P7, P8, P9, P10}**
- **→={…}**

◆ 前趋图（Precedence Graph）

➢请根据前趋图写出偏序关系节点的集合。



- **G={P, → }**
- **P={P1, P2, P3, P4, P5, P6, P7, P8, P9}**
- **→={ (P1, P2), (P1, P3), (P1, P4), (P2, P5), (P3, P5), (P4, P6), (P4, P7), (P5, P8), (P6, P8), (P7, P9), (P8, P10), (P9,P10)}**



◆ 前趋图（Precedence Graph）

➢根据下述五条语句的程序段绘制前趋图，并判断这是顺序执行还是并发执行?

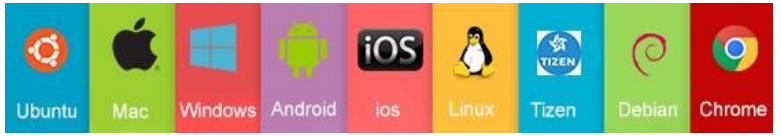
S1: a=5-x

S2: b:=a.x

S3: c=4.x

S4: d=b+c

S5: e=d+3



◆ 前趋图（Precedence Graph）

➢根据下述五条语句的程序段绘制前趋图，并判断这是顺序执行还是并发执行？

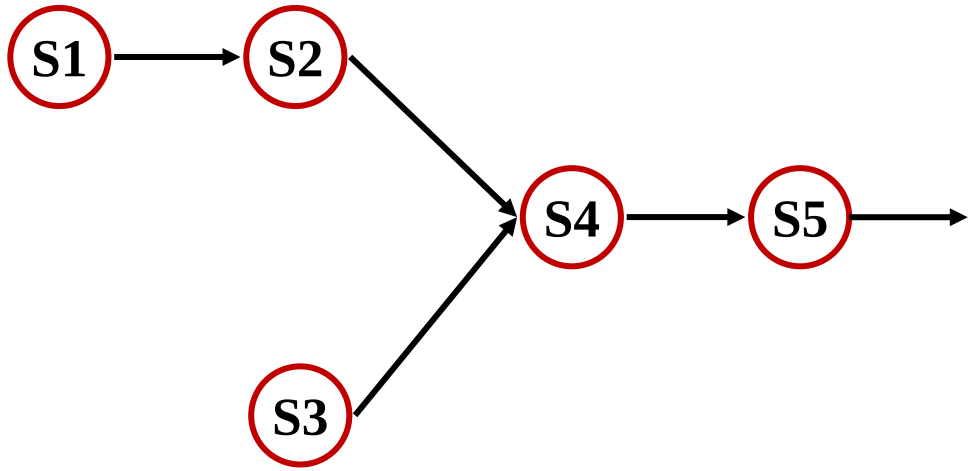
S1: a=5-x

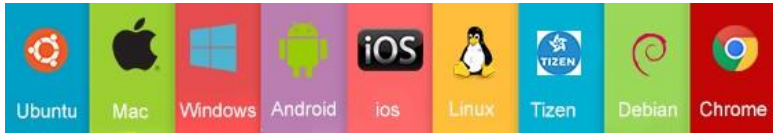
S2: b:=a.x

S3: c=4.x

S4: d=b+c

S5: e=d+3





◆ 前趋图（Precedence Graph）

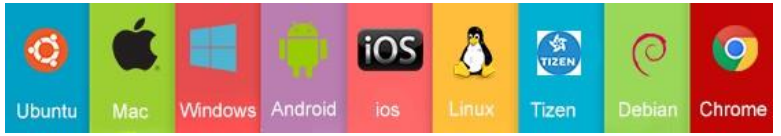
➢根据下述四条语句的程序段绘制前趋图，并判断这是顺序执行还是并发执行?

S1：a:=x+2

S2：b:=y+4

S3：c:=a+b

S4：d:=c+b



◆ 前趋图（Precedence Graph）

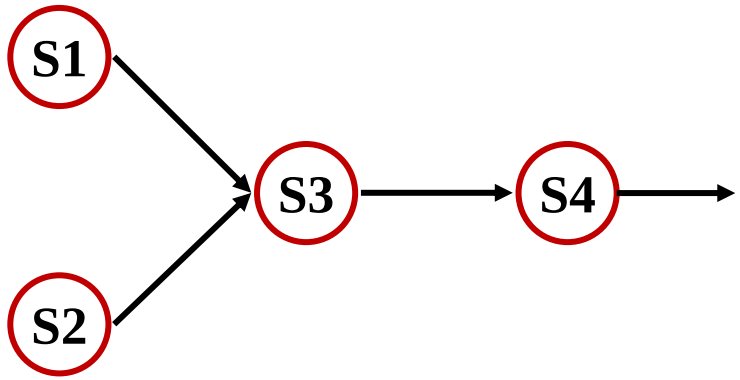
➢根据下述四条语句的程序段绘制前趋图，并判断这是顺序执行还是并发执行?

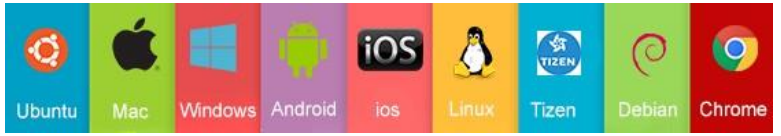
S1：a:=x+2

S2：b:=y+4

S3：c:=a+b

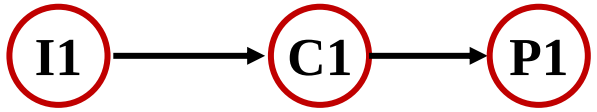
S4：d:=c+b





◆ 程序的顺序执行

- 例1：输入I → 计算C → 打印P



- 例2：

S1：a: = x + y

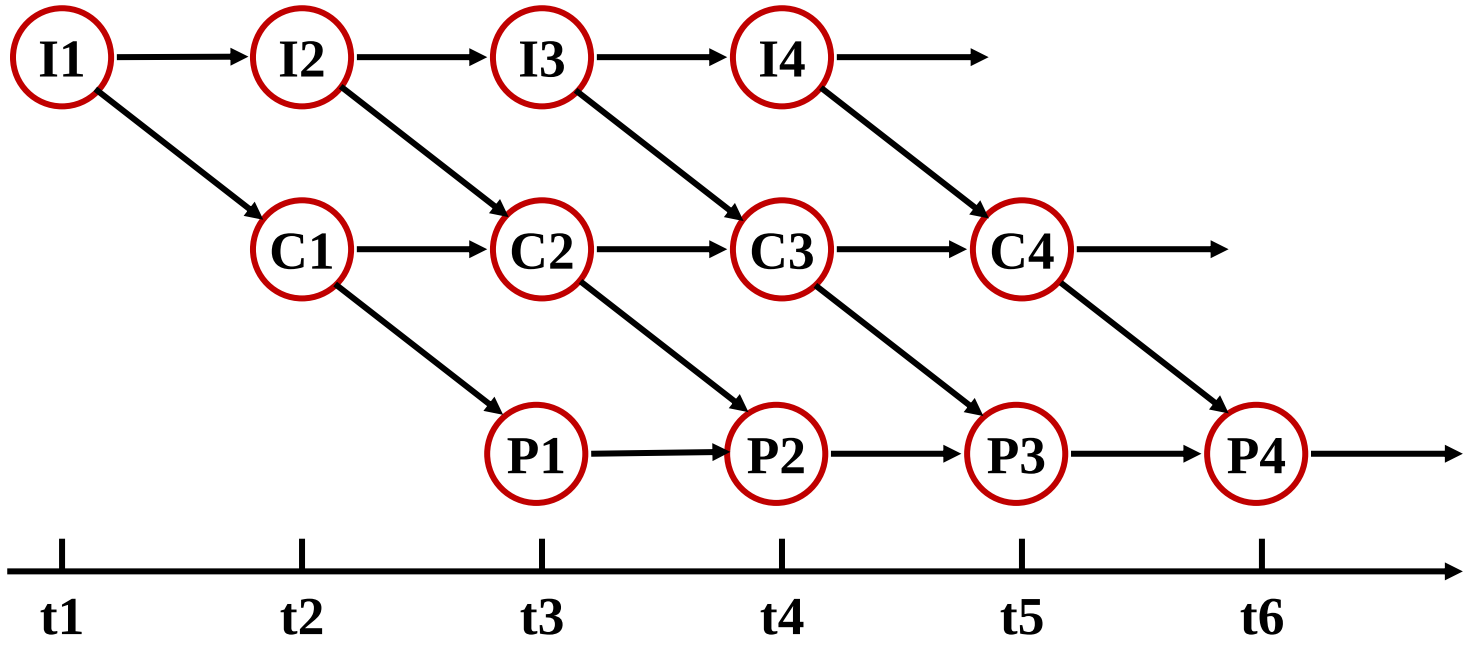
S2：b: = a - 5

S3：c: = b + 1

◆ 程序顺序执行的特征

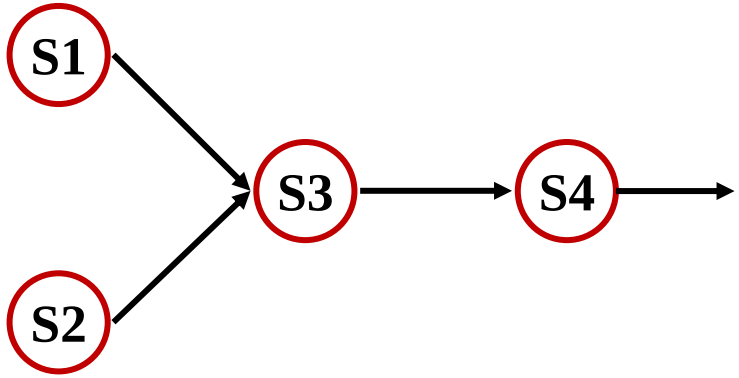
- **顺序性**：严格按照程序所规定的顺序执行
- **封闭性**：程序运行时资源独占，程序一旦执行，其执行结果不受外界因素影响
- **可再现性**：只要程序执行时的环境和初始条件相同，程序重复执行，结果相同

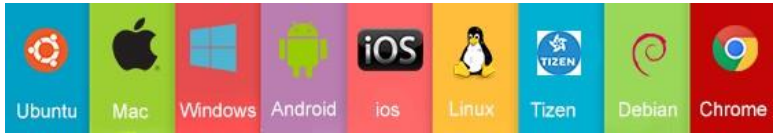
◆ 程序的并发执行



◆ 程序并发执行的特征

- 间断性制约
  - 资源共享致使并发执行的程序形成相互制约的关系
  - 并发程序：执行 → 暂停 → 执行
- 失去封闭性
  - 资源共享导致运行失去封闭性
  - 程序等待
- 不可再现性
  - 计算结果与并发程序的执行速度有关





◆ 程序并执行的特征

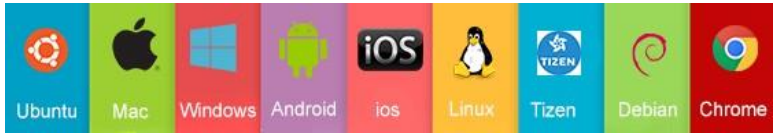
➢ 不可再现性

- 循环程序A：N: = N-1
- 循环程序B：Print（N），N: = N+1
- 程序A、B共享变量N
- 三种情况：

  例1：A→B

  例2：B→A

  例3：A在B的两个操作之间进行



◆ 程序并执行的特征

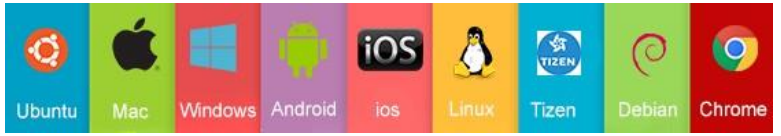
➢ 不可再现性

- 循环程序A：N: = N-1
- 循环程序B：Print（N），N: = N+1
- 程序A、B共享变量N
- 三种情况：

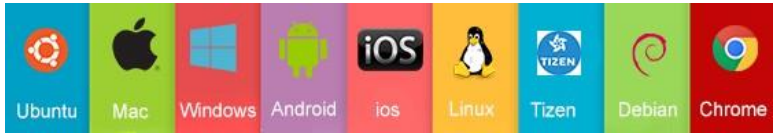
例1：A→B　　**n-1, n-1, n**

例2：B→A　　**n, n+1, n**

例3：A在B的两个操作之间进行　　**n, n-1, n**



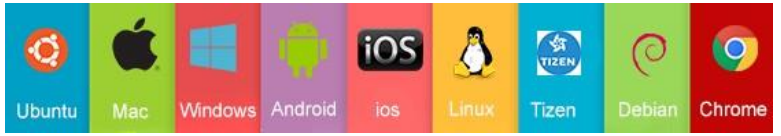
- 前趋图和程序执行
- 进程的描述
- 进程控制
- 进程同步 ★
- 经典进程的同步问题 ★
- 进程通信
- 线程的基本概念
- 线程的实现



前趋图和程序执行

进程的描述

1. 进程的定义和特征
2. 进程的基本状态及转换
3. 挂起操作和进程状态的转换
4. 进程管理中的数据结构



进程是资源分配和处理器调度的基本单位，是操作系统中最活跃的因素，也是操作系统并发性、共享性、虚拟性、异步性的体现

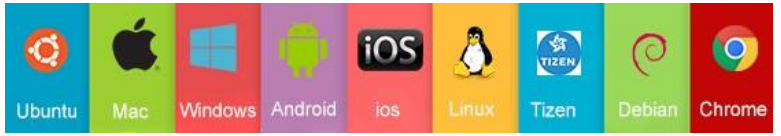
并发性：进程 ⇨ 两个或两个以上的事件在同一时间段内发生

共享性：资源 ⇨ 计算机系统中的资源能够被并发执行的多个进程共同使用

虚拟性：存储 ⇨ 系统通过某种技术将一个实际存在的实体变成多个逻辑上的对应体

异步性：进程 ⇨ 指多道程序环境中多个进程的执行、推进和完成时间都是随机、交替、不可预测的

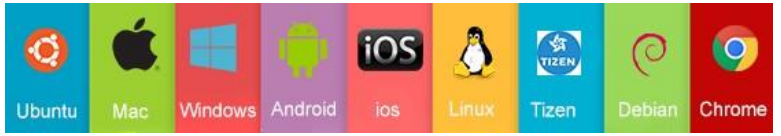
为避免进程切换导致的系统开销过大，现代OS在进程的基础上引入了线程的概念，用线程代替进程成为处理器调度的基本单位



◆ 进程实体（进程映像）：

- **程序段**
- **相关的数据段**
- **进程控制块（Process Control Block，PCB）：**专门的数据结构，用来描述进程的基本情况和活动过程，进而控制和管理进程

- **进程创建**
  - 创建进程实体中的PCB
- **进程撤销**
  - 撤销进程实体中的PCB

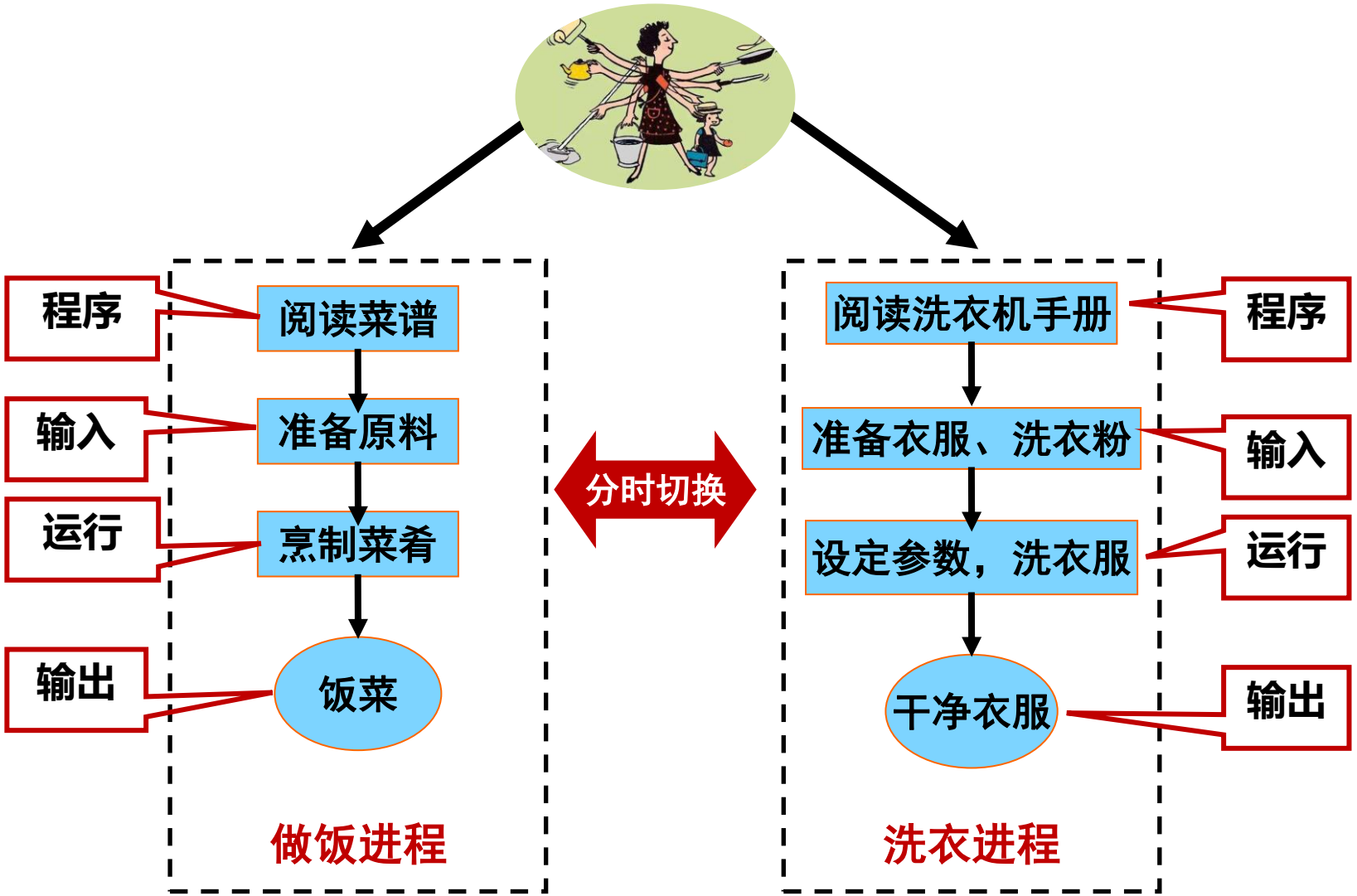


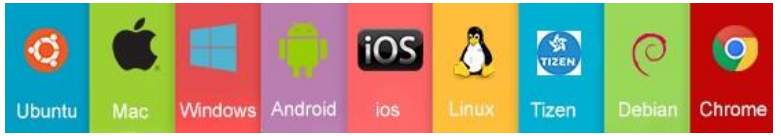
◆ 进程定义：

- 进程是程序的一次执行
- 进程是一个程序及其数据在处理机上顺序执行时所发生的活动
- 进程是具有独立功能的程序在一个数据集合上运行的过程，它是系统进行资源分配和调度的独立单位

进程是进程实体的运行过程，是操作系统进行资源分配和调度的基本单位

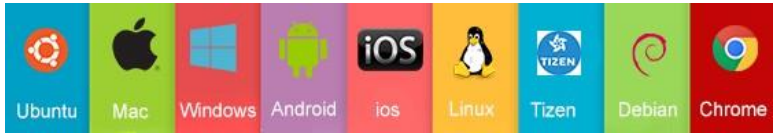
◆ 进程定义（直观举例）：


程序
阅读菜谱
输入
准备原料
运行
烹制菜肴
输出
饭菜
做饭进程
分时切换
阅读洗衣机手册
程序
准备衣服、洗衣粉
输入
设定参数，洗衣服
运行
干净衣服
输出
洗衣进程



◆ 进程概念的理解：

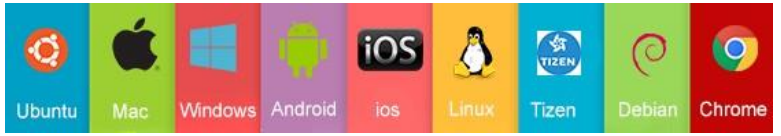
- 进程是程序运行过程
- 进程是以异步为主要特征并具有“活力”的过程
- 操作系统需要用数据结构描述进程
- 进程的运行轨迹是可以控制的
- 进程是资源分配的单位
- 进程与程序不同



◆ 进程概念的理解：

- 进程是程序运行过程

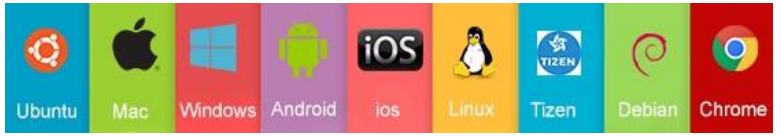
相同的程序可以多次运行，每次运行的环境可能不相同。这种多次运行可以发生在同一时间段中



◆ 进程概念的理解：

- 进程是程序运行过程
- 进程是以异步为主要特征并具有“活力”的过程

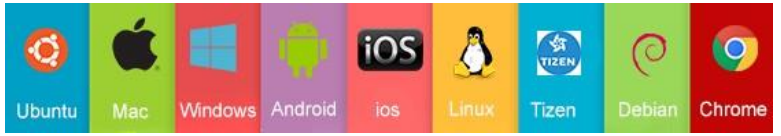
进程的异步特征表现形式是进程推进过程中的走走停停，某一进程被别的进程切换又切换其他的进程；进程的每一步执行都是一个向前推进的过程，是具有“活力”的过程



◆ 进程概念的理解：

- 进程是程序运行过程
- 进程是以异步为主要特征并具有“活力”的过程
- 操作系统需要用数据结构描述进程

操作系统为管理进程，对每个进程用一个数据结构用来描述其详细信息，该数据结构根据进程的动态过程不断更新

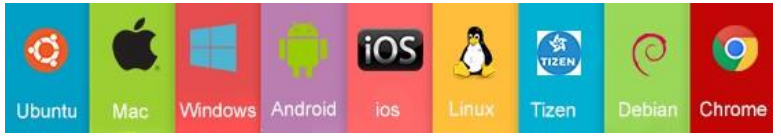


◆ 进程概念的理解：

- 进程是程序运行过程
- 进程是以异步为主要特征并具有“活力”的过程
- 操作系统需要用数据结构描述进程
- 进程的运行轨迹是可以控制的

操作系统通过进程的数据结构了解进程详细信息，

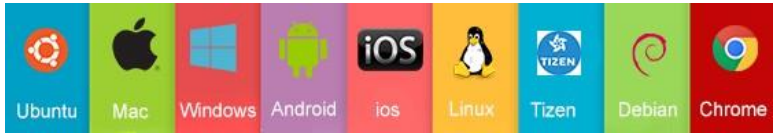
采用进程管理的方法实现对进程运行轨迹的控制



◆ 进程概念的理解：

- 进程是程序运行过程
- 进程是以异步为主要特征并具有“活力”的过程
- 操作系统需要用数据结构描述进程
- 进程的运行轨迹是可以控制的
- 进程是资源分配的单位

在以进程为基础的OS中，系统资源（硬件资源、软件资源，特别是处理器资源）以进程为单位进行分配



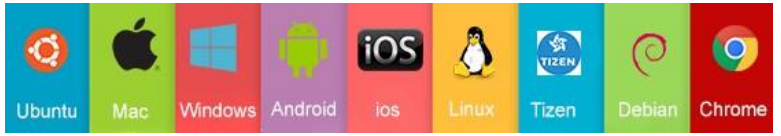
◆ 进程概念的理解：

虽然进程是程序运行，但是进程与程序却不能完全等同。

程序是静态的，是以文件形式存放在磁盘上的代码序列。

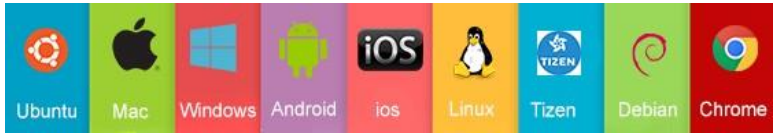
进程是动态的，是不断向前推进的过程，进程具有各种状态并可以在状态之间转换

- 进程与程序不同



◆ 进程的特征

- 结构性
- 动态性
- 并发性
- 独立性
- 异步性



◆ 进程的特征

结构性

进程包含有描述进程信息的数据结构（包含进程控制块、程序块和代码块等）和运行在进程上的程序，OS用PCB描述和记录进程的动态变化过程。

动态性

最基本特征，是程序执行过程，有一定的生命期：由创建而产生、由调度而执行，因得不到资源而暂停，由撤消而死亡。而程序是静态的，它是存放在介质上一组有序指令的集合，无运动的含义。

并发性

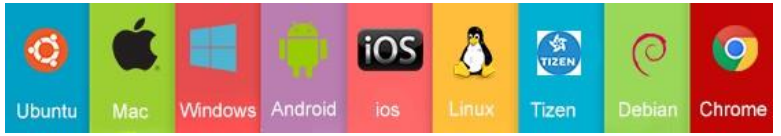
进程的重要特征，也是OS的重要特征。指多个进程实体同存于内存中，能在一段时间内同时运行。而程序（没有建立PCB）是不能并发执行。

独立性

进程是一个能独立运行的基本单位，即是一个独立获得资源和独立调度的单位，而未简历PCB的程序不能作为独立单位参加运行、获取资源。

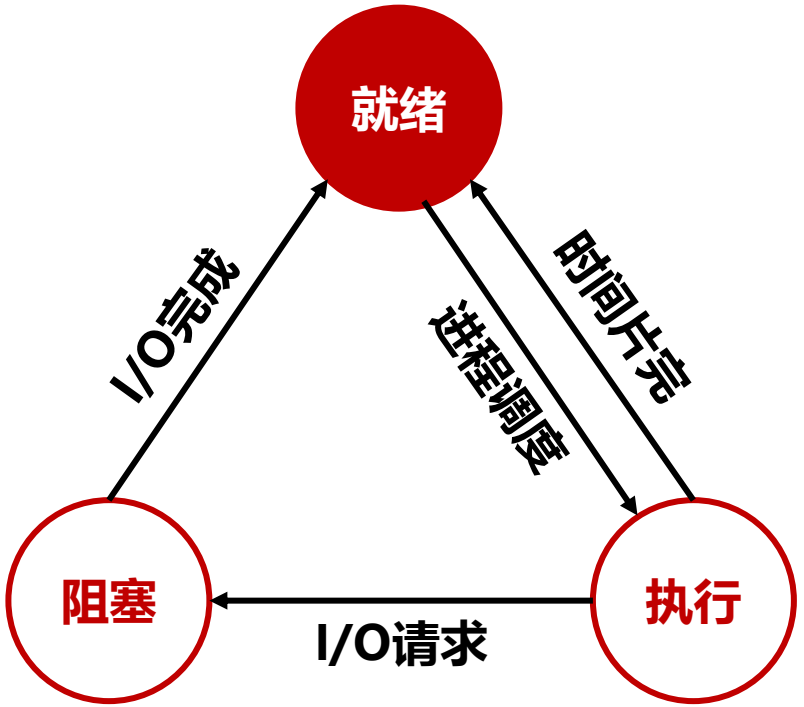
异步性

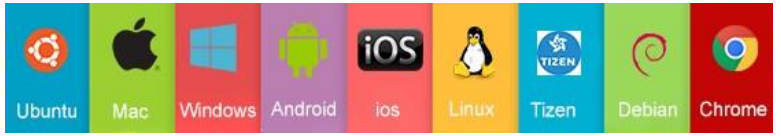
进程按各自独立的不可预知的速度向前推进（即按异步方式进行），正是这一特征导致程序执行的不可再现性，因此OS必须采用某种措施来限制各进程推进序列以保证各程序间正常协调运行。



◆ 进程的三种基本状态

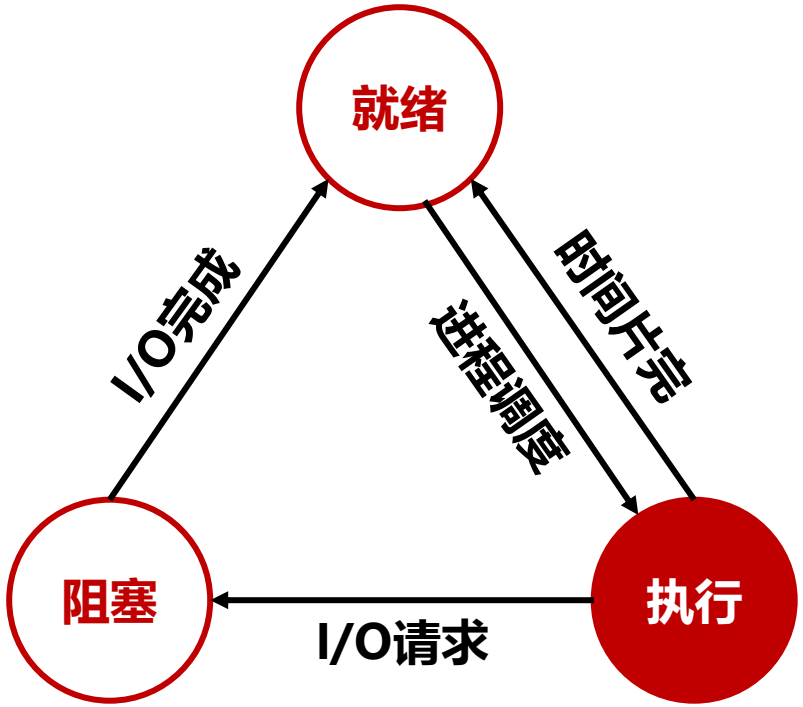
就绪状态：该进程运行所需的一切条件都得到满足，但因处理机资源个数少于进程个数，所以该进程不能运行，而必须在**内存**中等待分配处理机资源，一旦获得处理机就立即投入运行

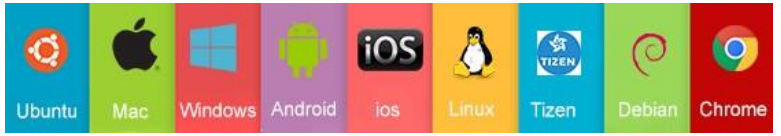




◆ 进程的三种基本状态

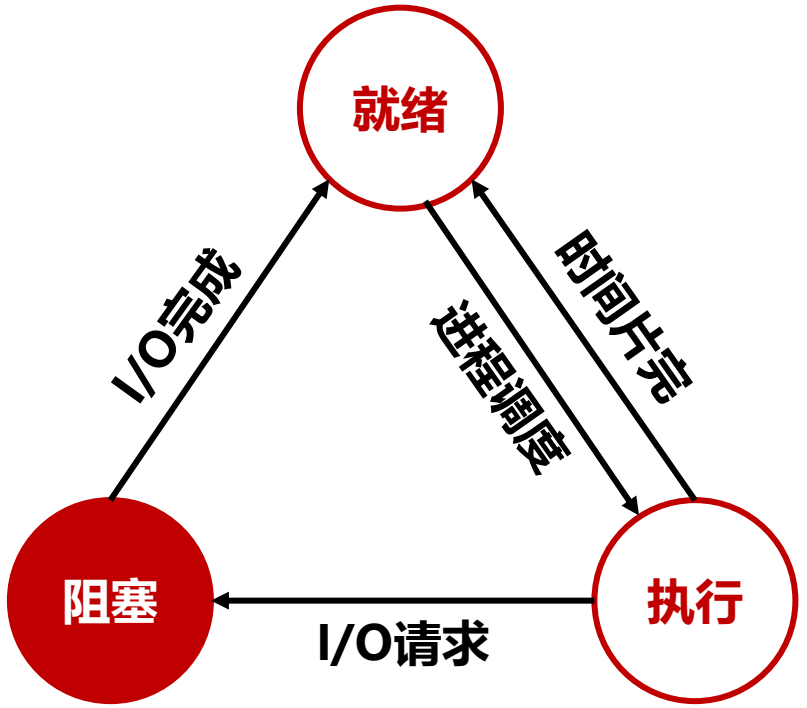
执行状态：进程正在处理机上运行的状态，该进程已获得必要资源，也获得了处理机，用户程序正在处理机上运行





◆ 进程的三种基本状态

阻塞状态：进程等待某种事件完成（如等待输入/输出操作的完成）而暂时不能运行的状态，处于该状态的进程不能参加竞争处理机，此时，即使分配给它处理机，它也不能运行



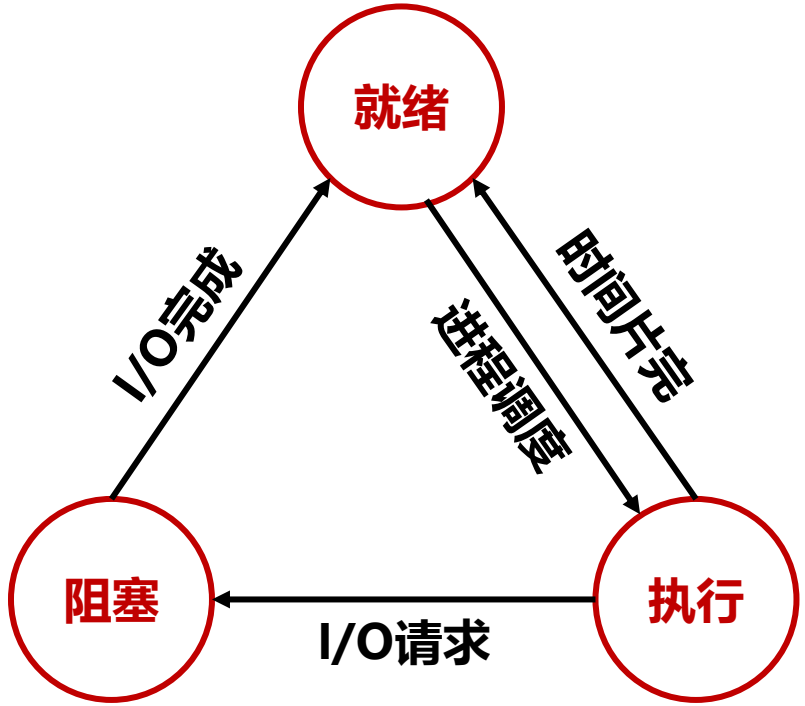
◆ 三种基本状态的转换

就绪状态 → 执行状态

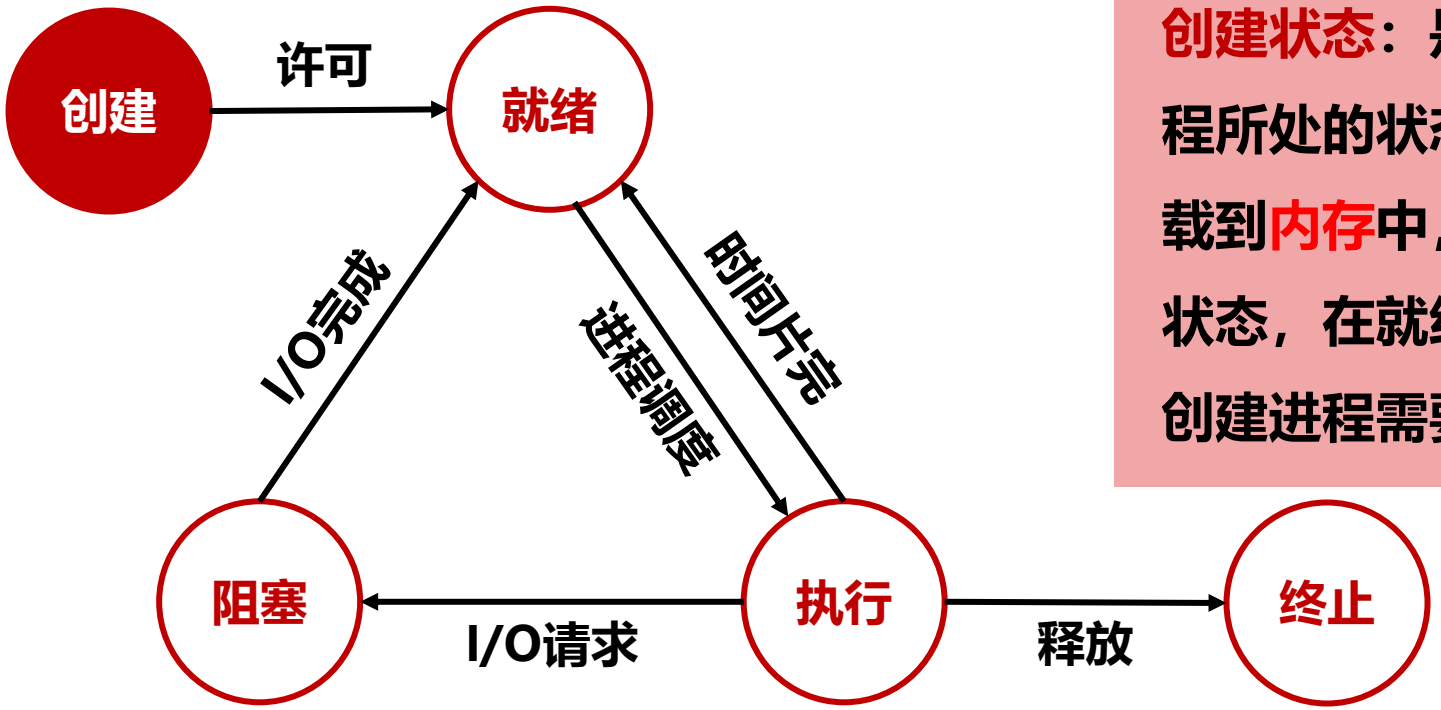
执行状态 → 就绪状态

执行状态 → 阻塞状态

阻塞状态 → 就绪状态



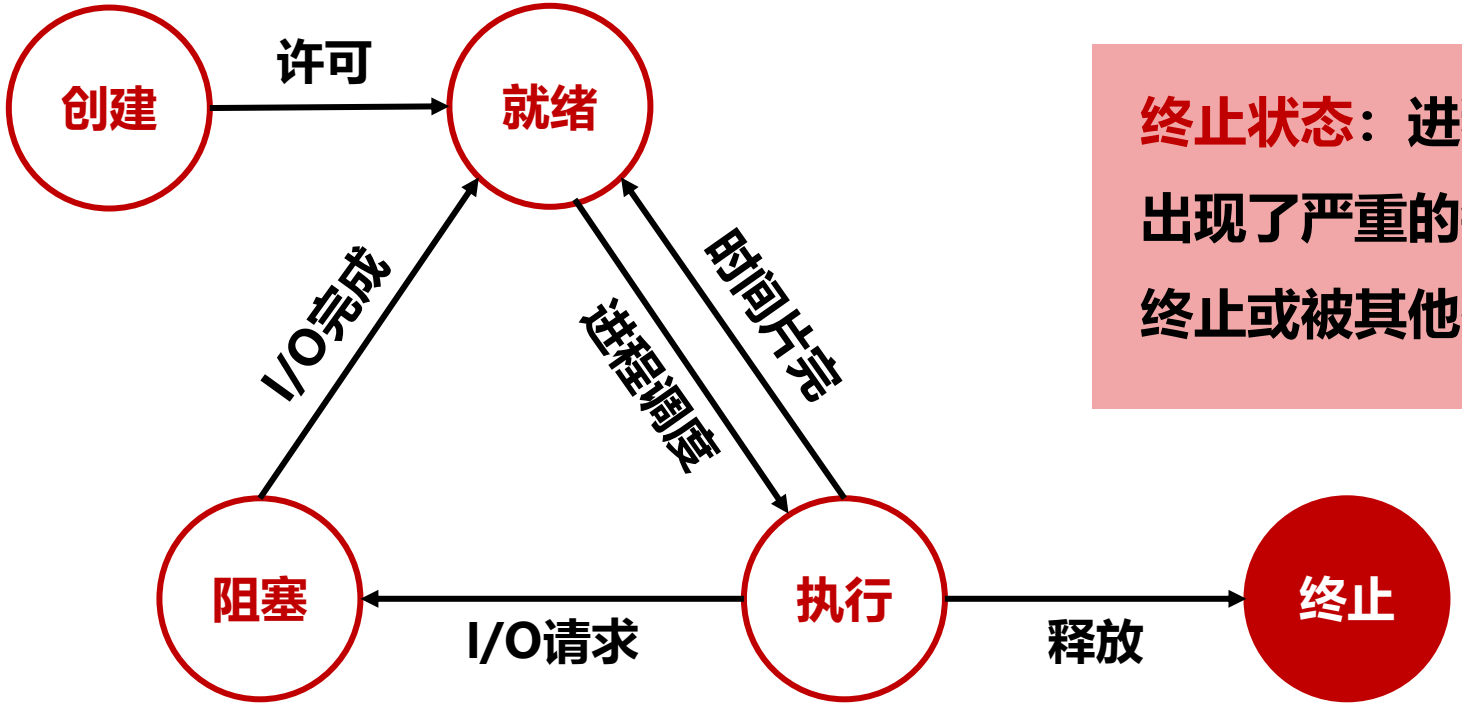
◆ 进程的两种常见状态



创建状态：是指操作系统创建进程时，进程所处的状态，相关信息（如PCB）会加载到内存中，进程新建成功后即转入就绪状态，在就绪进程队列中排队。操作系统创建进程需要为进程分配资源。

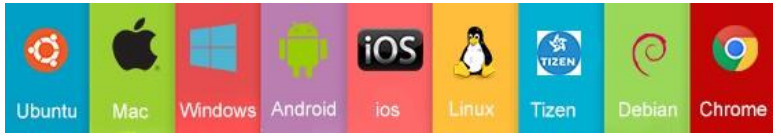
操作系统将根据系统的性能和内存容量的情况决定是否创建新的进程。如果系统性能较差或内存容量受到限制，不能为新进程分配资源，则操作系统会发出创建新进程失败的响应或将创建新进程的工作推迟

◆ 进程的两种常见状态



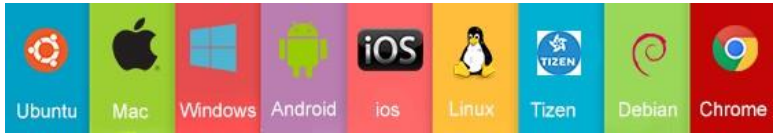
终止状态：进程达到了结束点或进程出现了严重的错误时，会被操作系统终止或被其他有终止权的进程终止。

进入终止状态的进程不再被执行，等待操作系统完成进程终止处理。操作系统完成进程终止处理后，操作系统会删除进程，收回进程所占用的资源



◆ 挂起操作的引入

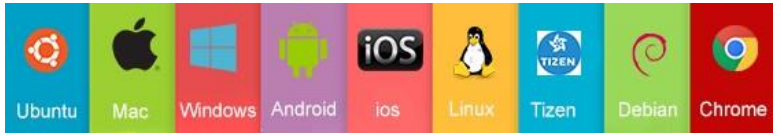
- 系统资源的需要
- 调节竞争或消除故障的需要
- 终端用户的需要
- 父进程的需要
- 调节进程的需要



◆ 挂起操作的引入

➢ 系统资源的需要

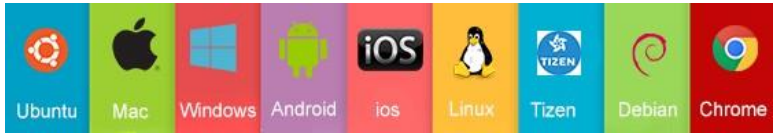
当系统中所有的进程均处于阻塞状态时，没有就绪进程，处理器处于空闲。如果这时内存空间已经被进程占满，不能装入更多的进程，需要将内存中处于阻塞状态的进程挂起，对换到外存上，让出内存空间接纳新创建的进程



◆ 挂起操作的引入

- 系统资源的需要
- 调节竞争或消除故障的需要

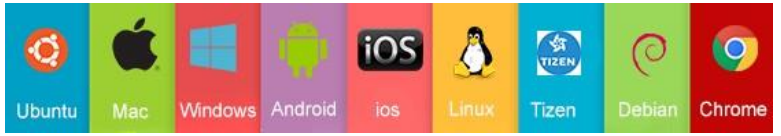
操作系统在检查进程运行中竞争资源的情况时，或在系统出现故障使某些功能受到破坏时，为了解决进程的资源竞争或消除系统故障，操作系统需要挂起某些对资源竞争的进程或怀疑引起系统故障的进程，将进程对换到外存



◆ 挂起操作的引入

- 系统资源的需要
- 调节竞争或消除故障的需要
- 终端用户的需要

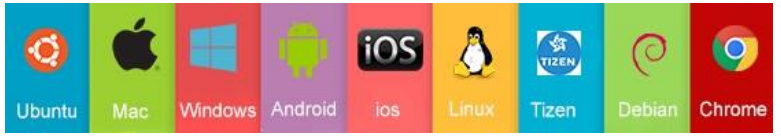
终端用户可以直接操作自己的程序。当程序员需要调试、检查和修改自己的程序时，可以要求挂起与程序相对应的进程，暂停进程的推进。



◆ 挂起操作的引入

- 系统资源的需要
- 调节竞争或消除故障的需要
- 终端用户的需要
- 父进程的需要

父进程对自己的子进程实施控制、修改和检查时，需要挂起自己的子进程，暂停子进程的推进。



◆ 挂起操作的引入

某些定期执行的进程，如系统监控进程、日志进程等，在执行时间未到而需要等待时，可以将进程挂起，对换到外存，从而减轻内存负担。当执行时间到时，再将这些进程激活换入到内存

- 调节进程的需要

◆ 引入挂起操作后三种基本状态的转换

挂起（Suspend）

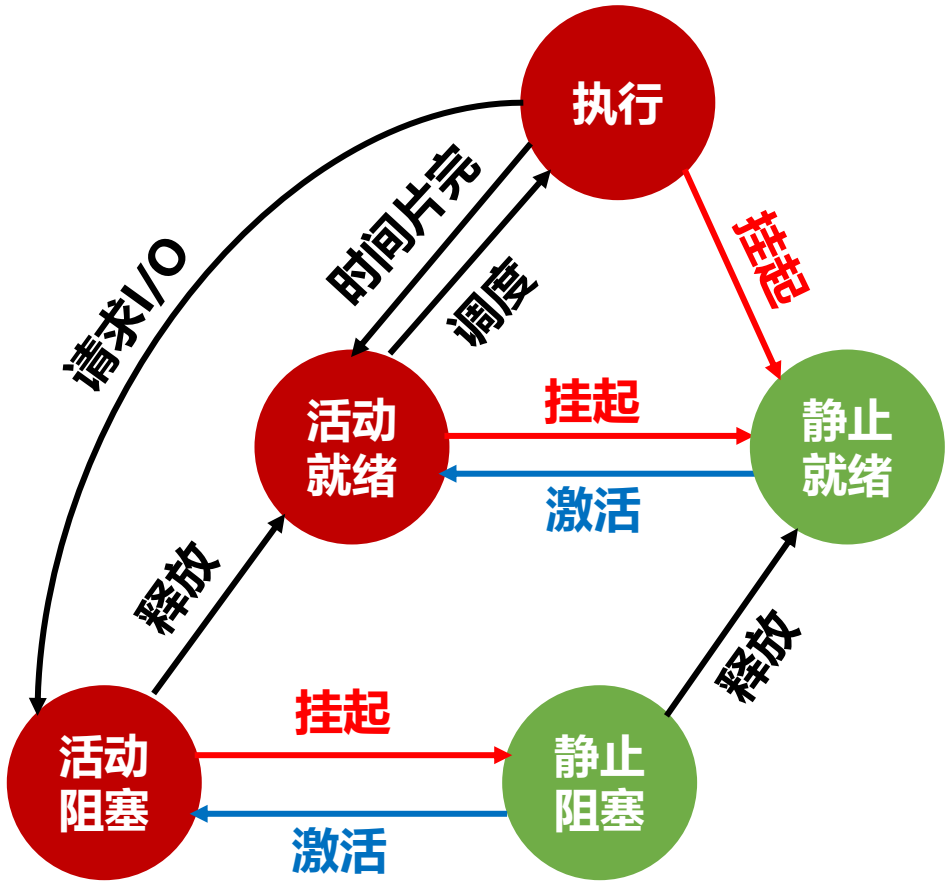
激活（Active）

活动就绪 → 静止就绪
Readya　　Readys

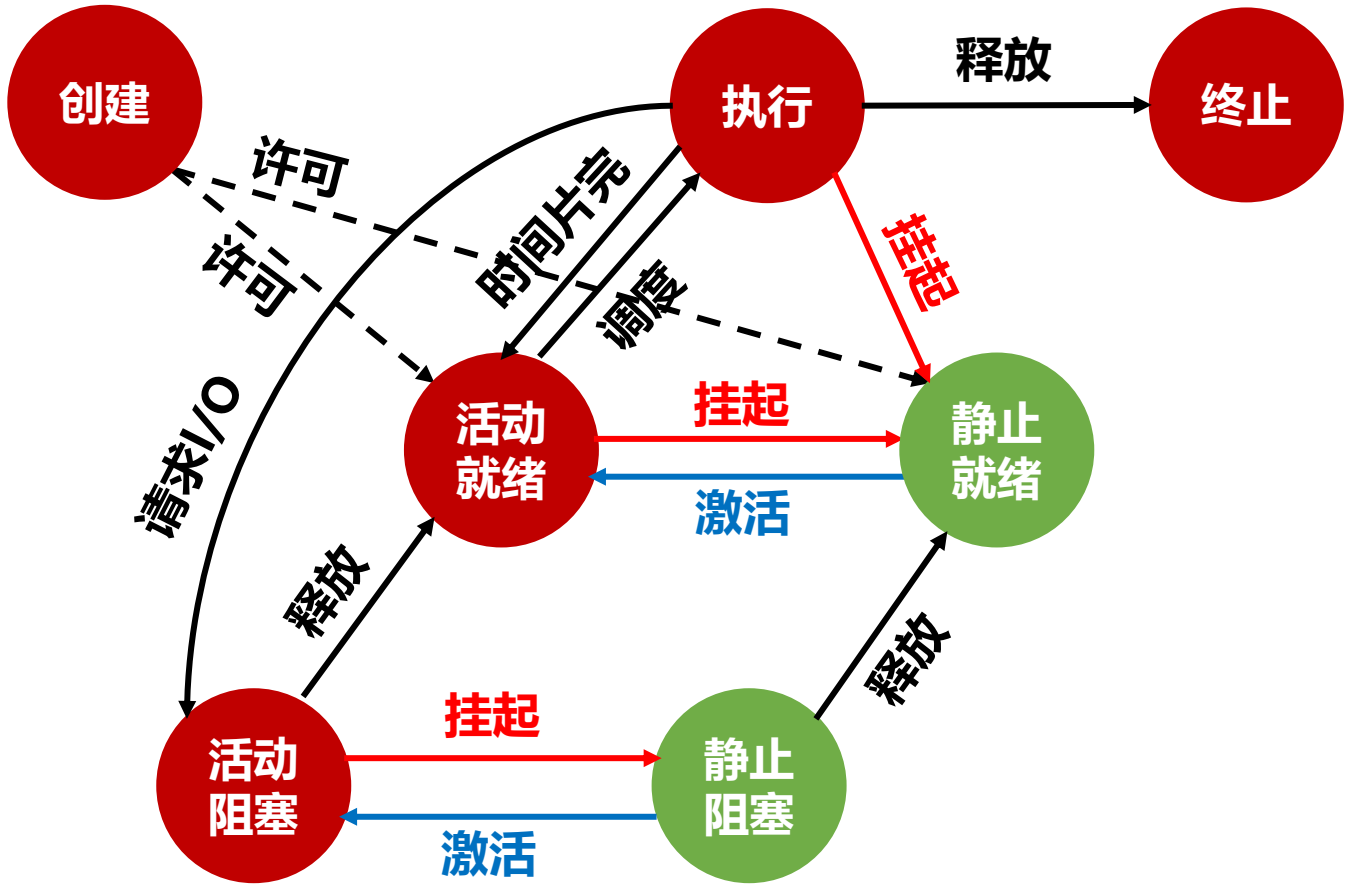
活动阻塞 → 静止阻塞
Blockeda　　Blockeds

静止就绪 → 活动就绪
Readys　　Readya

静止阻塞 → 活动阻塞
Blockeds　　Blockeda



◆ 引入挂起操作后五个进程状态的转换

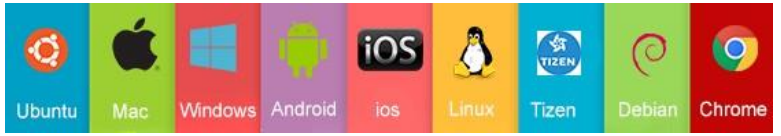


NULL → 创建

创建 → 活动就绪

创建 → 静止就绪

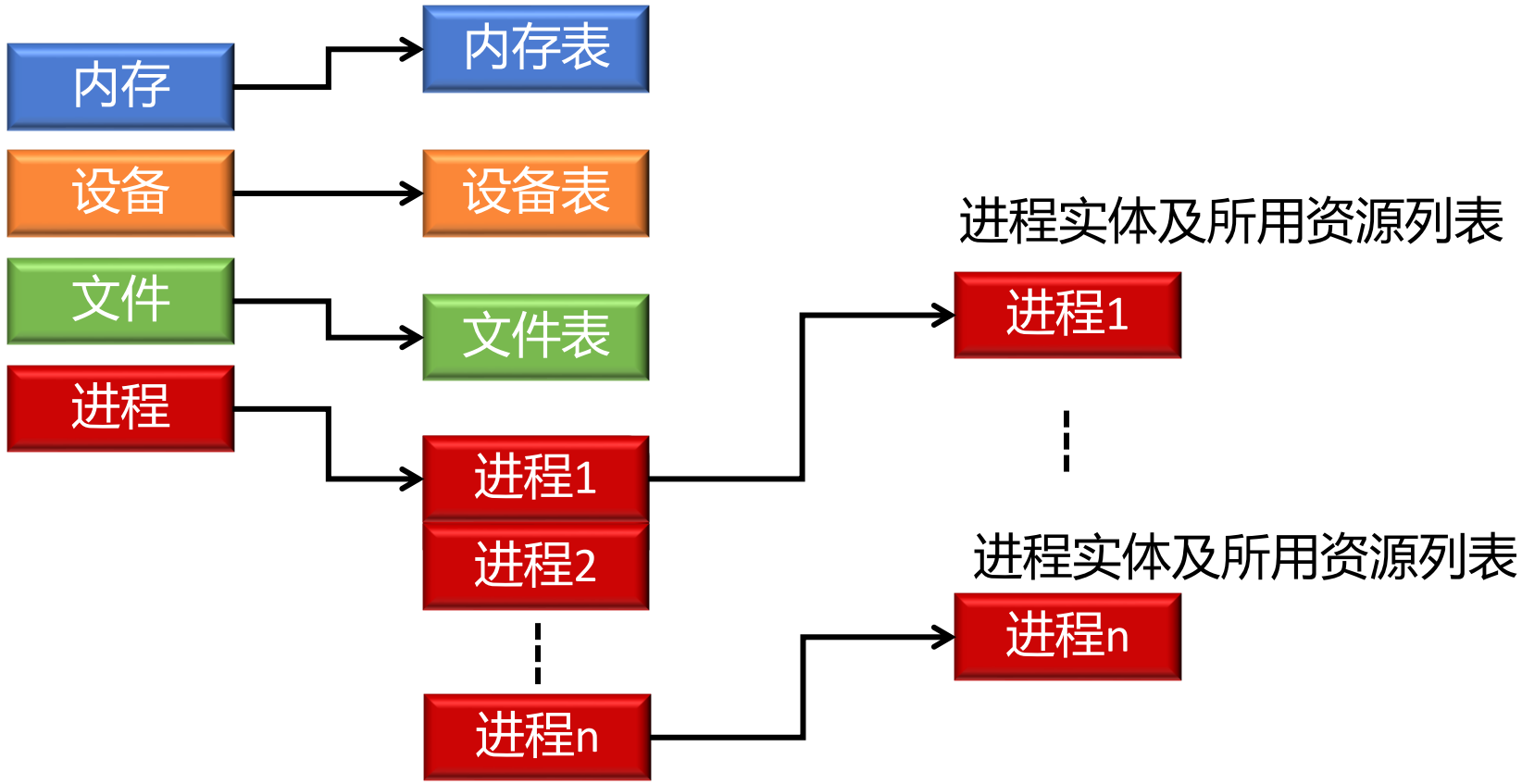
执行 → 终止

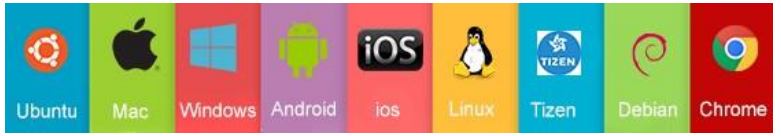


◆ OS通过各种数据结构来实现对信息的组织和维护

- OS将资源抽象为数据结构，并提供相关操作命令，用户借此执行相关操作，无需了解实现细节
- OS作为计算机资源的管理者，必须记录和查询各种资源的使用及各类进程运行情况的信息，对于这些信息的组织和维护也是通过简历和维护各种数据结构的方式实现
- 每个资源（资源信息表）和每个进程（进程信息表，又称为进程控制块PCB）都设置了一个数据结构，包含资源或进程的标识、描述、状态等信息和一批指针

◆ OS通过各种数据结构来实现对信息的组织和维护

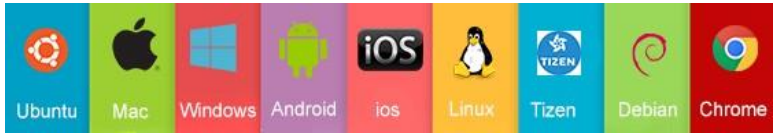




2.2.4 进程管理中的数据结构

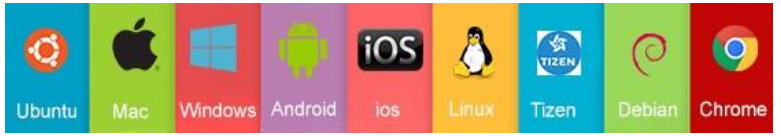
◆ 进程控制块PCB

- 对进程本质属性的描述，是操作系统管理进程所需要的基本信息
- 记录操作系统用于描述进程状况和控制进程运行所需要的基本信息
- 每个进程都有一个进程控制块，进程是动态变化的，进程控制块中的信息也是变化的，操作系统通过读或写进程控制块中的信息达到了解进程，记录进程变化的目的



◆ 进程控制块PCB的作用

- 作为独立运行基本单位的标志：系统根据PCB感知进程的存在，是进程存在于系统中的唯一标志
- 能实现间断性运行方式：当进程因阻塞而暂停运行时，保留运行时的CPU现场信息，供进程再次被调度时恢复CPU现场
- 提供进程管理所需要的信息：找到相应数据和程序，对文件或I/O设备进行访问，了解进程所需全部资源
- 提供进程调度所需要的信息：提供进程状态信息、优先级等
- 实现与其他进程的同步与通信：具有实现进程通信的区域或通信队列指针等



◆ 进程控制块PCB中的信息

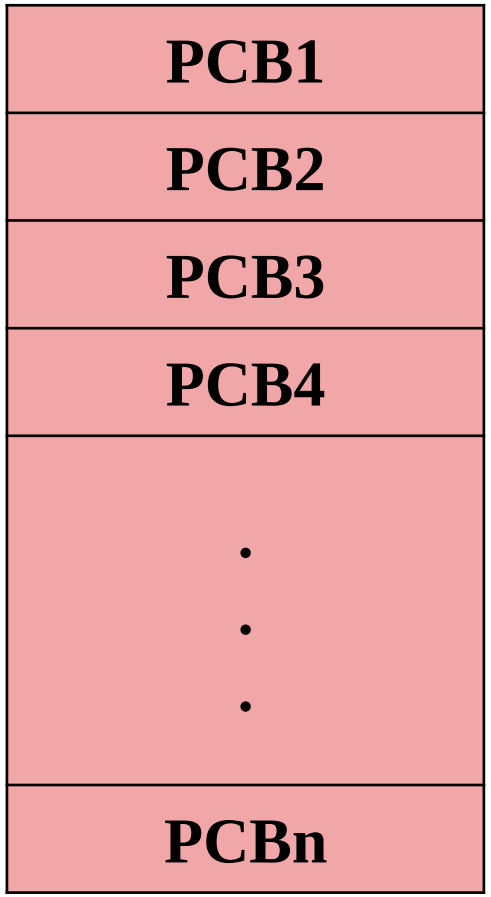
- 进程标识符（进程的唯一标识）
  - 内部标识符：用户对进程访问
  - 外部标识符：方便系统使用进程
- 处理机状态（上下文）：暂存信息的通用寄存器、存放下一条指令地址的指令计数器、程序状态字、用户栈指针
- 进程调度信息：进程状态、优先级、其他信息和事件等
- 进程控制信息：程序和数据地址、同步和通信机制、资源清单、链接指针

进程标志符	进程名
进程调度信息	进程状态
	进程优先级
进程控制信息	现场保留区
	指示出于同一状态进程的链指针
	资源清单
	进程起始地址
	家族关系
	其他

◆ 进程控制块的组织方式

➢ 线性方式

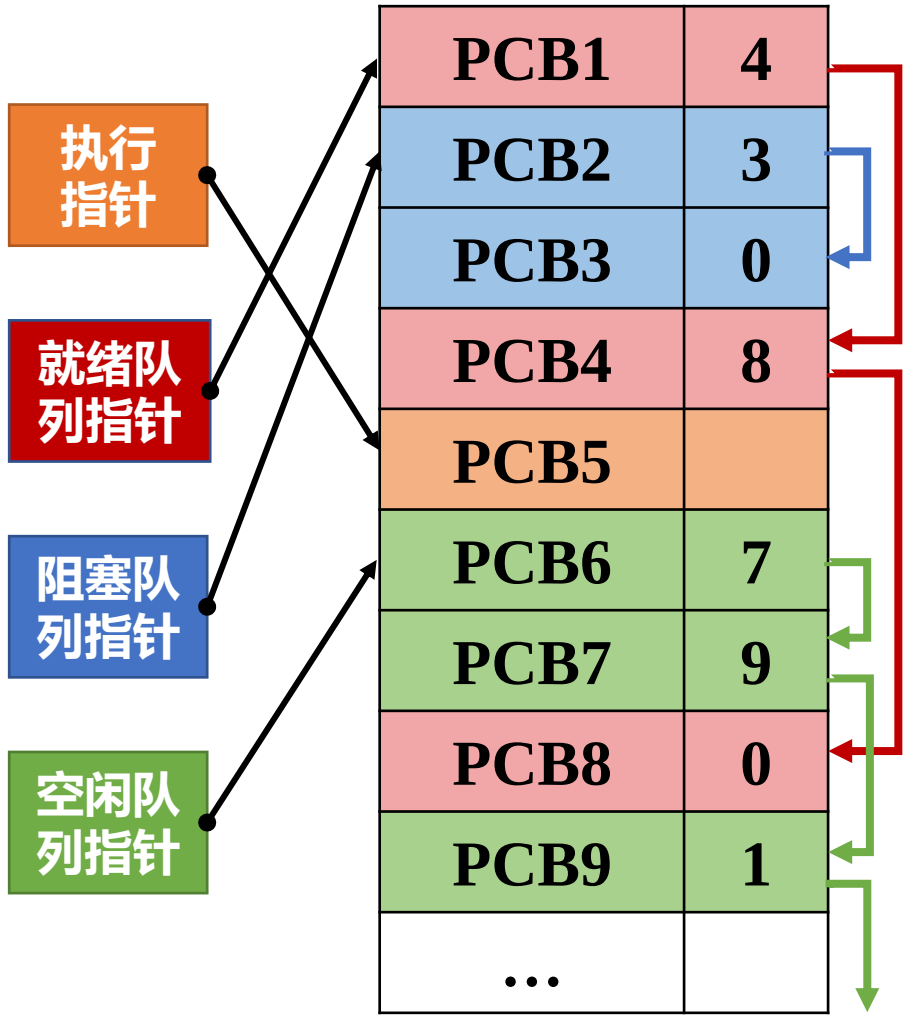
- 简单、开销小
- 须扫描整张表
- 适合进程数目少的系统



◆ 进程控制块的组织方式

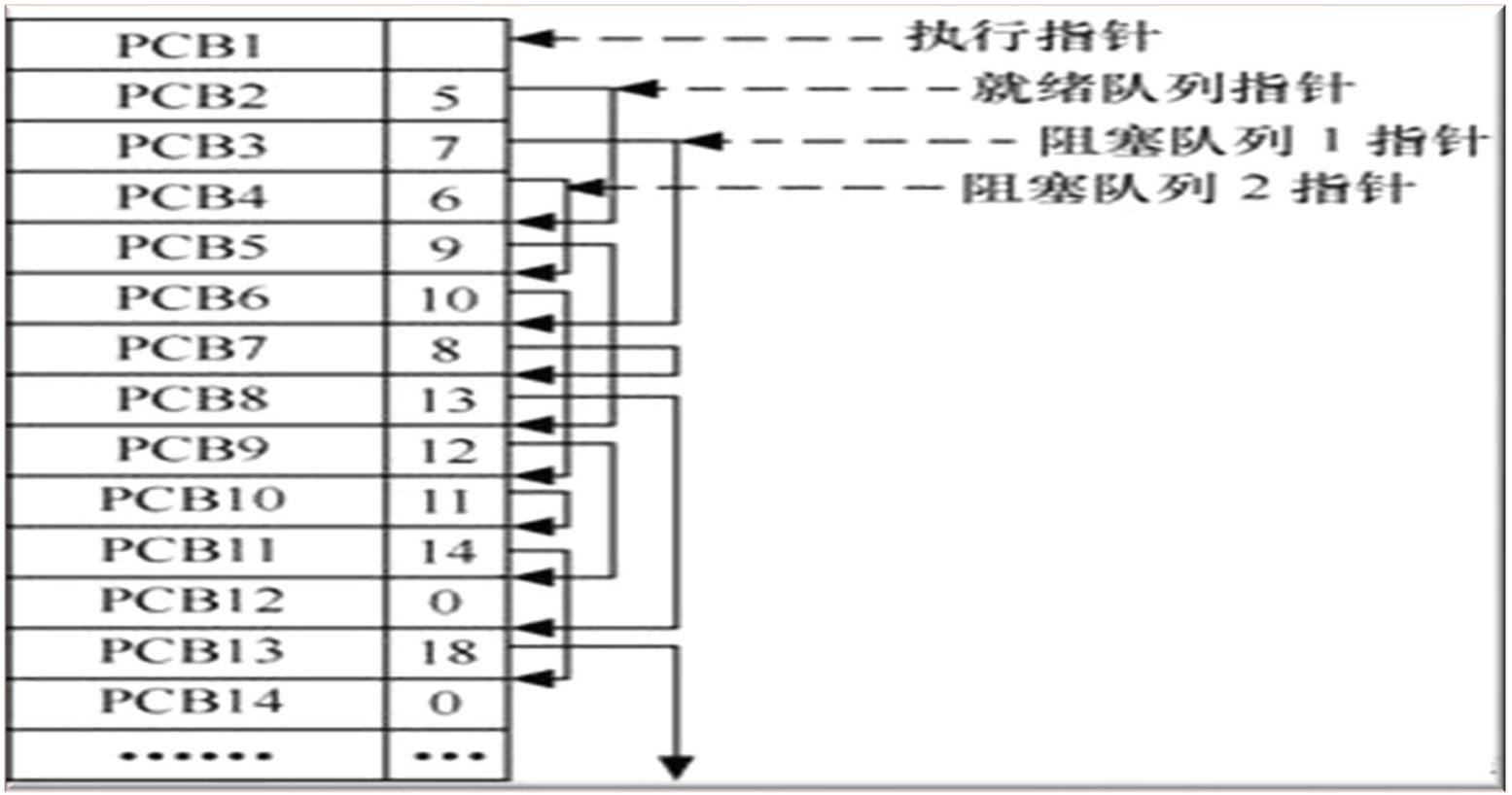
➢ 链接方式

- 具有相同状态的进程的PCB通过链接字链接成一个队列
- 按进程优先级将PCB从高到底排列



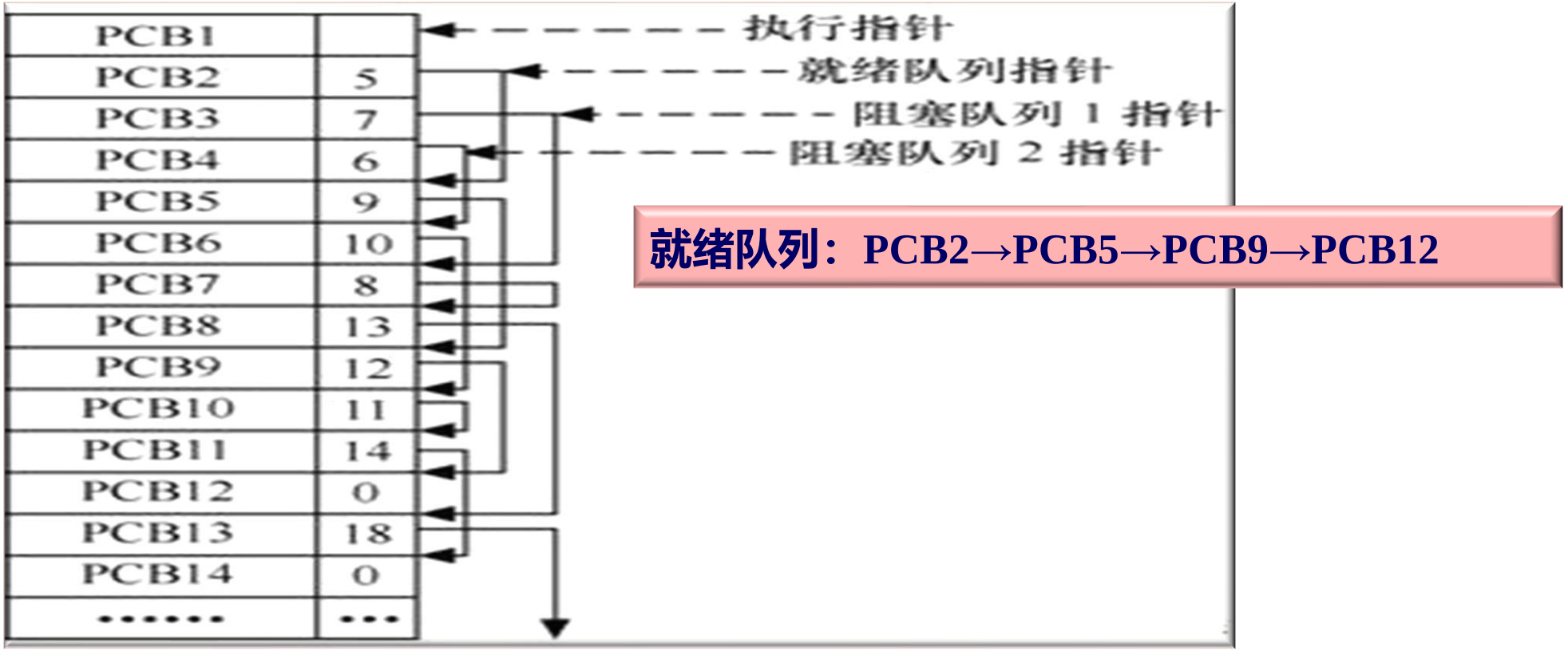
◆ 进程控制块的组织方式

➢ 链接方式



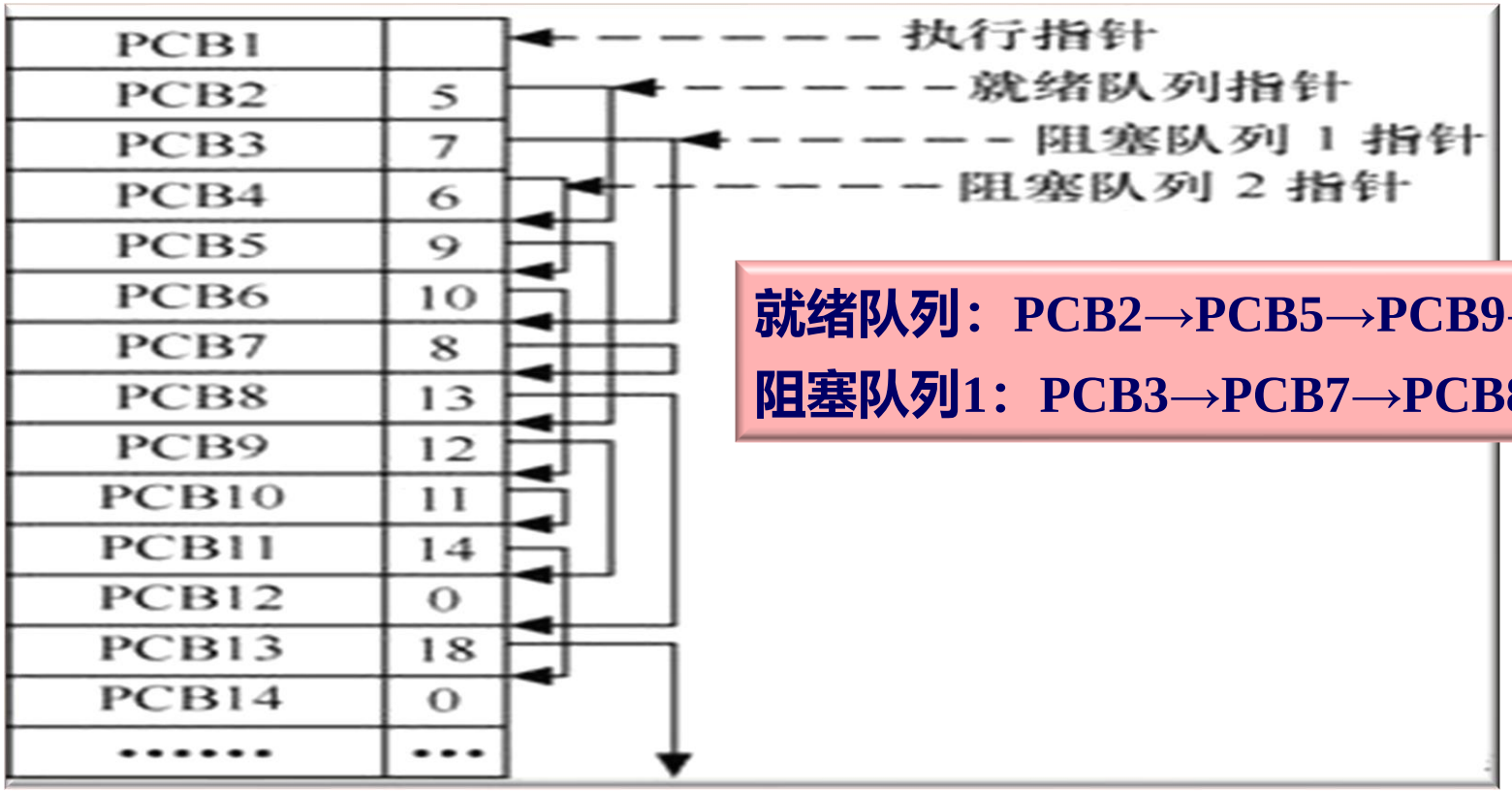
◆ 进程控制块的组织方式

➢ 链接方式



◆ 进程控制块的组织方式

➢ 链接方式

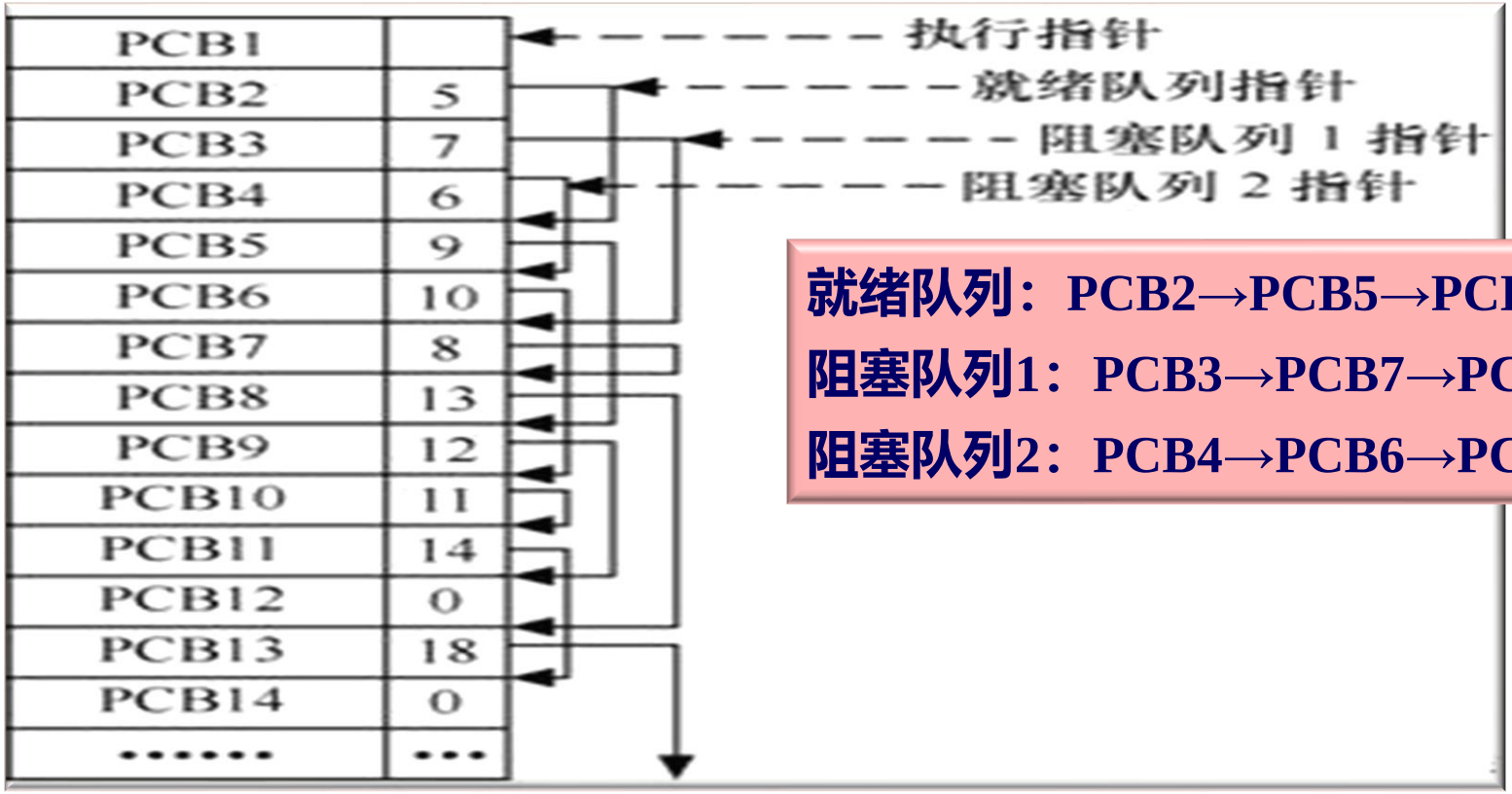


就绪队列：PCB2→PCB5→PCB9→PCB12

阻塞队列1：PCB3→PCB7→PCB8→PCB13→……

◆ 进程控制块的组织方式

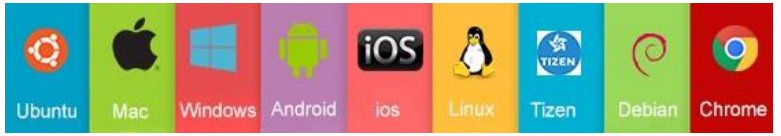
➢ 链接方式



就绪队列：PCB2→PCB5→PCB9→PCB12

阻塞队列1：PCB3→PCB7→PCB8→PCB13→…

阻塞队列2：PCB4→PCB6→PCB10→PCB11→PCB14



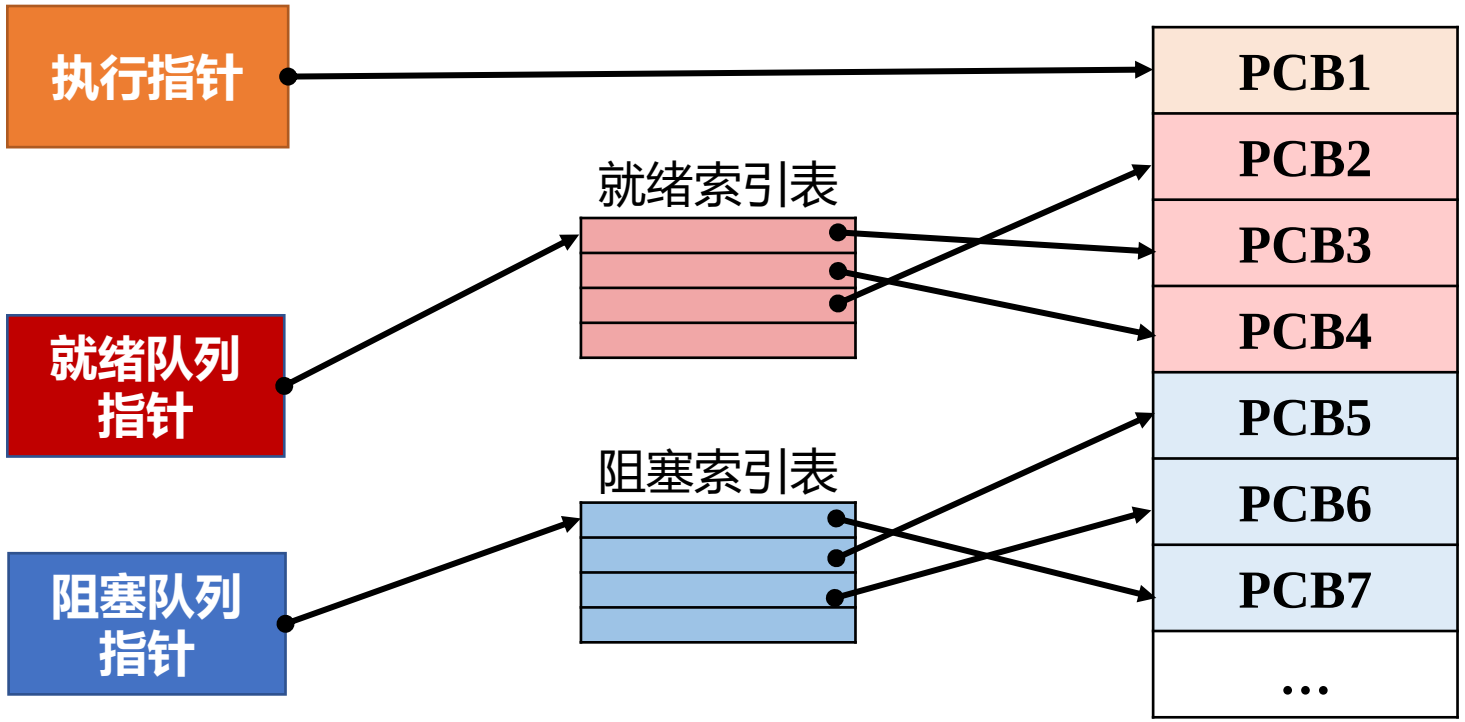
◆ 进程控制块的组织方式

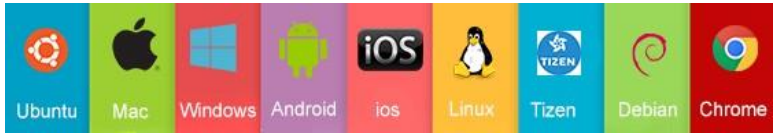
➢ 链接方式

- **优点：**直观，体现了进程的本身特性，如等待时间的长短、优先级的高低、需要处理时间的长短，为进程调度算法的实施提供了方便。
- **缺点：**以链表方式组织进程控制块的主要缺点是如果进程状态发生变化，则链接队列需要作相应的调整，进程控制块中的首部和尾部指针需要改变。

◆ 进程控制块的组织方式

➢ 索引方式：根据进程状态不同，建立索引表

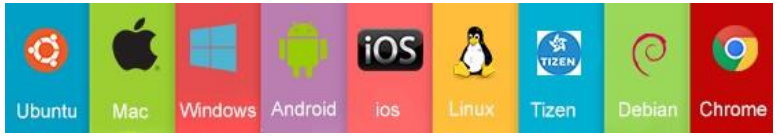




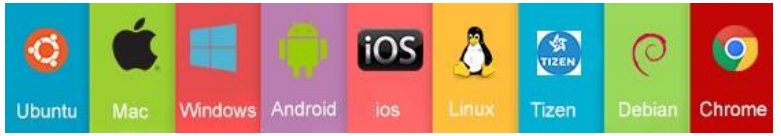
◆ 进程控制块的组织方式

➢ 索引方式

- **优点：**通过索引表可以快速得到进程控制块地址，不需要像链表方式一样，从链首到链尾查找；如果进程状态变化，不需要修改进程控制块的链接指针，只需要增加或删除索引表中的记录。
- **缺点：**索引表本身需占用内存空间；搜索索引表需要时间。



- 前趋图和程序执行
- 进程的描述
- 进程控制
- 进程同步 ★
- 经典进程的同步问题 ★
- 进程通信
- 线程的基本概念
- 线程的实现

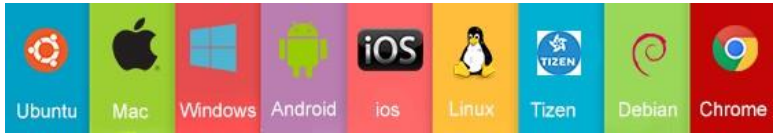


前趋图和程序执行

进程的描述

进程控制

1. 操作系统内核
2. 进程的创建
3. 进程的终止
4. 进程的阻塞与唤醒
5. 进程的挂起与激活

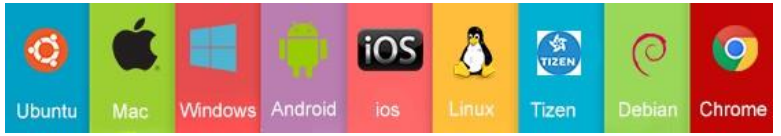


◆ 操作系统内核的定义

- 操作系统分层设计，不同功能放置在不同层次；
- 与硬件紧密结合的模块、常用设备驱动、运行频率的模块，安排在紧靠硬件的软件层次中，这些模块常驻内存，被称为OS内核
- **对软件进行保护，提高OS运行效率**

◆ 处理机的执行状态

- **系统态（管态/内核态）：** 具有较高特权，执行一切指令，访问所有寄存器和存储区；OS通常运行在管态
- **用户态（目态）：** 特权较低，仅能执行规定的指令，访问指定寄存器和存储区，应用程序只能运行在目态



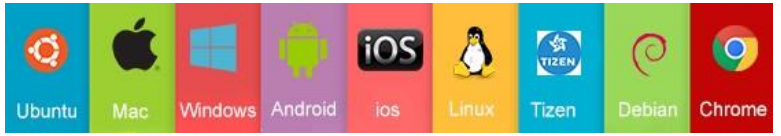
◆ 内核两大功能

➢ 支撑功能

- 中断处理：最基本的功能，OS赖以活动的基础
- 时钟管理：基本功能，时间片轮转调度、实时系统等
- 原语（Primitive）操作：若干条指令组成，执行过程中不允许被中断

➢ 资源管理功能

- 进程管理：运行频率较高，提高性能
- 存储器管理：运行频率较高，提高运行速度
- 设备管理：与硬件/设备紧密相关



◆ 创建步骤（creat原语）

命名进程：为进程设置进程标志符；

从PCB集合中为新进程申请一个空白PCB；

确定进程的优先级；

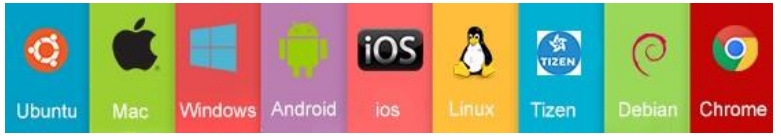
为进程的程序段、数据段和用户栈分配内存空间；如果进程中需要共享某个已在内存的程序段，则必须建立共享程序段的链接指针；

为进程分配除内存外的其他各种资源；

初始化PCB，将进程的初始化信息写入PCB；

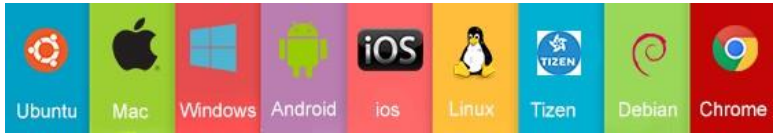
如果就绪队列能够接纳新创建的进程，则将新进程插入到就绪队列；

通知OS的其他管理模块，如记账程序、性能监控程序等。



◆ 引起创建进程的事件（原因）

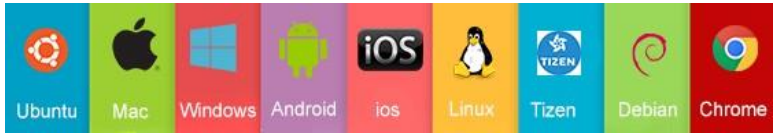
- ➢ 操作系统初始化：
- ➢ 提供用户服务
- ➢ 分时系统用户登录
- ➢ 用户请求系统创建新进程
- ➢ 执行创建新进程的系统调用
- ➢ 批处理作业的初始化和调度



◆ 引起创建进程的事件（原因）

➢ 操作系统初始化：

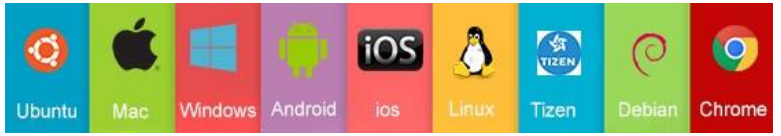
当操作系统启动时，通常会创建若干进程，特别是一些常驻系统的进程。这些进程有的运行在系统的前台，供用户交互；有的运行在后台，完成某些专门的功能。如网络服务器中的电子邮件服务进程，如果没有邮件需要处理，则进入休眠状态；如果邮件到来，则该进程被唤醒，处理接收到的邮件。



◆ 引起创建进程的事件（原因）

- ➢ 操作系统初始化：
- ➢ 提供用户服务

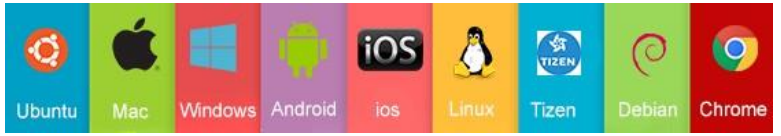
当运行中的程序需要某种服务时，系统专门创建一个进程来提供所需要的服务，如打印文件、屏幕输出等。



◆ 引起创建进程的事件（原因）

- ➢ 操作系统初始化：
- ➢ 提供用户服务
- ➢ 分时系统用户登录

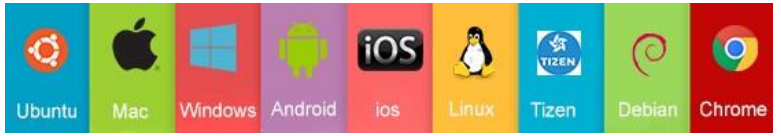
当用户在终端键入登录命令后，系统将为该终端用户建立一个进程。



◆ 引起创建进程的事件（原因）

- ➢ 操作系统初始化：
- ➢ 提供用户服务
- ➢ 分时系统用户登录
- ➢ 用户请求系统创建新进程

在交互式系统中，用户用键盘键入命令或用鼠标点击，则可以创建新进程。如在Windows操作系统中，用户可以用键盘运行多个程序，或用鼠标打开多个窗口。每个程序或每个窗口对应一个新的进程。



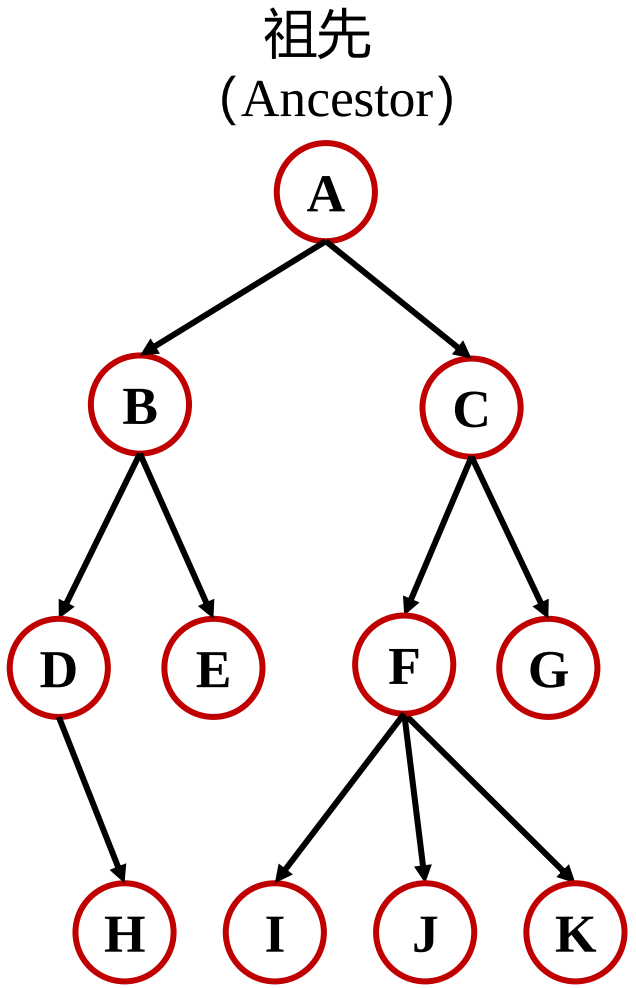
◆ 引起创建进程的事件（原因）

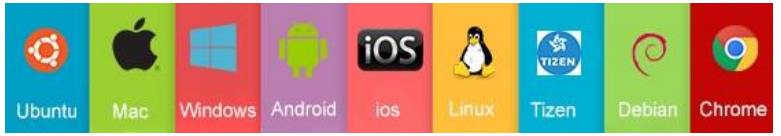
用户提交批处理作业，当操作系统能够提供资源时，作业调度程序按照一定的算法，从批处理作业清单中调度某个作业并装入内存，为作业分配必要的资源，创建作业进程

- **批处理作业的初始化和调度**

➢ 进程的层次结构

- 允许一个进程创建另一个进程，创建进程的进程为父进程Parent Process，被创建的进程为子进程Progeny Process，子进程可继续创建孙进程，形成树形结构的进程家族（组）
- 子进程可以继承父进程的所有资源
- 子进程撤销时，向父进程归还资源
- 父进程撤销时，所有子进程也被撤销
- 进程图描述进程间关系的一颗有向树





◆ 引起进程终止的事件（原因）

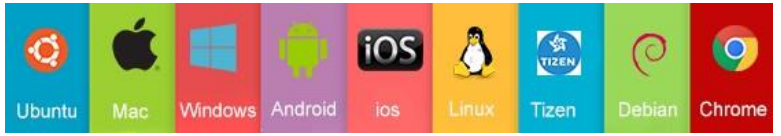
➢ 进程正常结束：halt指令或log off

➢ 异常结束

- 操作异常退出：保护错（写只读文件）、算术运算错、非法指令、特权指令错、I/O故障
- 时间指标超限引起进程异常结束：运行超时、等待超时
- 多个进程之间竞争资源
- 内存的使用出错：越界错

➢ 外界干预

- 父进程结束
- 父进程请求
- 操作系统终止：系统死锁



◆ 进程的终止过程

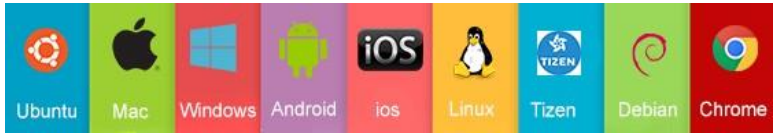
第一步：根据被终止进程的标识符，从PCB集合中查找对应进程控制块并读出该进程的状态；

第二步：若被终止进程正处于执行状态，则终止该进程的执行，并设置调度标志为真，用于指示该进程被终止后应重新进行调度，选择一新进程，把处理机分配给它。

第三步：若进程还有子孙进程，应将其所有子孙进程终止，以防它们成为不可控制的。

第四步：将进程所占有的全部资源释放（还给父进程或系统），释放进程控制块（若该进成为执行态，要进行进程调度）。

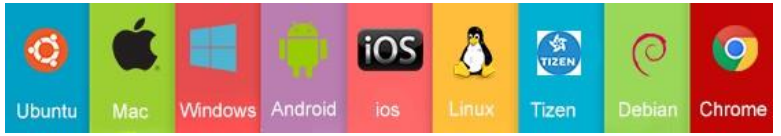
第五步：将被终止进程（它的PCB）从所在队列（或链表）中移出，等待其他程序来收集相关信息。



◆ 引起进程阻塞和唤醒的事件

- ➢ 向系统请求共享资源失败
- ➢ 等待某种操作的完成
- ➢ 新数据尚未到达
- ➢ 等待新任务的到达

- ➢ **进程阻塞过程：Block**
  - 停止进程在处理器中执行，保存现场信息到进程的进程控制块中；
  - 修改进程控制块中的进程状态等相关内容，并将进程控制块加入到相应阻塞进程的进程控制块队列中；
  - 进程调度程序转入其他进程调度。
- ➢ **进程唤醒过程：Wakeup**
  - 当等待事件到达或输入/输出操作完成时，会产生一个中断，激活操作系统，阻塞的进程被唤醒，其进程状态由阻塞转换为就绪
  - 唤醒进程的主要工作是修改进程控制块中的状态，将进程控制块转入到就绪进程的进程控制块队列

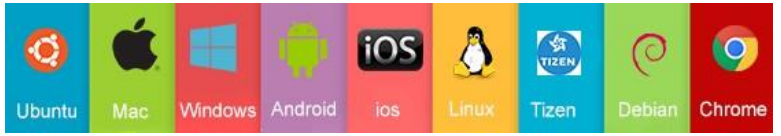


◆ 进程的挂起

- 处于阻塞状态、就绪状态和运行状态的进程都可以被挂起，根据进程挂起前的状态决定挂起后的状态。
- 如果进程被挂起，进程的PCB中的信息要修改，进程上下文被放到外存。

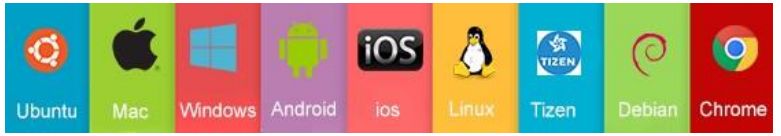
◆ 进程的激活

- 如果进程挂起时间到或者内存资源充足时，系统或有关进程，特别是进程同步中的原语操作，会激活被挂起的进程。
- 激活进程的主要工作是将进程上下文从外存调入内存，修改PCB信息并调入内存，修改PCB中的进程状态，并按照进程状态将PCB排入到相应的进程队列。



进程的描述和控制

- 前趋图和程序执行
- 进程的描述
- 进程控制
- 进程同步 ★
- 经典进程的同步问题 ★
- 进程通信
- 线程的基本概念
- 线程的实现



- 进程
- 多道程序并发执行
- 资源利用率
- 系统吞吐量
- 系统更加复杂

进程同步的基本概念

多道程序系统
进程同步机制

- 进程运行管理
- 对系统资源的无序争夺
- 系统混乱
- 结果不确定性
- 结果不可再现性

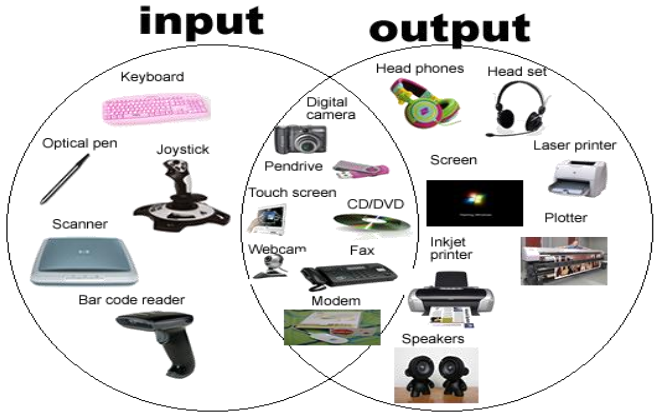
硬件同步机制

信号量机制

管程机制

任务：对多个相关进程在执行次序上进行协调，使并发执行的诸进程之间按照一定的规则(或时序)共享系统资源，并能很好地相互合作，从而使程序的执行具有可再现性

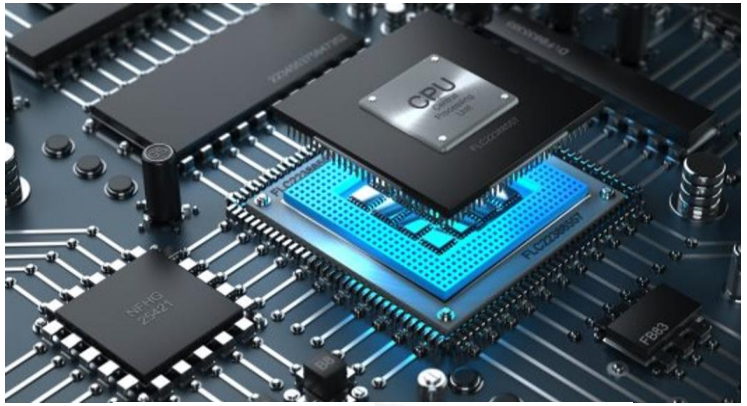
2.4.1 进程同步的基本概念



◆ 两种形式的制约关系

➢ 间接相互制约关系

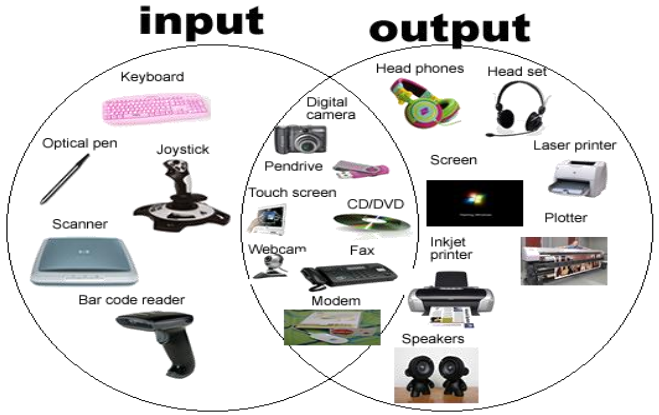
- 多个程序在并发执行时，对**共享系统资源**及**临界资源**访问时形成间接相互制约关系。
- 对于**互斥共享资源**，必须由系统实施统一分配，使用前，应提出申请，不允许用户进程直接使用。



➢ 直接相互制约关系

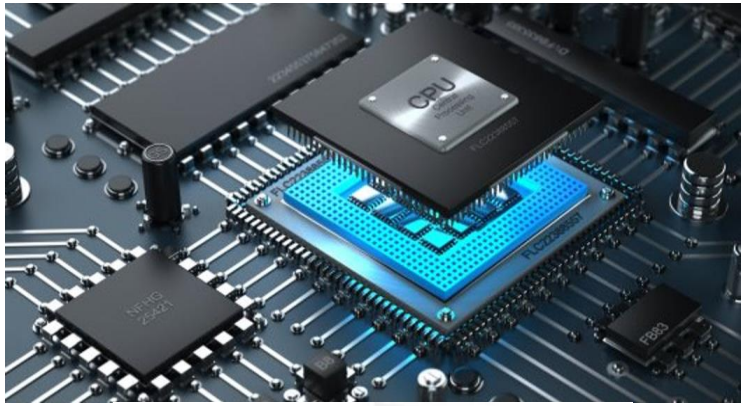
- 为完成**同一任务而相互合作**的进程（如输入进程和计算进程共享缓冲区）

2.4.1 进程同步的基本概念



◆ 两种形式的制约关系

- **异步性：** 进程在运行过程中是否能获得处理机运行与以怎样的速度运行，并不能由进程自身所控制
- **与时间有关的错误：** 对共享变量或数据结构等资源不正确的访问次序，造成进程每次执行结果的不一致，这种差错往往与时间有关
- 为了杜绝这种差错，必须对进程的执行次序进行协调，保证诸进程能按序执行。



2.4.1 进程同步的基本概念

◆ 进程同步和互斥

- ➢ 进程间的相互关系主要有三种：同步、互斥、通信
- ➢ 进程的同步关系
  - 若干合作进程为了完成一个共同的任务，需要相互协调运行进度，一个进程开始某个操作之前，必须要求另一个进程已经完成某个操作，否则前面的进程只能等待。
- ➢ 进程的互斥关系：和资源共享相关
  - 多个进程由于共享了独占性资源，必须协调各进程对资源的存取顺序，确保没有任何两个或以上的进程同时进行存取操作
- ➢ 互斥关系也属于同步关系，是一种特殊的同步。

2.4.1 进程同步的基本概念

◆ 进程同步和互斥

- ➢ 进程间的相互关系主要有三种：同步、互斥、通信
- ➢ 进程的同步关系
  - 例如学生课程学习考试包括四个功能：教师出卷、教师改卷、学生考试、学生订正。
  - 正确流程为：教师出卷→学生考试→教师改卷→学生订正。
  - 同步关系要求学生考试前，必须完成教师出卷，否则等待教师改卷前，必须完成学生考试，否则等待学生订正前，必须完成教师改卷，否则等待。

2.4.1 进程同步的基本概念

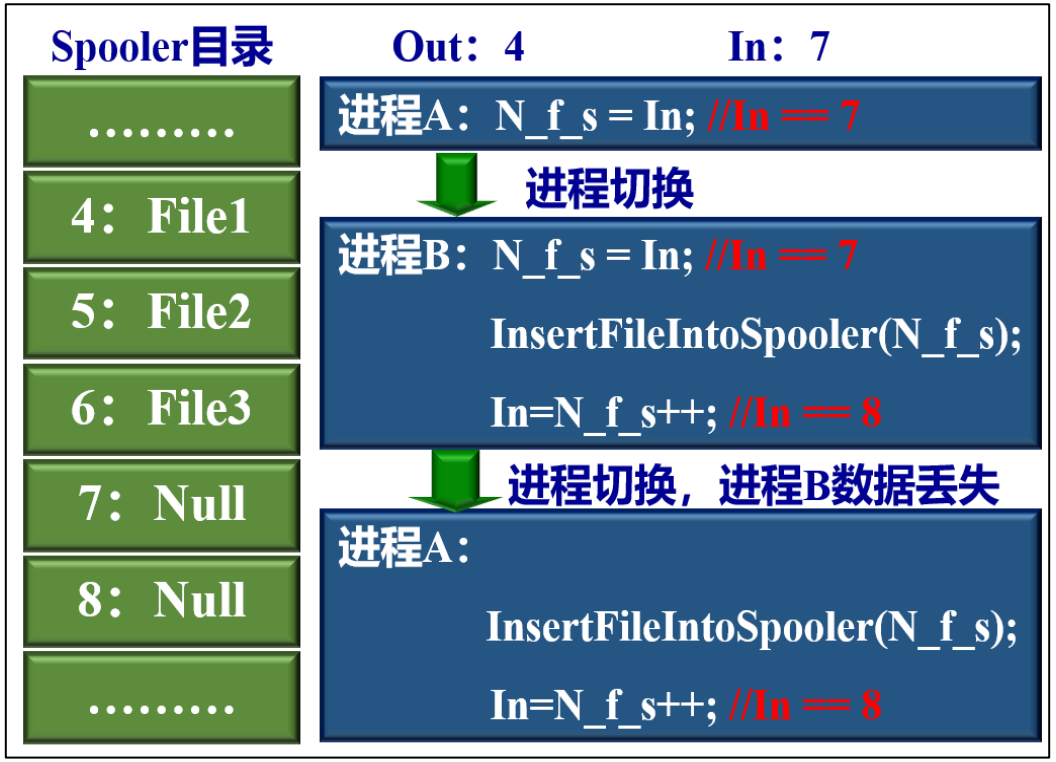
- **进程同步和互斥**
  - 进程间的相互关系主要有三种：同步、互斥、通信
  - 进程的互斥关系

程序A:	程序B:
…	…
i=100;	i=200;
…	…
print(i)	print(i)
…	…

2.4.1 进程同步的基本概念

◆ 进程同步和互斥

进程的同步

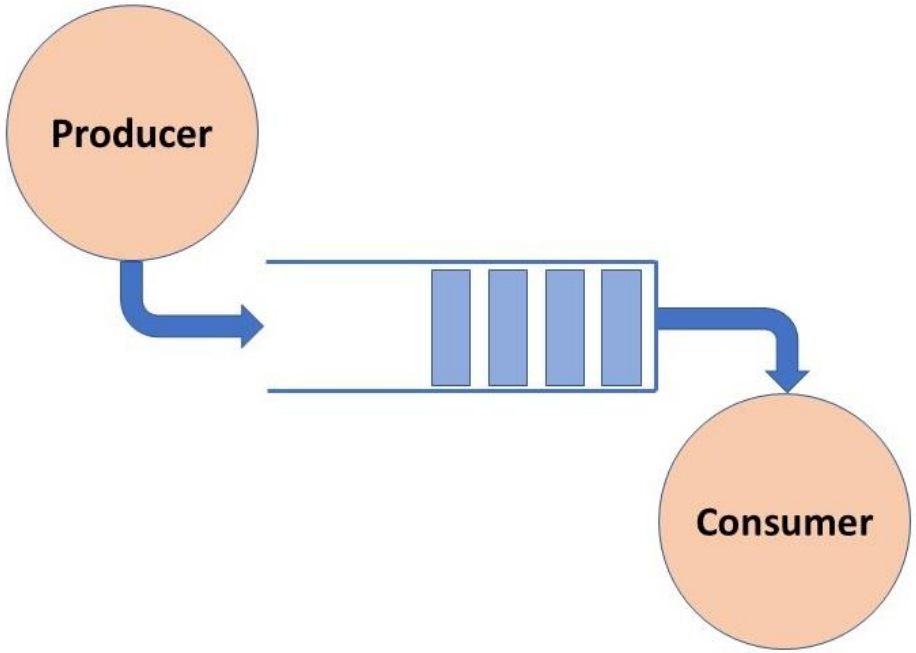


进程的互斥

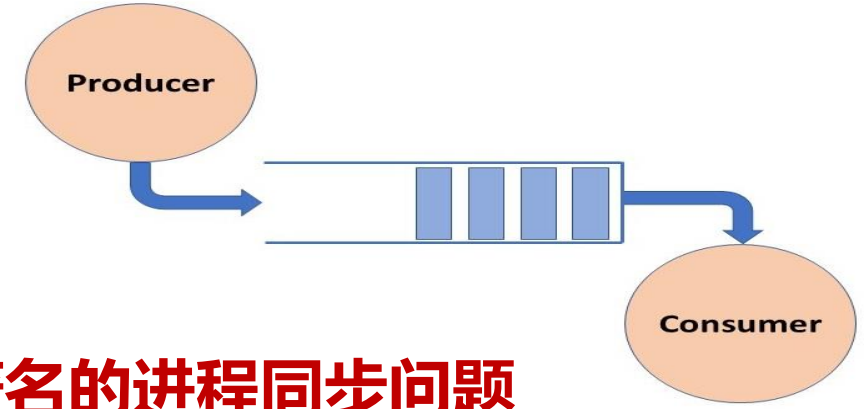
2.4.1 进程同步的基本概念

◆ 临界资源（Critical Resource）

➢ 进程采取互斥方式实现对资源的共享



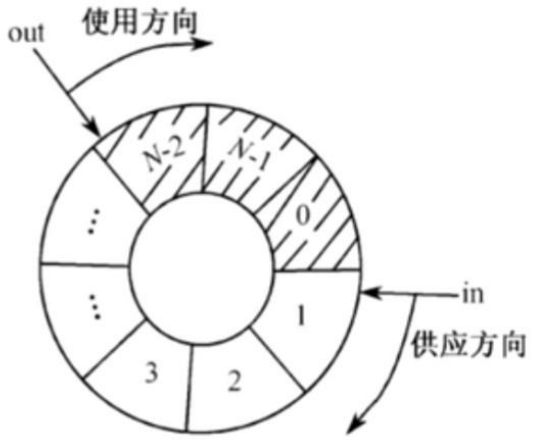
2.4.1 进程同步的基本概念

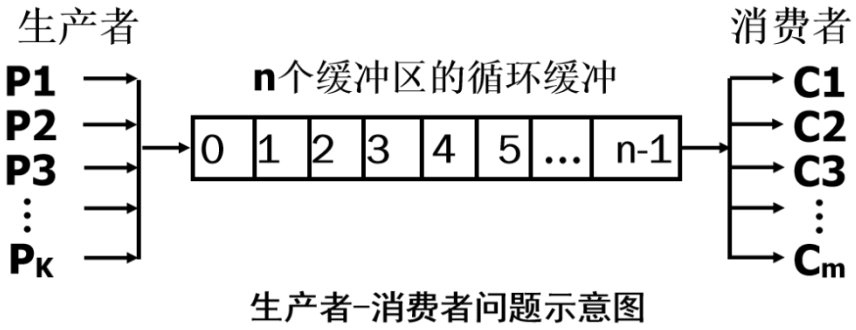


◆ 临界资源（Critical Resource）

➢ 生产者与消费者（producer-consumer）问题——著名的进程同步问题

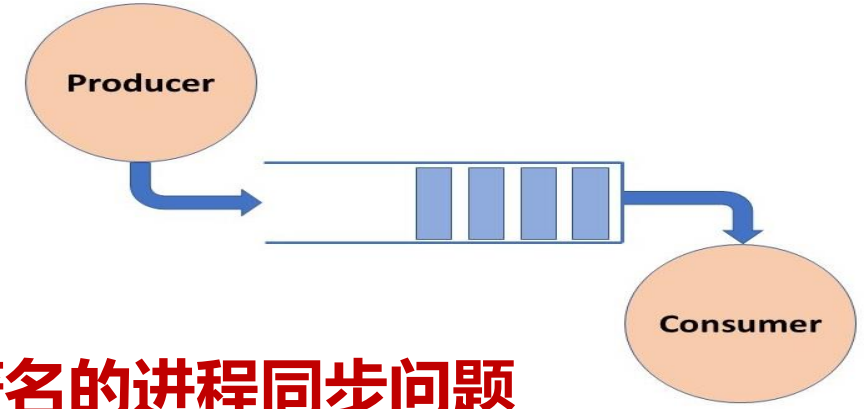
- ➢ 缓冲池：数组表示，具有n个(0，1，…，n-1)缓冲区
- ➢ 输入指针in：指示下一个可投放产品的缓冲区
- ➢ 输出指针out：指示下一个可从中获取产品的缓冲区
- ➢ 缓冲池采用循环组织，故：
  - 输入加1表示成 in:= (in+1) % n；
  - 输出加1表示成out:= (out+1) % n；
  - (in+1) % n=out时表示缓冲池满；
  - in=out则表示缓冲池空。
- ➢ 整型变量counter：
  - 生产者投放产品counter加1；
  - 消费者取走产品counter减1。





生产者-消费者问题示意图

2.4.1 进程同步的基本概念



◆ 临界资源（Critical Resource）

➢ 生产者与消费者（producer-consumer）问题——著名的进程同步问题

生产者进程

```
producer: repeat
    produce an item in nextp;
    while counter=n do no-op;
    buffer［in］:=nextp;
    in:=in+1 mod n;
    counter:=counter+1;
until false;
```

异步运行

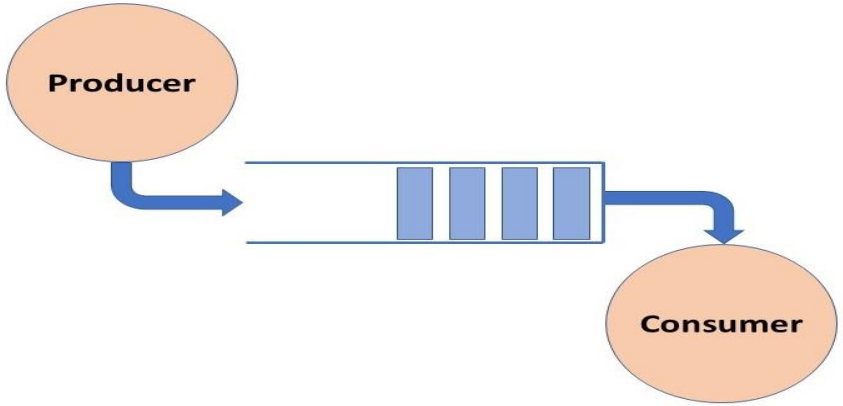
```
int in = 0;
int out = 0;
int count = 0;
item buffer[n];
```

保持同步

消费者进程

```
consumer: repeat
    while counter=0 do no-op;
    nextc:=buffer［out］;
    out:=(out+1) mod n;
    counter:=counter-1;
    consumer the item in nextc;
until false;
```

2.4.1 进程同步的基本概念



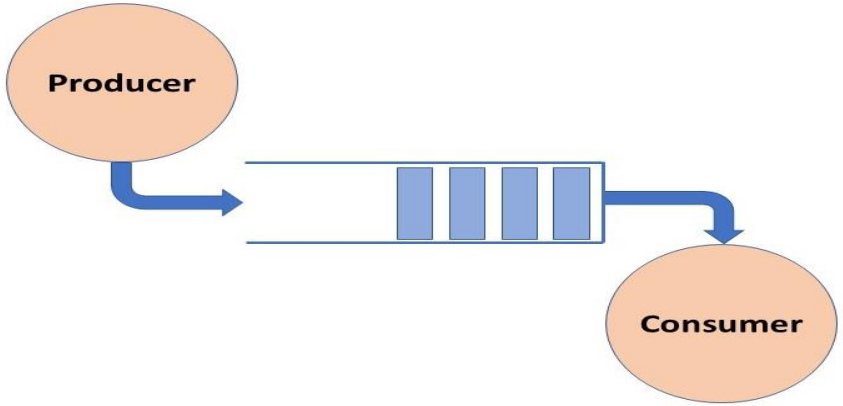
◆ 临界资源（Critical Resource）

➢ 问题的出现

```
register1:=counter;
register1:=register1+1;
counter:=register1;
```

```
register2:=counter;
register2:=register2-1;
counter:=register2;
```

2.4.1 进程同步的基本概念



◆ 临界资源（Critical Resource）

➢ 问题的出现

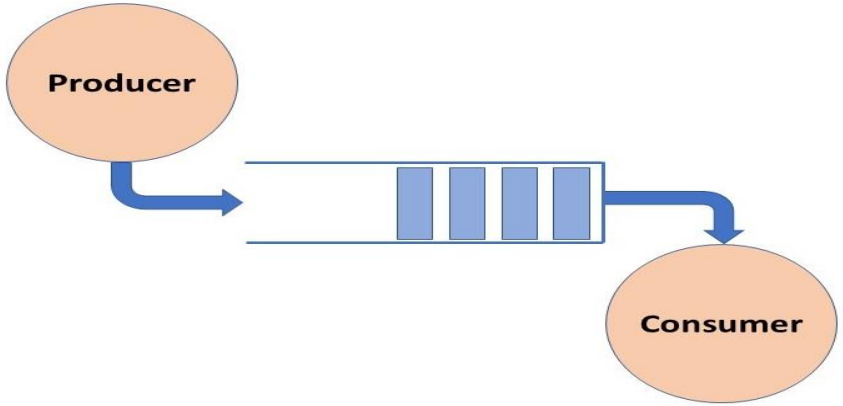
```
register1:=counter;
register1:=register1+1;
counter:=register1;
```

```
register2:=counter;
register2:=register2-1;
counter:=register2;
```

设counter=5:

register1:=counter;	(register1=5)
register1:=register1+1;	(register1=6)
counter:=register1;	(counter=6)
register2:=counter;	(register2=6)
register2:=register2-1;	(register2=5)
counter:=register2;	(counter=5)

2.4.1 进程同步的基本概念



◆ 临界资源（Critical Resource）

➢ 问题的出现

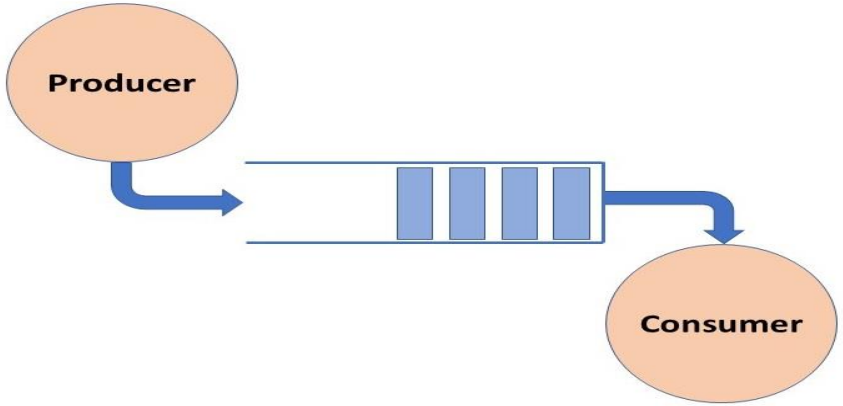
```
register1:=counter;
register1:=register1+1;
counter:=register1;
```

```
register2:=counter;
register2:=register2-1;
counter:=register2;
```

设counter=5:

register1:=counter;	(register1=5)
register1:=register1+1;	(register1=6)
register2:=counter;	(register2=5)
register2:=register2-1;	(register2=4)
counter:=register1;	(counter=6)
counter:=register2;	(counter=4)

2.4.1 进程同步的基本概念



◆ 临界资源（Critical Resource）

➢ 问题的出现

```
register1:=counter;
register1:=register1+1;
counter:=register1;
```

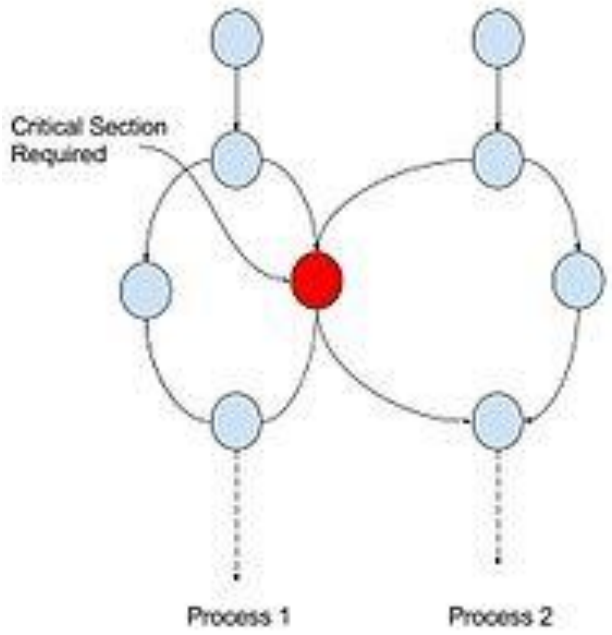
```
register2:=counter;
register2:=register2-1;
counter:=register2;
```

程序的执行失去了再现性，为预防这种错误，应把变量counter作为临界资源处理，即令生产者进程和消费者进程互斥地访问变量counter

2.4.1 进程同步的基本概念

◆ 临界区Critical Section

- 不论是硬件临界资源还是软件临界资源，多个进程必须互斥地对它进行访问
- **临界区(critical section)：在每个进程中访问临界资源的那段代码，即存取操作区域**
- 进程访问临界资源前都要判断该资源能否访问：
  - 如果能访问，进入到临界区访问临界资源；
  - 如果不能访问，则等待直到该资源能够访问；
  - 访问结束后，需要将资源归还，使其他进程能够知道临界资源已经被当前进程访问结束。



2.4.1 进程同步的基本概念

◆ 临界区Critical Section

- **临界区：**每个进程中访问临界资源的那段代码
- **进入区：**检查临界资源是否正在被访问的代码（**判别能否访问临界资源的关键**）
- **退出区：**将临界区正被访问的标志恢复为未被访问的标志
- **剩余区：**除进入区、临界区、退出区之外的代码

```
While ( Ture )
{
        entry_section;          //申请进入（进入区）
        critical_section;       //临界区
        exit_section;           //声明退出（退出区）
        remainder_section;      //剩余区
}
```

2.4.1 进程同步的基本概念

◆ 临界区Critical Section进入准则

- **单个入区：**一次仅允许一个进程进入。
- **独自占用：**处于临界区内的进程不可多于一个。如果已有一个进入临界区，其它试图进入的进程必须等待。
- **尽快退出：**访问完后尽快退出，以让出资源。
- **落败让权：**如果进程不能进入临界区，则应让出CPU，以免出现“忙等”现象。

2.4.1 进程同步的基本概念

◆ 同步机制应遵循的规则

➢ 空闲让进：

- 当无进程在互斥区时，任何有权使用互斥区的进程可进入

➢ 忙则等待：

- 不允许两个以上的进程同时进入互斥区

➢ 有限等待：

- 任何进入互斥区的要求应在有限的时间内得到满足

➢ 让权等待：

- 处于等待状态的进程应放弃占用CPU，以使其他进程有机会得到CPU的使用权


THEORY
PRACTICAL