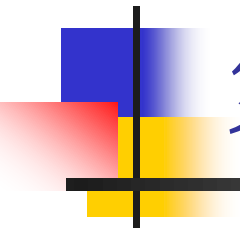
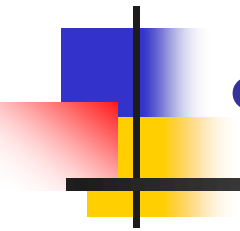


第2章 进程的描述与控制

第二章 进程的描述与控制

- 进程相关的基本概念
- 进程控制
- 进程同步
- 经典进程的同步问题
- 进程通信
- 线程



进程相关的基本概念

程序的顺序执行

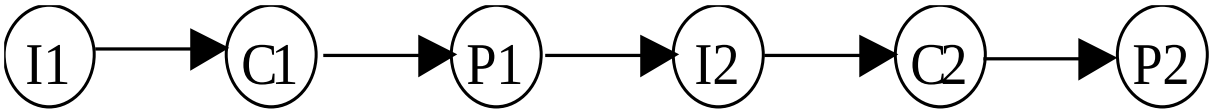
- 程序
  - 指令或语句序列
  - 执行方式：顺序执行和并发执行
- 顺序执行
  - 程序各部分（程序段或指令）依次顺序执行。当前操作完成后才能执行其后继操作
  - 例：

S1: a:=x+y;

S2: b:=a-5;

S3: c:=b+1;

三条语句的顺序执行



顺序执行的特征

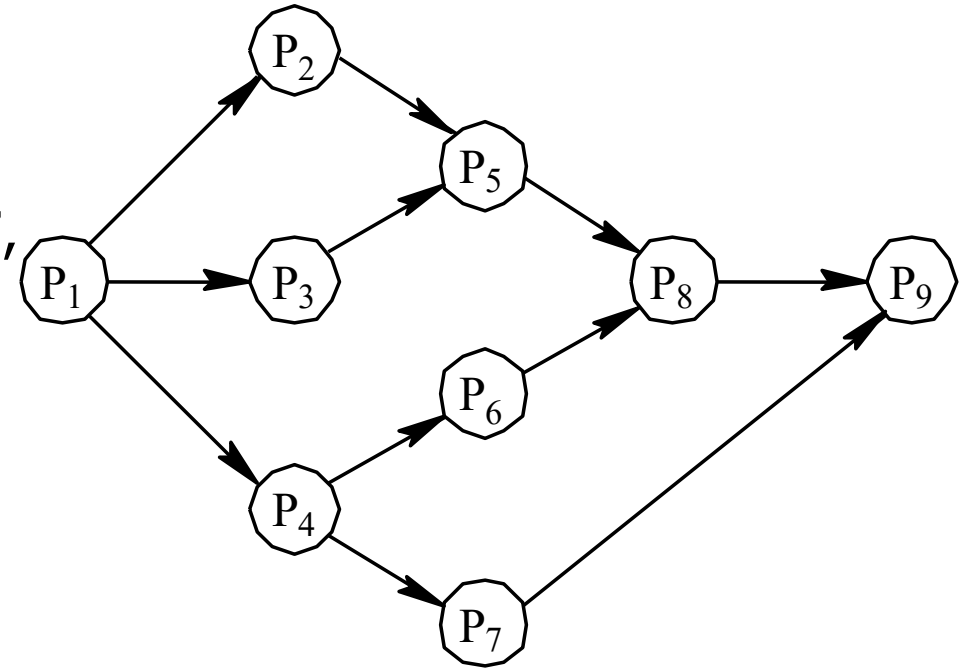
- 顺序性
  - 严格依次运行指令
- 封闭性
  - 独占全机资源
  - 执行结果不受外界影响
- 可再现性
  - 程序运行结果与程序执行速度无关，只要初始状态相同，结果应相同

前驱图(Precedence Graph)

- 有向无循环图，**DAG(Directed Acyclic Graph)**
- 用于描述进程之间执行的前后关系。
- 结点表示一个程序段、进程或语句
- 有向边表示两个结点之间的偏序**(Partial Order)**或前趋关系**(Precedence Relation)**“→”。
- **→={(Pi, Pj)|Pi must complete before Pj may start}**
- 如果**(Pi, Pj)**∈→,可写成**Pi→Pj**，称**Pi**是**Pj**的直接前趋，而称**Pj**是**Pi**的直接后继。
- 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)**。

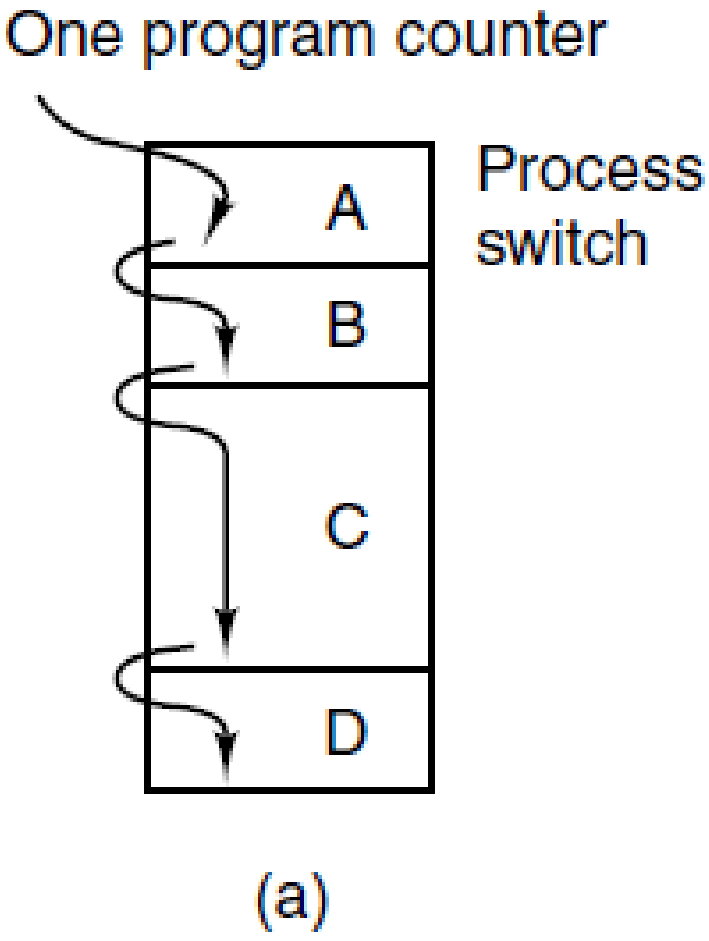
前驱图举例，

- 例1：S1→S2→S3
- 例2：
  - P1→P2, P1→P3, P1→P4,
    P2→P5, P3→P5, P4→P6,
    P4→P7, P5→P8, P6→P8,
    P7→P9, P8→P9
  - 或表示为：
  - 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, P9)}
- 前趋图中必须不存在循环

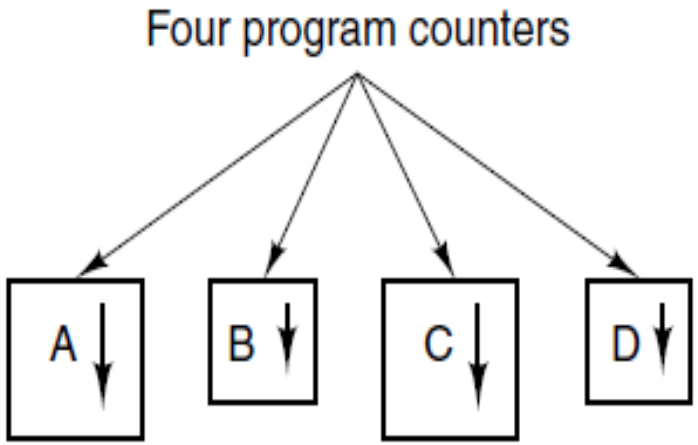


程序的并发执行

- 一组逻辑上相互独立的程序或者程序段在执行过程中，其执行时间在客观上互相重叠。
- 并发（**concurrency**）
- 并行（**parallel**）

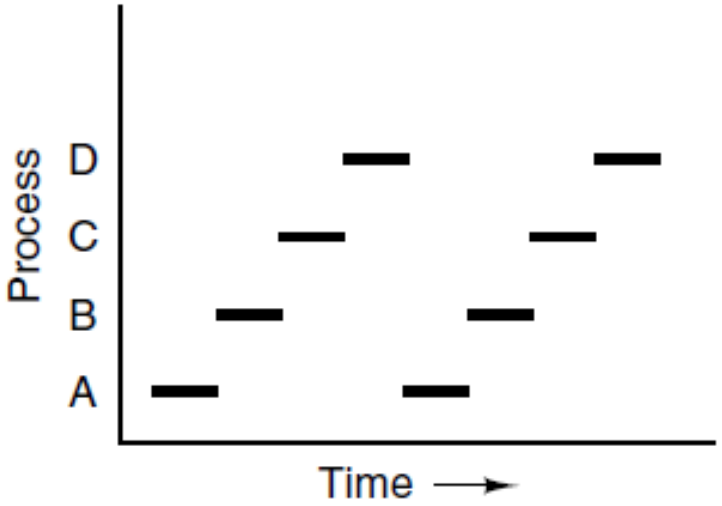


Multiprogramming of four programs



(b)

Conceptual model of four independent, sequential processes

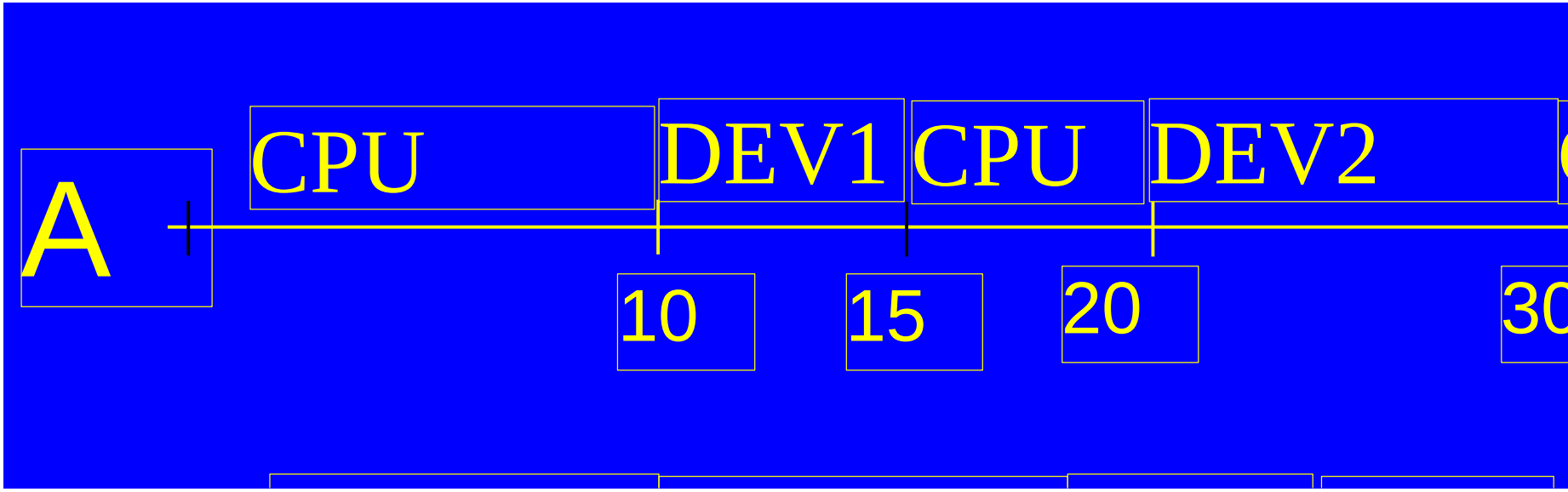


(c)

Only one program is active at once

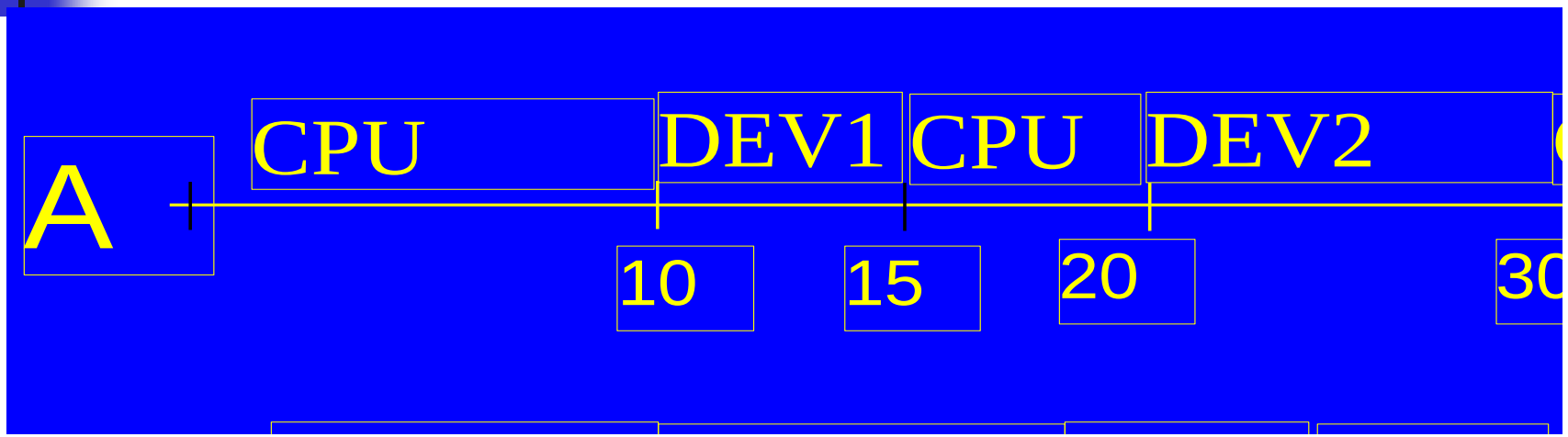
引入并发的目的，

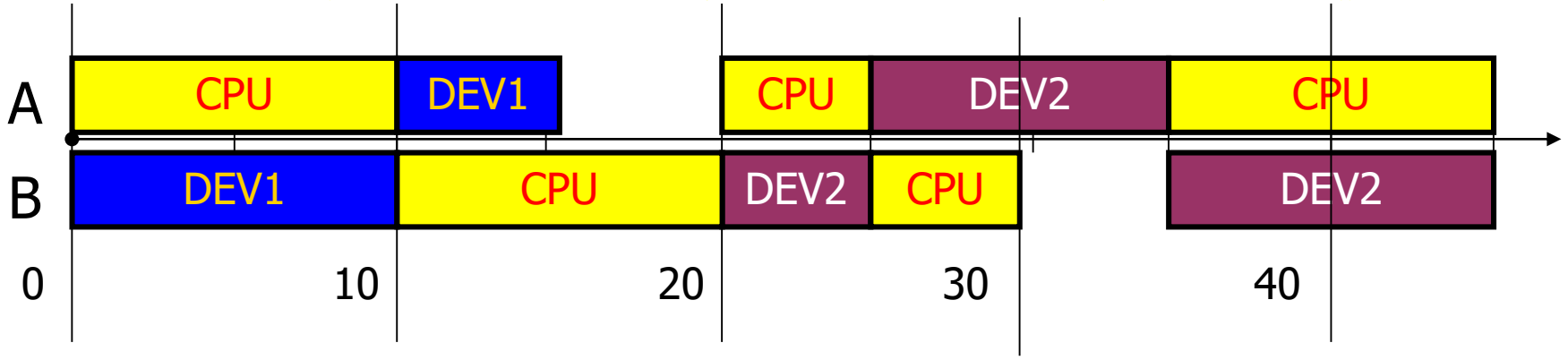
- 引入并发是为了提高资源利用率，从而提高系统效率。



- 在顺序环境下
  - **CPU利用率 = 40/80 = 50%**
  - **DEV1利用率= 15/80 = 18.75%**
  - **DEV2利用率= 25/80 = 31.25%**

引入并发的目的。



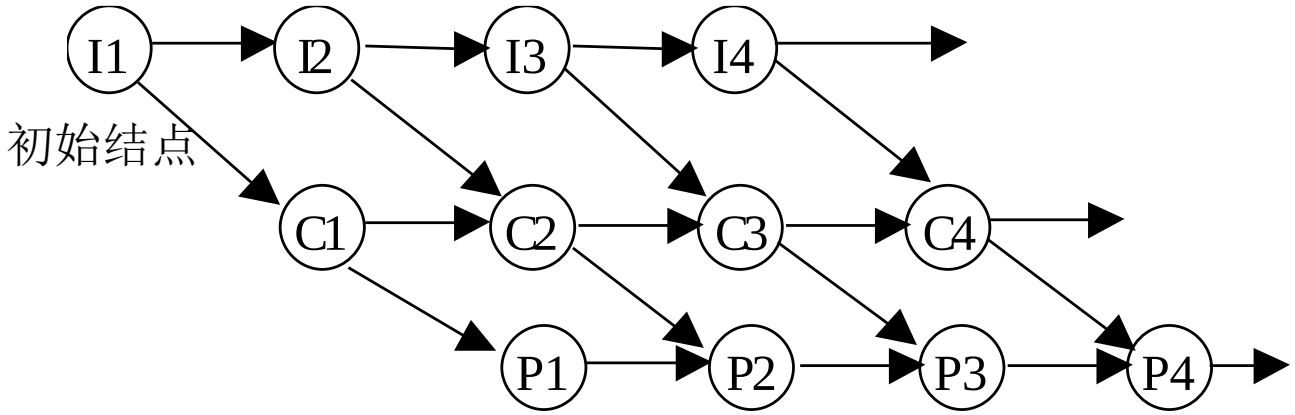


- 在并发环境下
  - **CPU利用率=40/45= 89%**
  - **DEV1利用率=15/45=33%**
  - **DEV2利用率=25/45=55.6%**

程序的并发执行可用一个无环有向图(前趋图)表示：

结点：表示一条语句，程序段，进程。

边：　表示两结点间的偏序(或前趋)关系。



如图：I1是C1的前趋

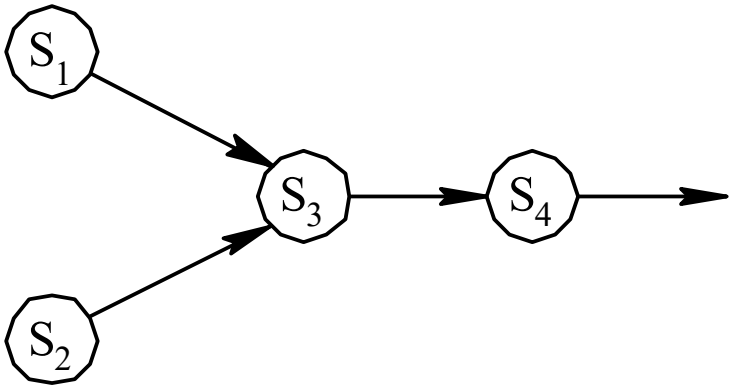
C1是I1的直接后继

I3，C2，P1可并发执行

程序并发执行

- 程序段
  - **S1: a:=x+2**
  - **S2: b:=y+4**
  - **S3: c:=a+b**
  - **S4: d:=c+b**

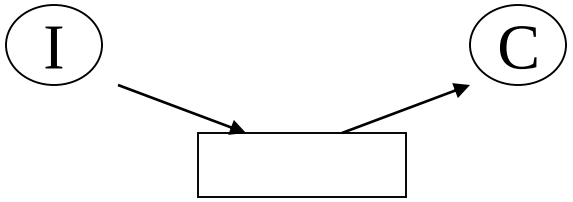
  - **S1和s2可以并发执行**



程序并发执行的特征

(1) 间断性

- 具有“走走停停”的规律，原因：
  - 缺乏共享资源，可能要等。
  - 几个程序合作完成一项任务时的相互间的制约。



只有I输入后C才能取数。

(2) 失去封闭性

- 程序并发执行，共享系统中的各种资源，资源的状态将由多个程序来改变，致使程序的运行已失去了封闭性。

(3) 不可再现性

- 程序在并发执行时，将失去其可再现性。例如下面两个程序在并发执行时是不可再现的。

```
Var     n:integer;
begin
        N:=0;
        parbegin
                program A:begin
                                repeat
                                    .
                                N:=N+1;
                                    .
                                Until false
                                end
                program B:begin
                                repeat
                                    .
                                Print(N)
                                N:=0;
                                    .
                                Until false
                                end
        parend
end
```

此程序执行失去了封闭性，为什么？

考虑并发执行的程序在任何地方都是可以中断的！！！

程序并发执行带来的影响。

- 如何控制并发执行的程序？

- 程序：顺序性、静态性、孤立性
  - 无法描述执行过程的随机性、并发性
  - 无法反映资源共享
- 进程
  - 刻画系统的动态性
  - 解决共享性
  - 描述程序的动态执行过程和使用共享资源的基本单位

进程

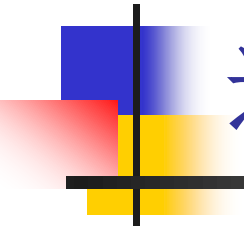
- 概念
  - **Process, 60年代中期由MIT在Multics系统中首先引入的，IBM370（task）**
  - 行为的一个规则叫做程序，程序在处理器上执行时所发生的活动称为进程－**Dijkstra**；
  - 进程是这样的计算部分，它可以与其他进程并发执行。（**Madnick&Donowan**）
  - 进程是程序在一个数据集合上运行的过程，它是系统进行资源分配和调度的一个独立单位－**Peter Denning**
  - 进程是一个具有独立功能的程序对某数据集在处理机上的的执行过程，也是操作系统进行资源分配的基本单位
  - “进程是进程实体的运行过程，是系统进行资源分配和调度的一个独立单位”

进程的特征

- 结构性
  - 为使程序并发执行，引入进程控制块以管理其运行
  - 程序、数据集和进程控制块
- 动态性
  - 进程是程序在数据集合上的一次执行过程，是动态概念。具有生命周期。
- 并发性
  - 多进程实体同存内存，在一段时间内同时执行
  - 任何进程都可以同其他进程一起向前推进。进程的执行是可以被打断的，或者说，进程执行完一条指令后在执行下一条指令前，可能被迫让出处理器，由其他若干个进程执行若干条指令后才能再次获得处理器而执行。
- 独立性
  - 进程是资源分配的基本单位，也是独立运行的基本单位。凡是未建立进程的程序，都不能作为独立单位参与运行。
- 异步性
  - 各进程按各自独立的不可预知的速度推进，按异步方式运行

程序与进程之间的区别

- 程序是静态的，进程是动态的：程序是有序代码的集合；进程是程序的执行。
- 进程有生命周期，有诞生有消亡，是暂时的；而程序通常对应着文件，是静态和可以复制，是相对可长久保存的。
- 进程具有并行特性，可以描述并发，而程序不能。
- 进程与程序的组成不同：进程的组成包括程序、数据和进程控制块（即进程状态信息）。
- 通过多次执行，一个程序可对应多个进程。
- 进程是竞争计算机系统资源的基本单位



进程状态及转换

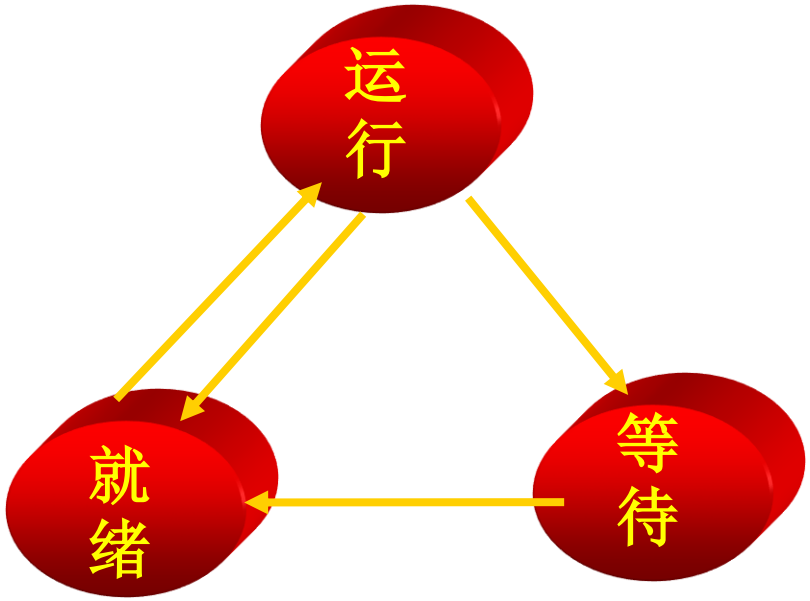
进程的基本状态

- 进程的生命周期
- 至少有三种基本状态
- 运行态（**Running**）：
  - 进程占有**CPU**，并在**CPU**上运行
- 就绪态（**Ready**）：
  - 一个进程已经具备运行条件，但由于无**CPU**暂时不能运行的状态（当调度给其**CPU**时，立即可以运行）
- 等待态**(Wait) /**阻塞**(Blocked)/**睡眠**(sleep)**：
  - 指进程因等待某种事件的发生而暂时不能运行的状态
  - （即使**CPU**空闲，该进程也不可运行）

进程状态转换

在进程运行过程中，由于进程自身进展情况及外界环境的变化，这三种基本状态可以依据一定的条件相互转换

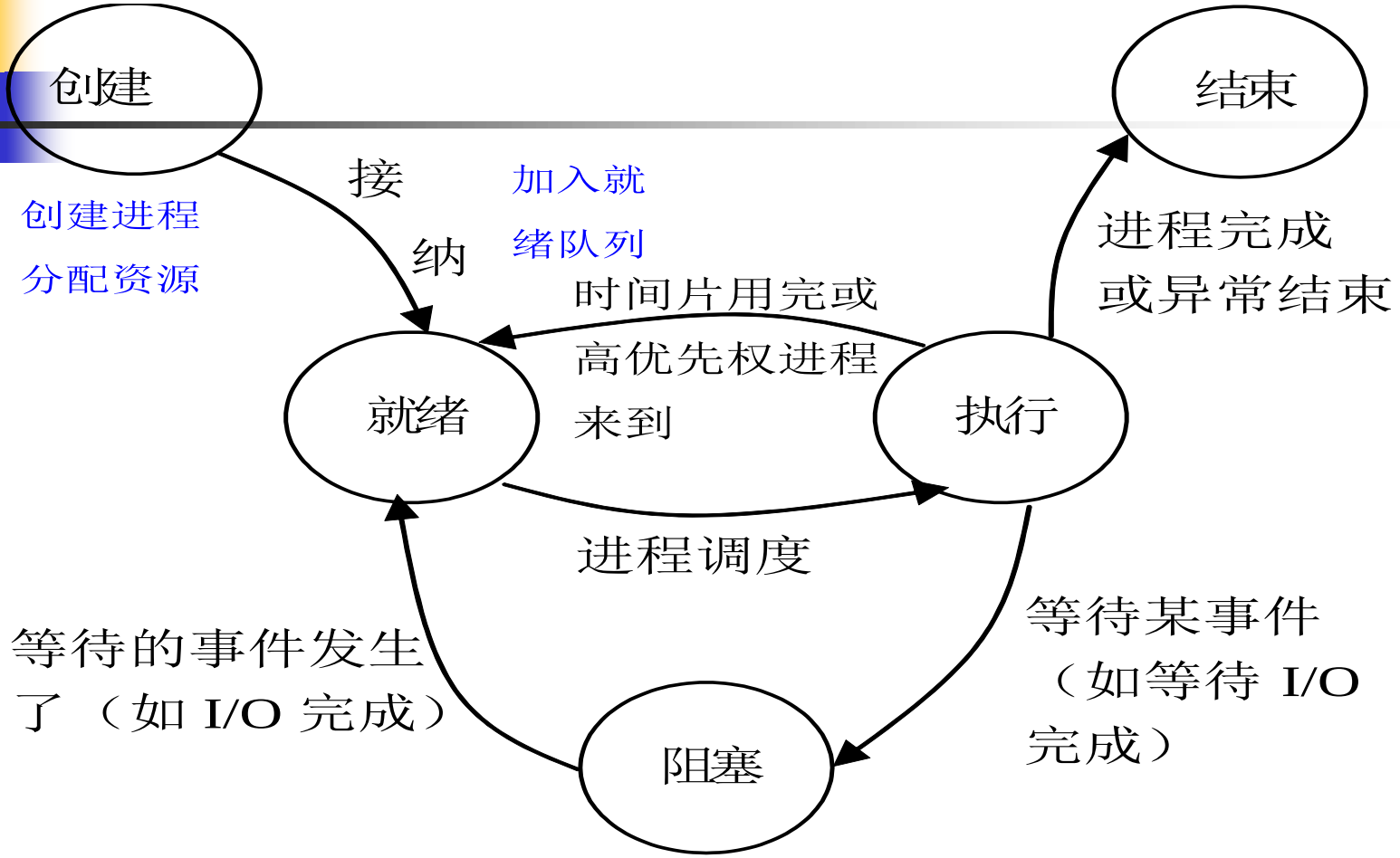
- 就绪 --> 运行
  - 调度程序选择一个新的进程运行
- 运行 --> 就绪
  - 运行进程用完了时间片
  - 运行进程被中断，因为一高优先级进程处于就绪状态
- 运行 --> 等待
  - 当一进程所需事件未发生时
  - **OS**尚未完成服务
  - 对一资源的访问尚不能进行
  - 初始化**I/O** 且必须等待结果
  - 等待某一进程提供输入
- 等待 -->就绪
  - 当所等待的事件发生时



进程的状态及其转换

进程状态－其他状态：

- 创建状态(new)
  - 创建PCB，填写必要信息,申请必要资源，将进程转入就绪状态并插入就绪队列,完成创建
  - 创建完成后才可参与调度
  - 系统可控制创建进程的提交，控制系统内进程数量和系统负载
- 终止状态(exit)
  - 中止后进程移入该状态，它不再有执行资格
  - 进程已结束运行，回收除PCB之外的其他资源，并让其他进程从PCB中收集有关信息（如记帐，将退出码exit code传递给父进程）。
  - 当信息不再需要后，进程被删除



注意：

*可逆　：仅就绪←→执行

*微观上：执行状态时，程序在运行

*宏观上：进程建立后直至撤消，程序都在运行。

挂起进程模型

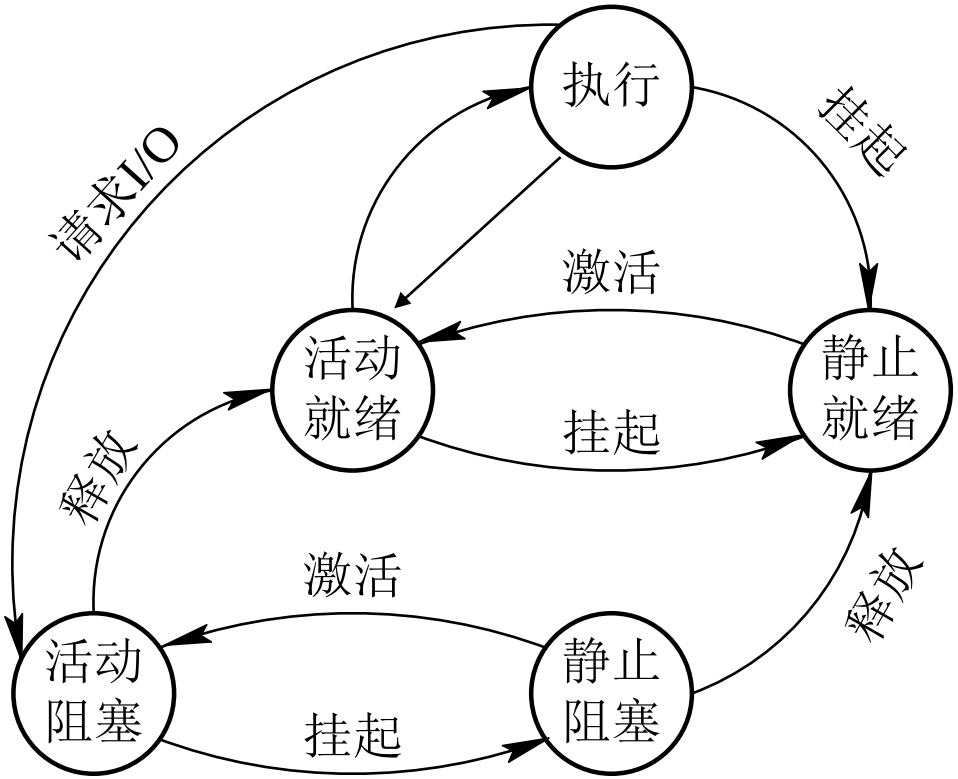
- 挂起（**Suspend**）：静止状态

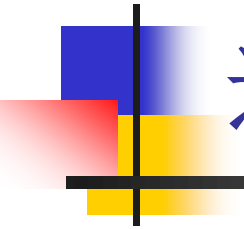
- 引入目的
  - 终端用户的请求。
    - 调试程序需要，在调试时，挂起被调试进程，从而对其地址空间进行读写
  - 父进程请求
    - 挂起子进程，考察修改，或协调各子进程
  - 负荷调节的需要
    - 实时系统负荷重时，挂起不重要进程，保证实时任务的运行
    - 内存不能满足进程的运行要求，需要把某些进程对换到磁盘，减轻系统负荷。如：换出等待进程以满足就绪进程的资源要求。
  - 操作系统的需要
    - 系统挂起进程以检查系统资源使用情况；出现故障；

挂起进程模型（续）

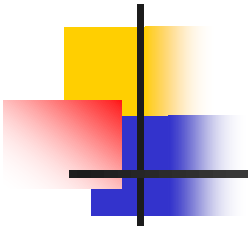
- 新增进程状态
  - 挂起（静止）状态，非挂起（活动）状态
  - 静止就绪；活动就绪；静止阻塞；活动阻塞

- 新增状态转换
  - 活动就绪 ->（挂起） ->静止就绪：
    - 当有高优先级就绪进程和低优先级就绪进程竞争时，系统可能会选择挂起低优先级就绪进程；
  - 活动阻塞→（挂起） →静止阻塞
    - 就绪进程要求更多内存资源时；
  - 静止阻塞 -->静止就绪
    - 事件出现，当等待的事件发生时
    - 针对外存进程的事件出现；
  - 静止就绪→（激活） →活动就绪
    - 当内存中没有就绪进程时
  - 静止阻塞→（激活） →活动阻塞
    - 当一个进程释放足够内存时，系统会把一个高优先级阻塞挂起进程（系统认为会很快出现所等待的事件）调入内存；
  - 运行->静止就绪
    - 高优先级抢占CPU，且内存不足
    - 自我挂起


执行
挂起
请求I/O
激活
活动
就绪
静止
就绪
挂起
释放
激活
释放
活动
阻塞
静止
阻塞
挂起



进程管理中的数据结构



- 进程的静态描述
  - 有关程序段＋数据集＋进程控制块**Process Control Block**
  - 其中，系统通过**PCB**识别和控制进程。 **PCB**中包含了进程的描述信息、控制信息和资源信息，

进程控制块

- 系统为了管理进程设置的一个专门的数据结构，用它来记录进程的相关信息，描述进程的运动变化过程
- 每个进程在**OS**中的登记表项（可能有总数目限制），系统利用**PCB**来控制和管理进程，所以**PCB**是系统感知进程存在的唯一标志
- 进程与**PCB**是一一对应的
- 处于核心段，通常不能由应用程序自身的代码来直接访问，而要通过系统调用，

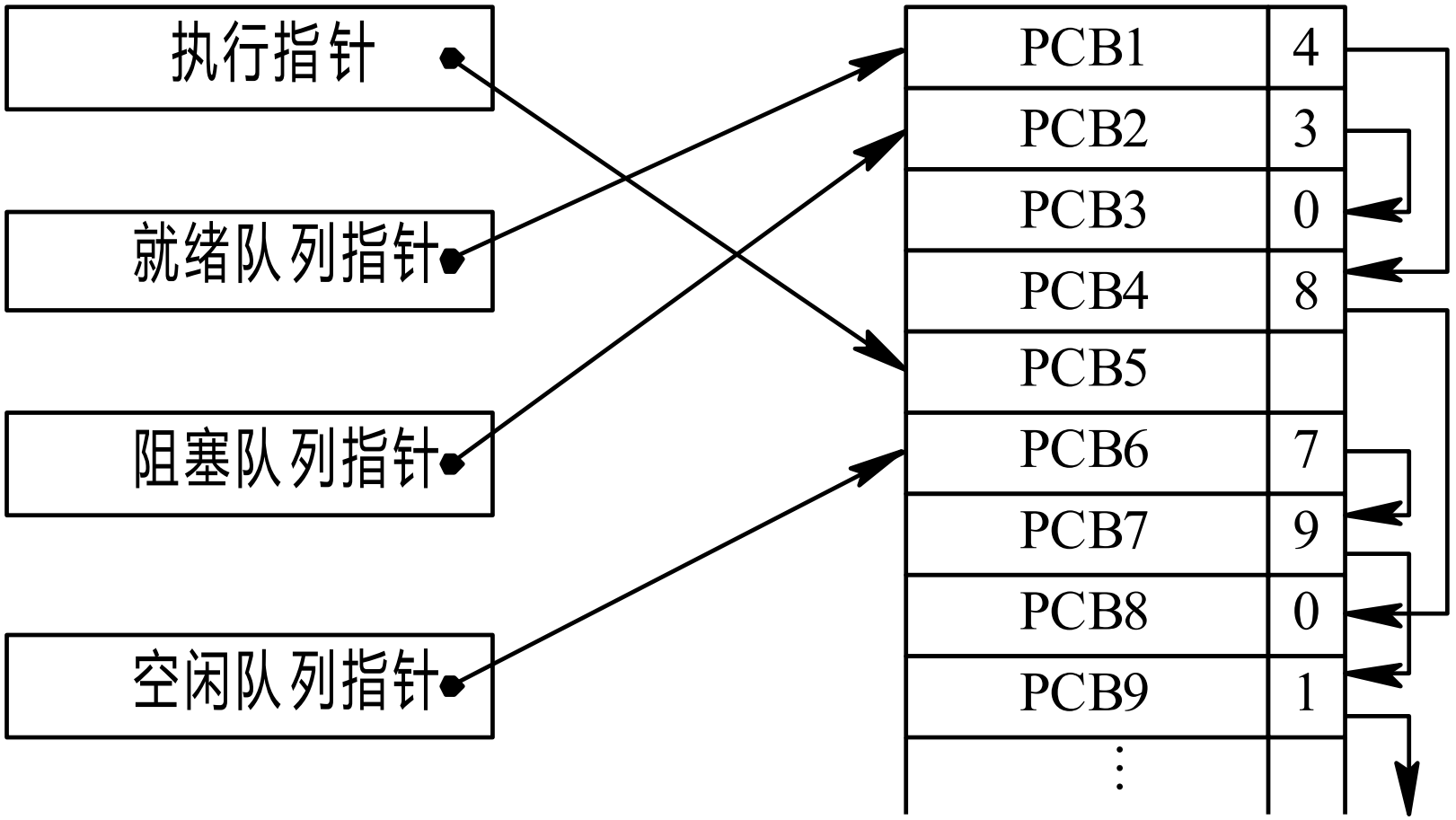
PCB中的信息

- 进程标志符
  - 内部标识符(process ID)，唯一，通常是一个整数；
  - 外部标志符，进程名，创建者提供；
  - 父进程标志、子进程标志，用户标志等，说明进程家族关系
- 处理机状态
  - 寄存器信息：通用寄存器，指令寄存器，程序状态字。用户堆栈指针。
- 进程调度信息
  - 进程当前状态，优先级，调度所需其它信息
  - 事件
- 进程控制信息
  - 程序和数据的地址
  - 进程同步和通信所需信息
  - 资源清单
  - 链接指针

PCB的组织方式

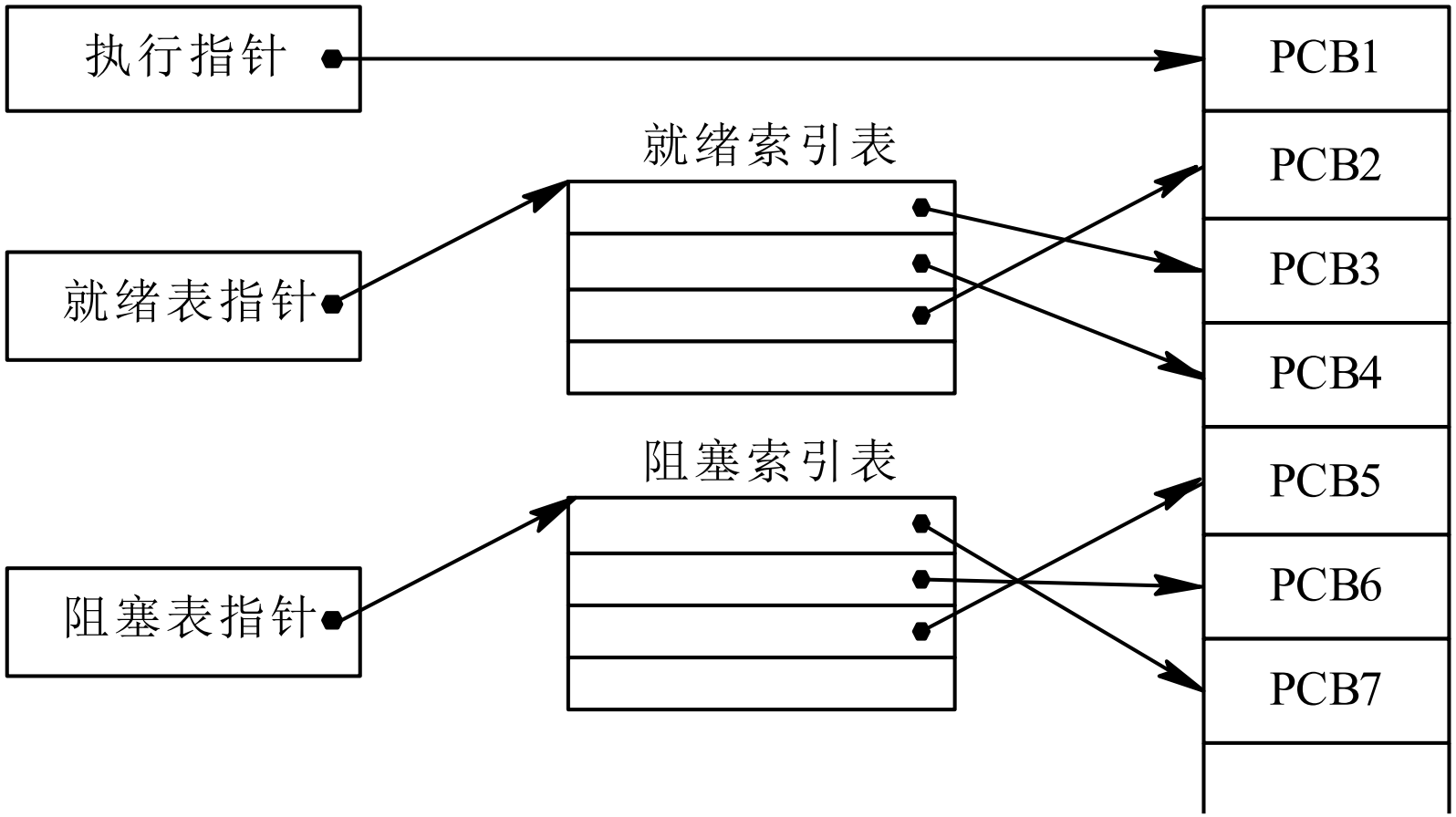
- **PCB表：** 系统把所有**PCB**组织在一起，并把它们放在内存的固定区域，就构成了**PCB**表
  - **PCB**表的大小决定了系统中最多可同时存在的进程个数，称为系统的并发度。
- 进程队列
  - 同一状态的所有进程控制块链接在一起构成进程队列；
  - 进程的状态发生变化时，进程相应地变换队列；
- 链接方式
- 索引方式

1) 链接方式



PCB链接队列示意图

2) 索引方式



按索引方式组织PCB

第二章 进程的描述与控制

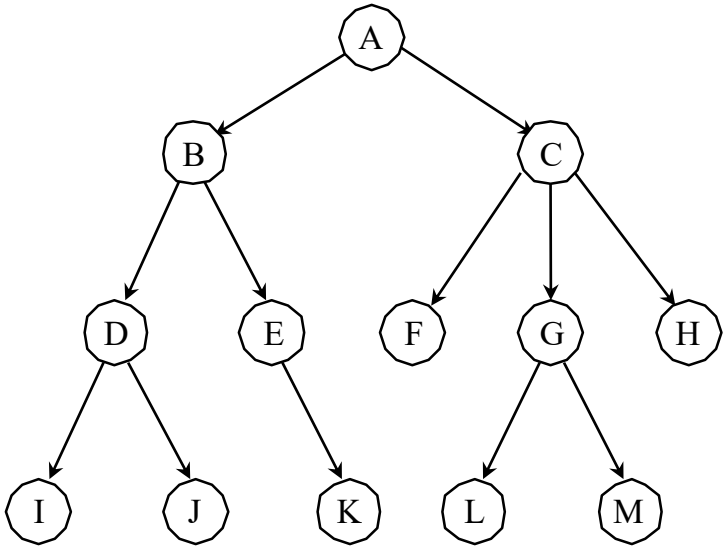
- 进程的基本概念
- 进程控制
- 进程同步
- 经典进程的同步问题
- 进程通信
- 线程

进程控制

- 目的
  - 处理机管理，协调执行并发进程，实现资源共享
- 内容
  - 创建、撤消进程以及完成进程各状态之间的转换。
- 手段
  - 进程控制程序段采用原语的形式
  - 原语(primitive)
    - 系统态下运行，完成系统特定功能的过程。
    - “原子操作(atomic operation)”过程，作为一个整体而不可分割——要么全都完成，要么全都不做。
    - 机器指令级
      - 执行期间不可中断
    - 功能级
      - 作为原语的程序段不允许并发执行

一、 进程创建

- 进程图
  - 描述进程家族关系的有向树
  - 根结点代表家族祖先
  - 父进程指向子进程
  - 子进程可以继承父进程的资源
  - 子进程撤销时应归还父进程资源
  - 撤销父进程要求撤销其子进程



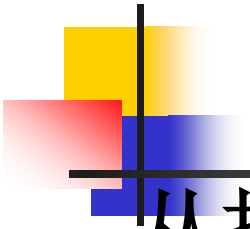
进程创建

- 引起创建进程的事件
  - 用户登录
    - 分时系统中用户在终端登录
  - 作业调度
    - 批处理系统中，选调作业进入内存，分配资源，创建进程
  - 提供服务
    - 系统创建进程为用户进程服务
  - 应用请求
    - 应用进程自己创建新进程

  系统进程或父进程可以创建新的进程

进程创建

- 创建原语
  - 申请PCB,赋予一个统一进程标识符
  - 分配资源, 如内存
  - 初始化进程控制块
    - 标志信息: 进程标志符,父进程标志符写入PCB
    - 处理机状态: PC指向程序入口地址,栈指针指向栈顶
    - 处理机控制信息: 状态为就绪
    - ...
  - 设置相应的链接
    - 如: 把新进程加到就绪队列的链表中



从技术上来说，只有一种创建进程的方法，即在一个已经存在的进程（用户进程或系统进程）当中，通过系统调用来创建一个新的进程。

Unix：fork函数；

Windows：CreateProcess函数;

二、进程终止

- 引起终止的事件
  - 进程运行结束
    - 停止指令或退出
  - 异常结束
    - 错误，故障
  - 外界干预
    - 操作员或操作系统终止该进程
    - 父进程终止子孙进程，父进程终止

进程的终止过程

(1) 根据被终止进程的标识符，从PCB集合中检索出该进程的PCB，从中读出该进程的状态。

(2) 若被终止进程正处于执行状态，应立即终止该进程的执行，并置调度标志为真，用于指示该进程被终止后应重新进行调度。

(3) 若该进程还有子孙进程，还应将其所有子孙进程予以终止，以防他们成为不可控的进程。

(4) 将被终止进程所拥有的全部资源，或者归还给其父进程， 或者归还给系统。

(5) 将被终止进程(它的PCB)从所在队列(或链表)中移出，等待其他程序来搜集信息。

三、进程的阻塞和唤醒

- 引发事件
  - 请求系统服务未果
  - 启动某种操作，操作未完成前
  - 新数据尚未到达
  - 当前工作完成，无新工作可做前
- 阻塞过程
  - 停止执行；
  - 修改**PCB**状态；插入阻塞队列；
  - 调度程序重新调度；切换处理机
- 唤醒过程
  - 移出阻塞队列；修改**PCB**状态；插入就绪队列

四、挂起与激活

- 挂起
  - 引起挂起事件
    - 用户进程挂起自己；父进程挂起子进程；系统挂起指定进程
  - 过程
    - 检查被挂起进程状态；活动就绪→静止就绪；活动阻塞→静止阻塞；拷贝**PCB**到指定内存区域
- 激活
  - 引起激活事件
    - 父进程或用户进程激活指定进程；内存空间足够
  - 过程
    - 从外存调入内存；检查挂起进程状态：静止就绪→活动就绪；静止阻塞→活动阻塞；如采用抢占调度策略，检查激活进程与当前进程的优先级，决定是否重新调度