第2章　操作系统及进程管理

2.1 操作系统概述

2.2 进程及进程通信

2.3 线程

2.4 文件

2.5 网络操作系统

第二章　操作系统及进程管理

2.2进程及进程通信

2.2.1 进程的引入

2.2.2 进程描述及进

2.2.3 进程控制

2.2.4 进程的同步与互斥

2.2.5 信号量机制

2.2.6 经典进程同步问题

2.2.7 进程通信

程序与进程
进程引入

2

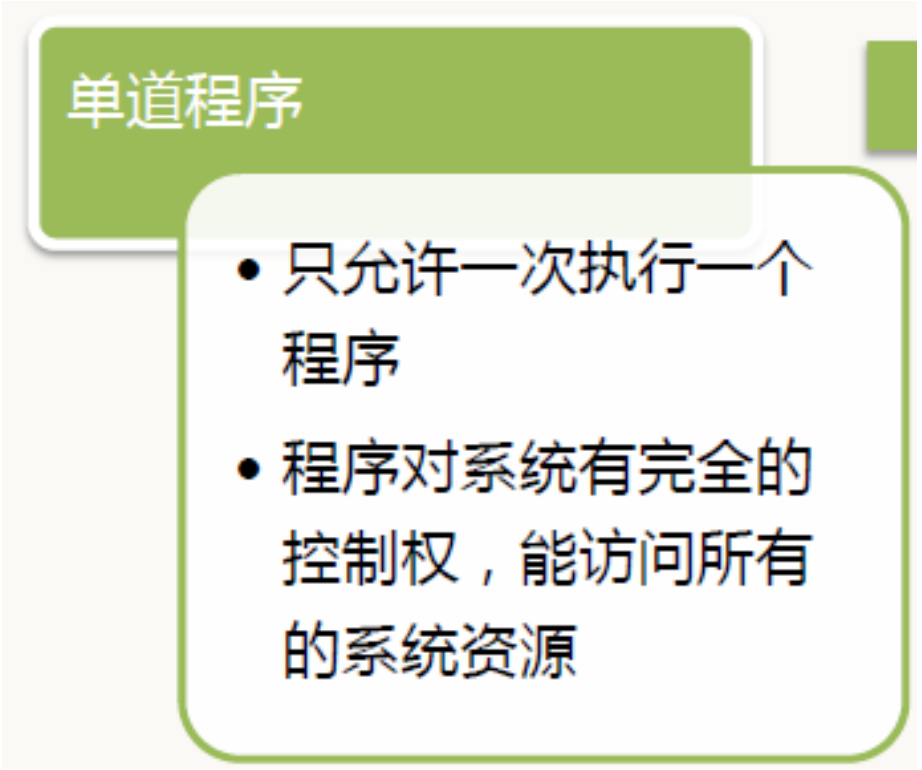
2.2.1 进程的引入

- 进程是信息在网络环境下通信的基本单位
- 在多道环境下，程序并不能独立运行，作为资源分配和独立运行的基本单位都是***进程***。
- 操作系统所具有的四大特征也都是基于进程而形成的，并可从进程的观点来研究操作系统。
- 进程是理解和控制系统并发活动的最基本、最重要的概念

3

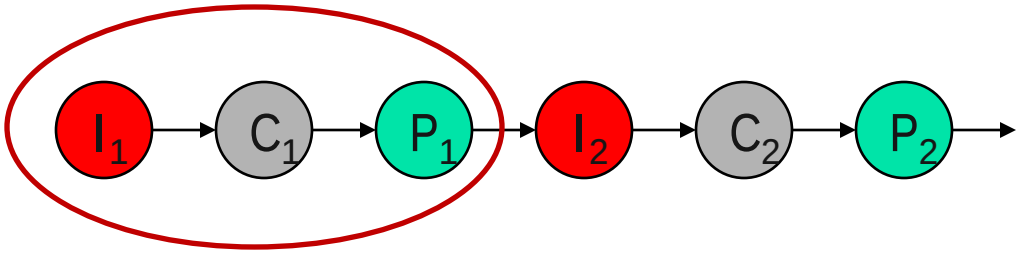
2.2.1 进程的引入

- 单道环境下程序称为单道程序
- 程序顺序执行



2.2.1 进程的引入
-- 程序的顺序执行及其特征

1.程序的顺序执行



5

2.2.1 进程的引入
-- 程序的顺序执行及其特征

2.程序顺序执行时的特征

1）顺序性：

- 处理机的操作严格按照程序所规定的顺序执行，即每一操作必须在下一个操作开始之前结束。

2）封闭性：

- 程序是在封闭的环境下执行的。

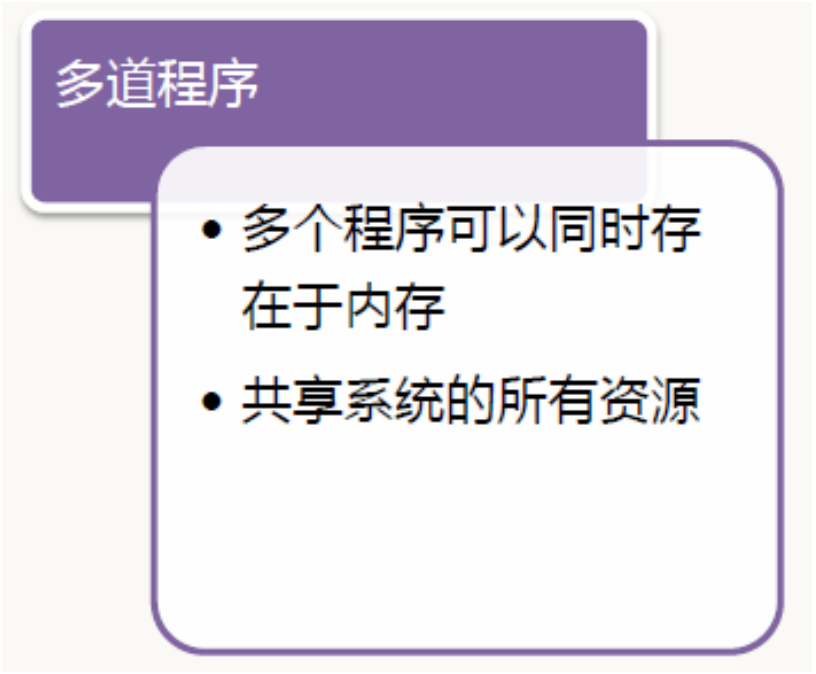
3）可再现性：

- 条件相同，结果相同

6

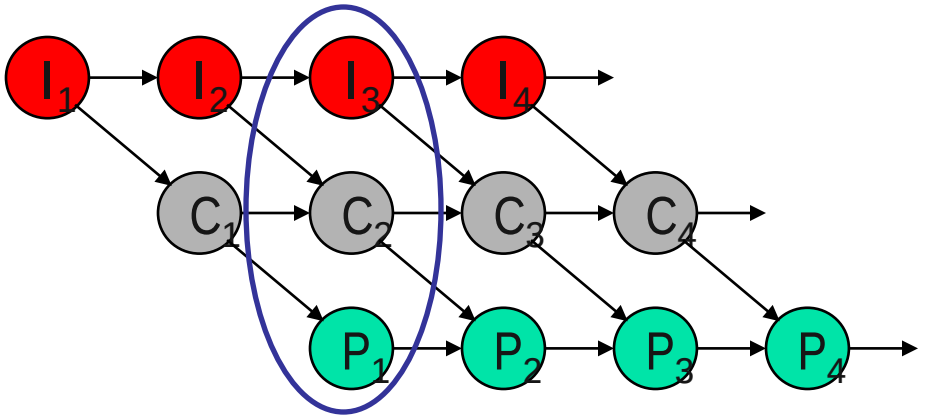
2.2.1 进程的引入

- 多道环境下程序称为多道程序
- 程序并发执行



2.2.1 进程的引入
-- 程序的并发执行及其特征

- 并发执行时的条件（Bernstein条件）
  - 一个程序的计算与另一个程序的I/O一般无交叉，可以并发执行：
- 同一个程序，输入→计算→输入串行执行
  - $I_i \rightarrow C_i$; $C_i \rightarrow P_i$;
- 不同程序，串行使用同一种设备
  - $I_i \rightarrow I_{i+1}$;
  - $C_i \rightarrow C_{i+1}$;
  - $P_i \rightarrow P_{i+1}$



2.2.1 进程的引入
-- 程序的并发执行及其特征

- 并发执行时的特征——不确定性
  - 间断性：
    - 相互制约程序出现“执行－暂停－执行”间断规律
  - 失去封闭性：
    - 一个程序环境受其它程序影响
  - 不可再现性：
    - 结果不确定

9

不确定性例：
两程序并行执行，共享变量N

```
Begin
  integer N;
  N:=0;
  begin                          begin
  L1:N:=N+1;                      L2:print(N);
      GOTO L1;                        N:=0;
   end;                                GOTO L2;
                                   end;
end
```

交替运行

10

- 并发程序不确定性的示例。
  - 假设一个火车订票系统程序，其中读取某车次车票余额并售出车票的程序片段为ticketP，现在两个窗口T1和T2并发执行这段程序，两个并发程序必须共享某车次的剩余车票数的变量tNum。

```
ticketP
… …
//从共享文件中读取车票数tNum
Read(tNum);
//如果还有余票，则售出，票数减1，假设每次只能售一张，否则票数不变，返回
if  tNum>=1 then tNum--;
else return(-1) ;
//车票数据写回共享文件
Write(tNum);
… …
```

2.2.1 进程的引入

顺序执行，结果正确

时刻	t0	t1	t2	t3	t4	t5	
变量 tNum 值	1	1 → 0	0	0	0	0	共 1 张车票
窗口 T1 执行	Read(tNum)	tNum--	Write(tNum)				卖出 1 张
窗口 T2 执行				Read(tNum)	return(-1)		无票

a.正确的情况

并发执行，
与时间相关错误

运行环境的非封闭性、结果不可再现性、间断性

时刻	t0	t1	t2	t3	t4	t5	
变量 tNum 值	1	1	0	0	0	-1	共 1 张车票
窗口 T1 执行	Read(tNum)		tNum--	Write(tNum)			卖出 1 张
窗口 T2 执行		Read(tNum)			tNum--	Write(tNum)	卖出 1 张

b.不正确的情况

2.2.1 进程的引入

- 错误原因：
  - 窗口T1和T2运行的程序使用的变量是共享的，两个程序运行环境不封闭
- 这个例子充分表现了并发的程序有间断性，失去封闭型，和结果不可再现性
- 我们也称这种问题为与时间有关的错误

单道程序

- 顺序性：处理机的操作严格按照程序所规定的顺序执行
- 封闭性：程序运行的环境资源只能由程序本身访问和修改
- 可再现性：只要它的运行条件（初始数据）相同，其运行结果一定相同

多道程序

- 间断性：各程序在执行时间上是重叠的，相互制约
- 失去封闭性：一个程序的环境可能会受其它程序影响
- 不可再现性：并发程序的运行结果不确定

程序的概念无法满足描述要求，引入新概念——进程

第二章　操作系统及进程管理

2.2进程及进程通信

- 2.2.1 进程的引入
- 2.2.2 进程描述及进程状态
- 2.2.3 进程控制
- 2.2.4 进程的同步与
- 2.2.5 信号量机制
- 2.2.6 经典进程同步问题
- 2.2.7 进程通信

进程概念及特点
进程控制块及进程映像
进程状态及状态变化

15

2.2.2 进程描述与状态

- 1.进程的特征和定义
- 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
- 不同的定义
  - 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
  - 进程是程序在一个数据集合上的运行过程，它是系统进行资源分配和调度的一个独立单位
  - 可以与其它程序并行执行的程序的一次执行

16

2.2.2 进程描述与状态

- 进程实体：
  - 进程映像，由程序、数据以及描述进程状态的数据结构组成的
- **进程的特征：**

与操作系统特点相同

动态性是进程的最基本特性

17

2.2.2 进程描述与状态

- **2.进程与程序比较：**

1.程序是指令的集合，是静态的概念；

进程是程序的执行过程，是动态的概念；

2.进程是程序的执行，因而它有生命过程，从投入运行到运行完成，所以进程有诞生和死亡

- 进程的存在是暂时的
- 程序的存在是永久的；

3.一个程序可以对应多个进程(即由多个进程共享)

一个进程又可顺序的执行多个程序。

18

2.2.2 进程描述与状态

3.进程控制块与进程映像

- 进程控制块PCB（Process Control Block）
  - 是唯一标识进程存在的数据结构
  - 包含了进程的描述信息和控制信息
  - 是进程的动态特征的集中反映。

2.2.2 进程描述与状态

– 进程控制块PCB（Process Control Block）

✓ 包含了进程的描述信息和控制信息

✓ 包含内容：

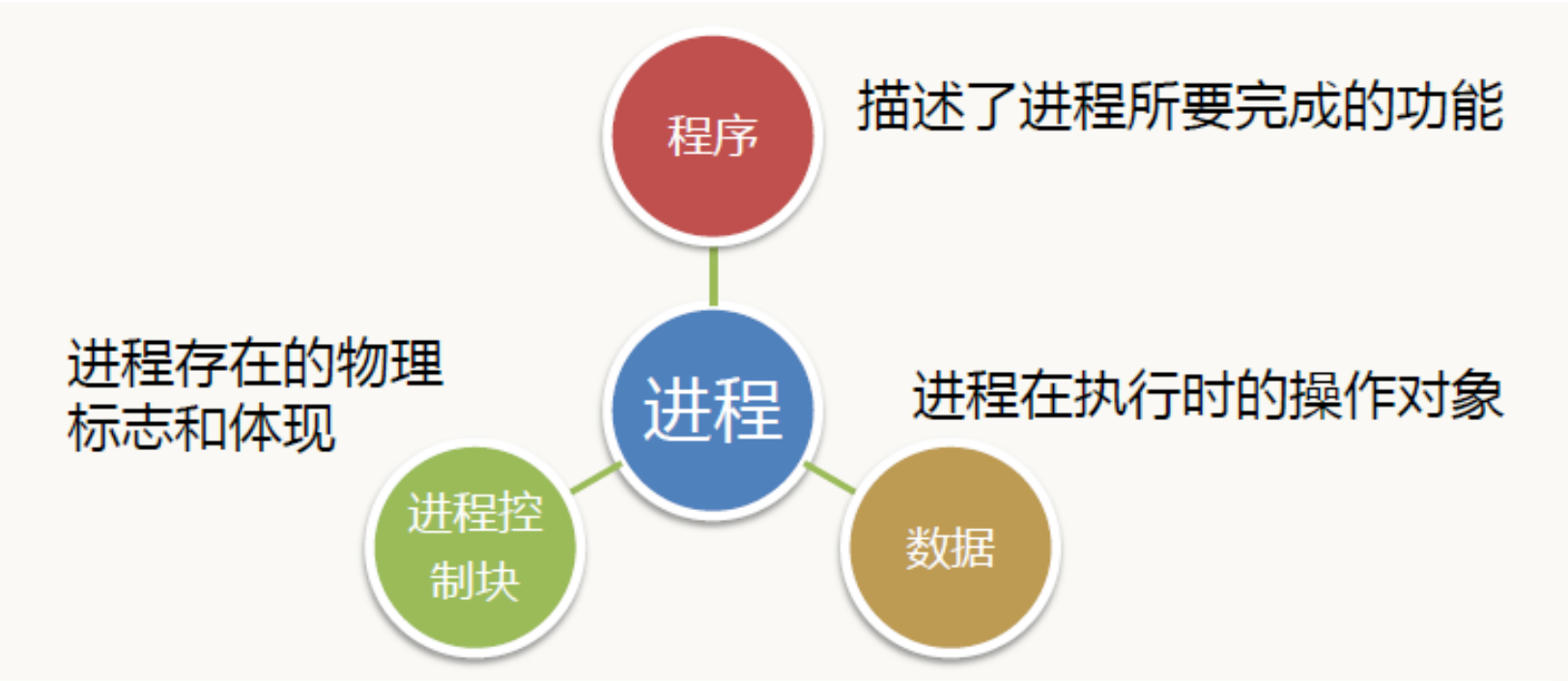
进程标识符	进程状态信息	进程执行现场信息	进程调度信息	进程控制信息

✓ 唯一标识进程存在的数据结构

✓ 使一个在多道程序环境下不能独立运行的程序（含数据），成为一个能独立运行的基本单位，一个能与其他进程并发执行的进程

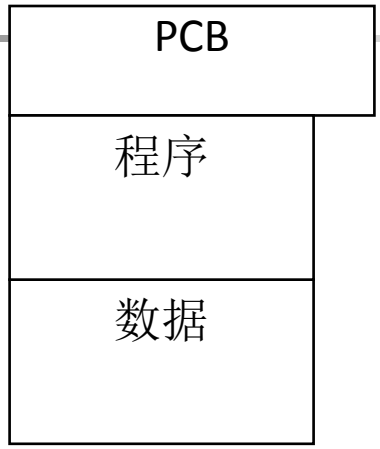
2.2.2 进程描述与状态

- 进程的实体即进程映像
  - 是由程序、数据和进程控制块组成的。



2.2.2 进程描述与状态

a．程序与数据合一



b.程序与数据独立

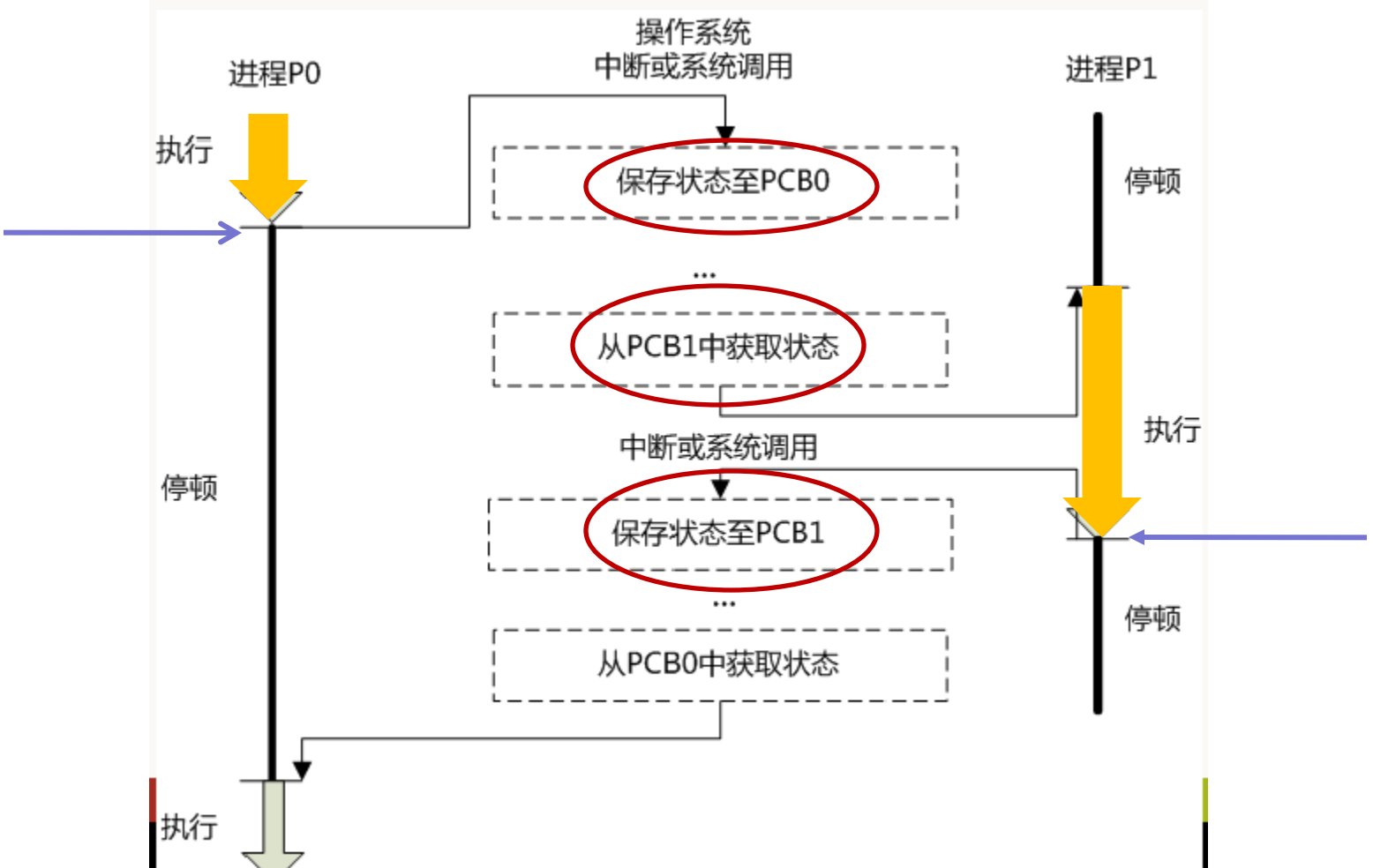
进程映像的三部分可以有不同的组合：

c.一个程序多个进程

22

2.2.2 进程描述与状态

- OS是根据PCB来对并发执行的进程进行控制和管理的

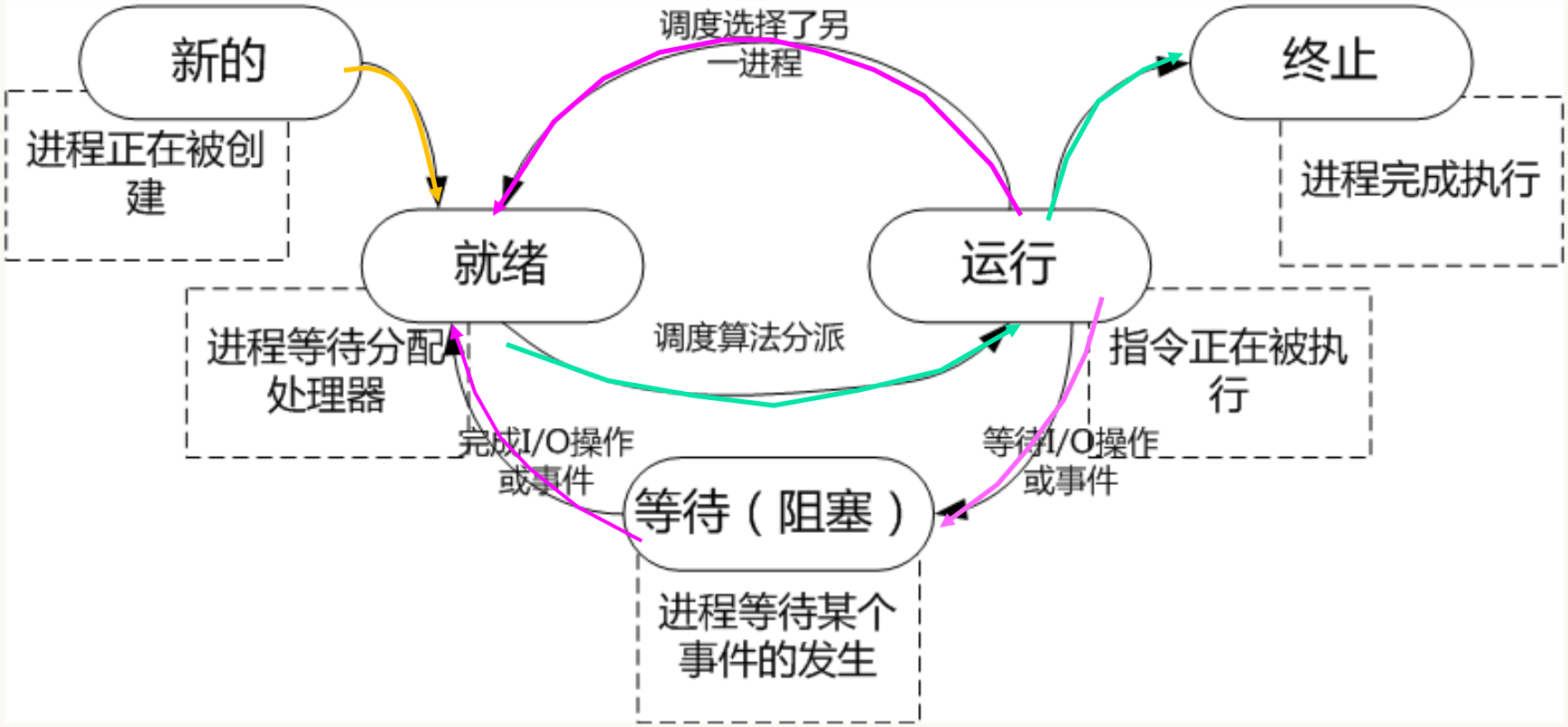


2.2.2 进程描述与状态

4.进程状态及状态变化

- 就绪状态（Ready）
  - 得到了除CPU以外的所有必要资源
- 执行状态（Running）
  - 已获得处理机，程序正在被执行
- 阻塞状态（Blocked）
  - 因等待某事件发生而暂时无法继续执行，从而放弃处理机，使程序执行处于暂停状态

2.2.2 进程描述与状态


新的
进程正在被创建
调度选择了另一进程
终止
进程完成执行
就绪
运行
进程等待分配处理器
调度算法分派
指令正在被执行
完成I/O操作或事件
等待I/O操作或事件
等待（阻塞）
进程等待某个事件的发生

第二章　操作系统及进程管理

2.2进程及进程通信

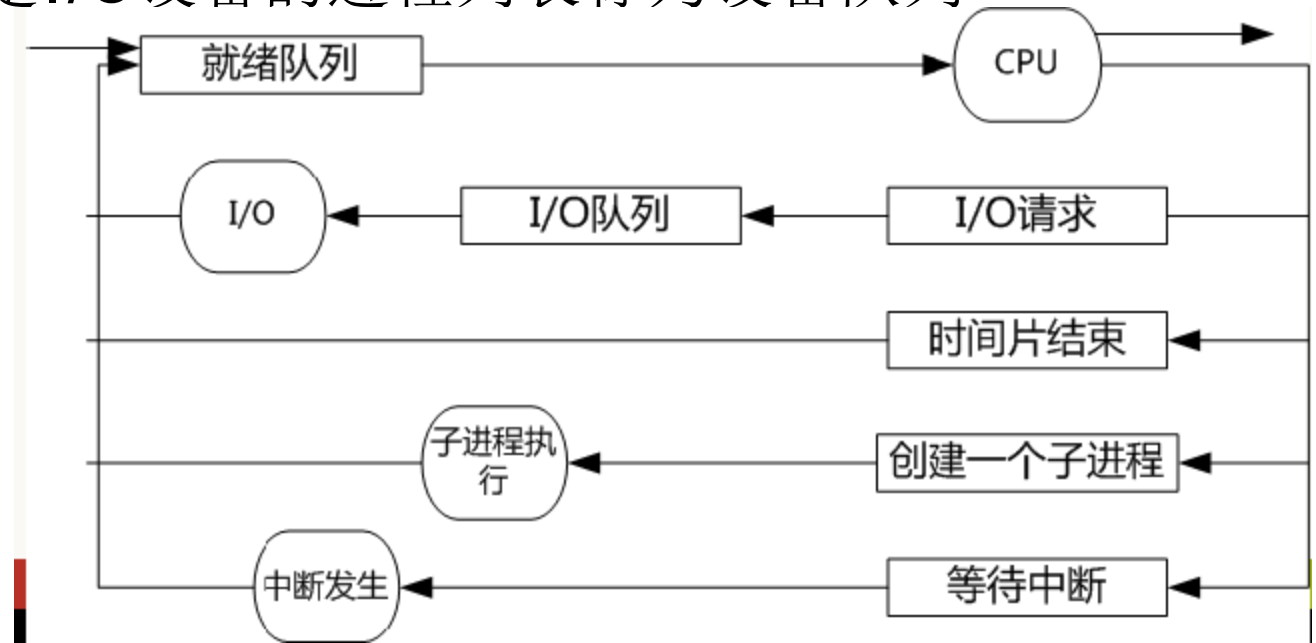
- 2.2.1 进程的引入
- 2.2.2 进程描述及进程状态
- 2.2.3 进程控制
- 2.2.4 进程的同步与
- 2.2.5 信号量机制
- 2.2.6 经典进程同步问题
- 2.2.7 进程通信

创建原语
撤销原语
阻塞原语
唤醒原语

26

2.2.3 进程控制

- 进程实现
  - 进程调度选择一个可用的进程到CPU上执行
  - 进程在不同的队列之间移动 驻留在内存中就绪的、等待运行的进程保存在就绪队列中
  - 等待特定I/O设备的进程列表称为设备队列



2.2.3 进程控制

- 进程状态变化是由系统的进程控制机构完成的
  - 负责控制进程从创建到撤消的自动执行与协调
  - 进程控制机构构成操作系统的内核（常驻内存）
- 内核
  - 是硬件的首次延伸，是加到硬件的第一层软件。
  - 由一些特殊的称为原语的程序段组成。
- 原语（Atomic Operation）：
  - 执行时不可分割的程序段
  - 可以看成是机器指令的延伸

28

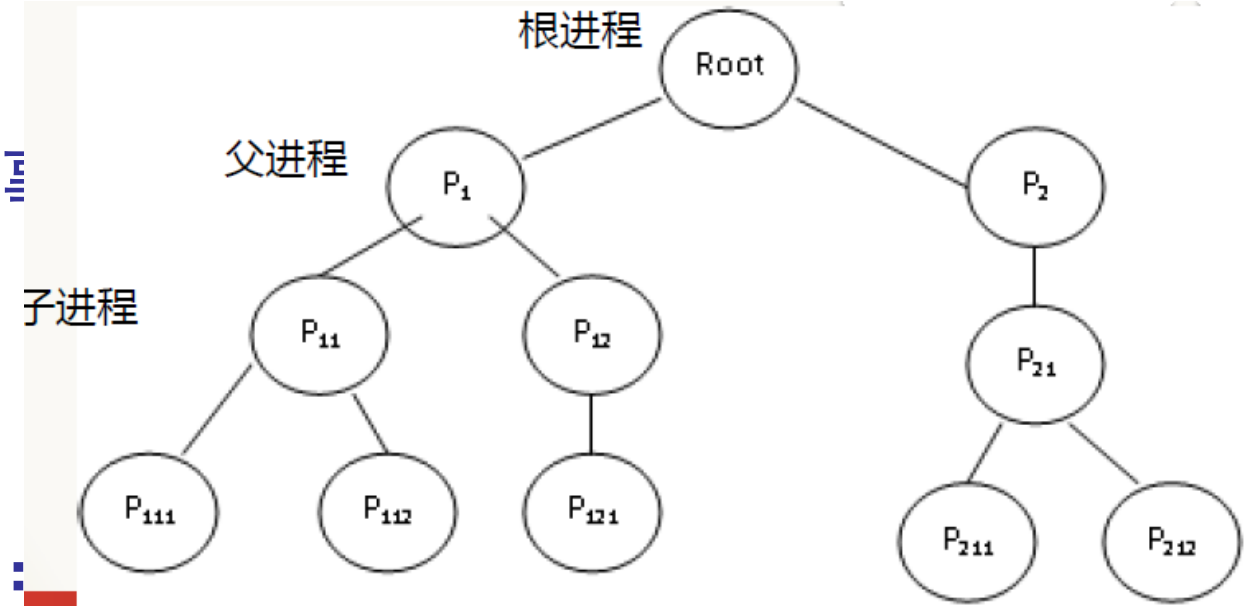
2.2.3 进程控制

- 进程控制包括
  - 进程创建
  - 进程撤消
  - 进程阻塞
  - 进程唤醒
  - 进程挂起与激活

2.2.3 进程控制

1.进程的创建

- 引起创建进程的事
  - 用户登录
  - 用户请求创建
  - 系统提供服务
- 进程之间的关系：
  - 父进程创建子进程
- 一个系统所有进程形成一棵树



2.2.3 进程控制

- 创建原语 主要工作：
  - 为被创建进程建立一个进程控制块PCB，分配进程标识符
- 进程的创建过程
  1）申请空白PCB
  2）为新进程分配资源
  3）初始化进程控制块
  4）将新进程插入就绪队列

```
Procedure
    Create(n,s0,k0,m0,R0,acc)
//n:进程名；s0:处理机初始状态；
//k0:优先级；m0:初始主存区；
//R0:其它资源清单；acc:计费信息
begin
   i:=get internal name(n);
   id(i):=n;priority(i):=k0;
   cpustate(i)=s0;
   mainstore(i):=M0;restores(i):=r0;
   status(i):=“readys”;
   sdata(i):=RL;
   parent(i):=*;
   progeny(i):=^;
   insert(progeny(*),i);
   set accounting data;
   insert(RL,i);
end
```

2.2.3 进程控

2.进程的终止

- 引起进程终止的事件
  - 正常结束：
    - 执行到最后的结束指令、中
  - 异常结束：
    - 出现错误或因故障而被迫终
  - 外界干扰：
    - 进程应外界的请求而终止运
- 进程的终止主要任务
  - 释放资源
  - 重新调度（需要时）

```
Procedure destroy(n)
begin
  sched:=false;
  i:=get internal name(n);
  KILL(i);
  If sched then scheduler;
end

Procedure KILL(i)
Begin
  if status(i)=“running” then
     begin stop(i);sched:=true
end;
  remove(queune(i),i);
  for all s∈progeny(i) do KILL(s);
  for all r∈(mainstore(i) U
resources(i)) do release(R);
  remove process control block(i);
end
```

2.2.3 进程控制—进程的终止

2.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。

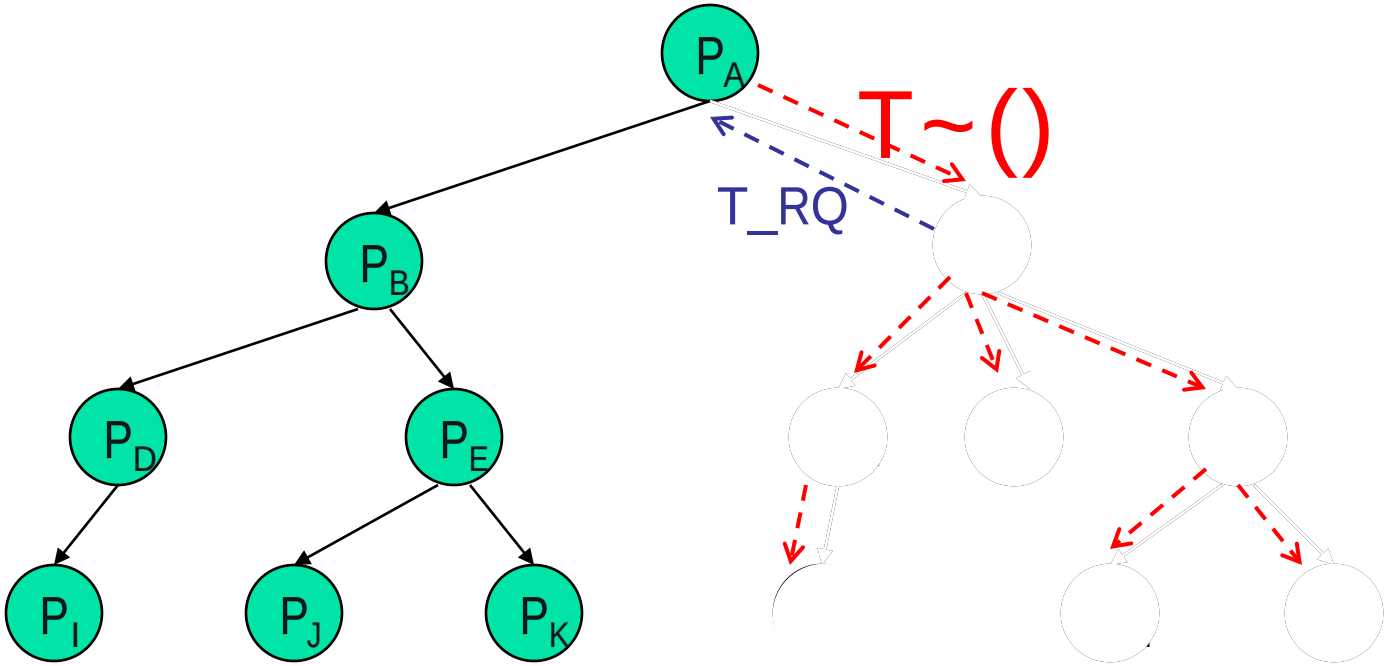
3）若该进程还有子孙进程，应将其子孙进程终止。

4）将被终止进程所拥有的资源，归还其父进程或系统

5）将被终止进程PCB从所在队列移出。

2.2.3 进程控制—

进程的终止（Terminat( )）



2.2.3 进程控制

```
Procedure block(n)
  begin
    i:=get internal name(*);
    Stop(i);
    Status(i):=“blockeda”;
    insert(WL(r),i);
    Scheduler;
  end
```

3.进程阻塞（Block( )）

- 引起阻塞的事件
  - 请求系统服务、启动某种操作、数据尚未到达
- 进程阻塞的过程
  - 发现上述事件，调用阻塞原语把自己阻塞
  - 停止进程的执行，修改PCB中的状态信息，并将其插入相应的阻塞队列
  - 转调度程序

2.2.3 进程控制

```
Procedure Wakeup(n)
begin
   i:=get internal name(n);
   remove(WL(r),i);
   Status(i):=if s="blockeda"
         then "readya" else readys"
   insert(RL,i);
end
```

4.进程唤醒（Wakeup( )

- 引起唤醒的事件
  - 与引起阻塞的事件相对应
- 进程唤醒的过程
  - 阻塞进程所期待的事件出现，有关的进程调用唤醒原语，将等待该事件的进程唤醒
  - 将PCB从阻塞队列中移出，修改PCB中的状态信息，再将其插入到就绪进程队列中
- 阻塞与唤醒要匹配使用，以免造成“永久阻塞”

第二章　操作系统及进程管理

2.2进程及进程通信

- 2.2.1 进程的引入
- 2.2.2 进程描述及进程状态
- 2.2.3 进程控制
- 2.2.4 进程的互斥与同步
- 2.2.5 信号量机制
- 2.2.6 经典进程同步问题
- 2.2.7 进程通信

- 互斥、同步概念
- 同步问题

37

2.2.4 进程互斥与同步

- 多道环境下的操作系统支持进程并发，并发的进程既有独立性又有相互制约性。
  - 独立性是指各进程都可以独立向前推进；
  - 制约性是指进程之间有时会相互制约，这种制约分为两种
- 两种形式的制约关系
  - 间接相互制约：源于进程对资源的共享
  - 直接相互制约：源于进程间的合作

38

2.2.4 进程互斥与同步

- 我们将并发进程的相互制约分为同步和互斥
  - 进程同步：合作完成同一个任务的多个进程，在执行速度或某些时序点上必须相互协调的合作关系。
  - 进程互斥：一个进程正在访问临界资源，另一个要访问该资源的进程必须等待。
- 操作系统要有进程间的同步与互斥措施控制，这是并发系统的关键问题，关系到操作系统的成败

2.2.4 进程互斥与同步--基本概念

1.互斥

- 竞争资源的进程首先面临的是互斥的要求，这种要求与竞争的资源特性有关
- 临界资源：一次仅允许一个进程使用的资源称为临界资源
- 许多物理设备属于临界资源，如：打印机、磁带机；
- 许多变量、数据、队列也可以若干进程共享使用，这时这些资源也是临界资源。如：车票文件

40

2.2.4 进程互斥与同步

- 一个进程可能包含使用临界资源和不使用临界资源两部分程序，我们将这两段程序从概念上分开
  - **临界区CS**（Critical Section）：进程（程序）中访问临界资源（硬件资源、软件资源）的代码段
  - **同类临界区**：与同一个临界资源相关联的临界区
  - 非临界区non_CS：不访问临界资源的代码段

41

2.2.4 进程互斥与同步

- 使用临界区的基本要求：
  - 互斥进入：在共享同一个临界资源的所有进程中，在同一时间，每次至多有一个进程处在临界区内，即只允许一个进程访问该临界资源；
  - 不互相阻塞：如果有若干进程都要求进入临界区，必须在有限时间内允许一个进程进入，不应互相阻塞，以至于哪个进程都无法进入；
  - 公平性：进入临界区的进程要在有限时间内退出，不让等待者无限等待

2.2.4 进程互斥与同步

例：临界区问题。考虑两个进程P1、P2共享变量count,程序如下：

```
Count:=0;
P1:                    P2:
   R1:=count;             R2:=count;
   R1:=R1+1;              R2:=R2+1;
   count:=R1;             count:=R2;
```

情况1：P1、P2顺序执行，
结果：count=2

正确！

情况2： P1、P2并发执行

T0:R1=count
T1:R2=count
T2:R1=R1+1,count=R1
T3:R2=R2+1,count=R2

结果：count=1

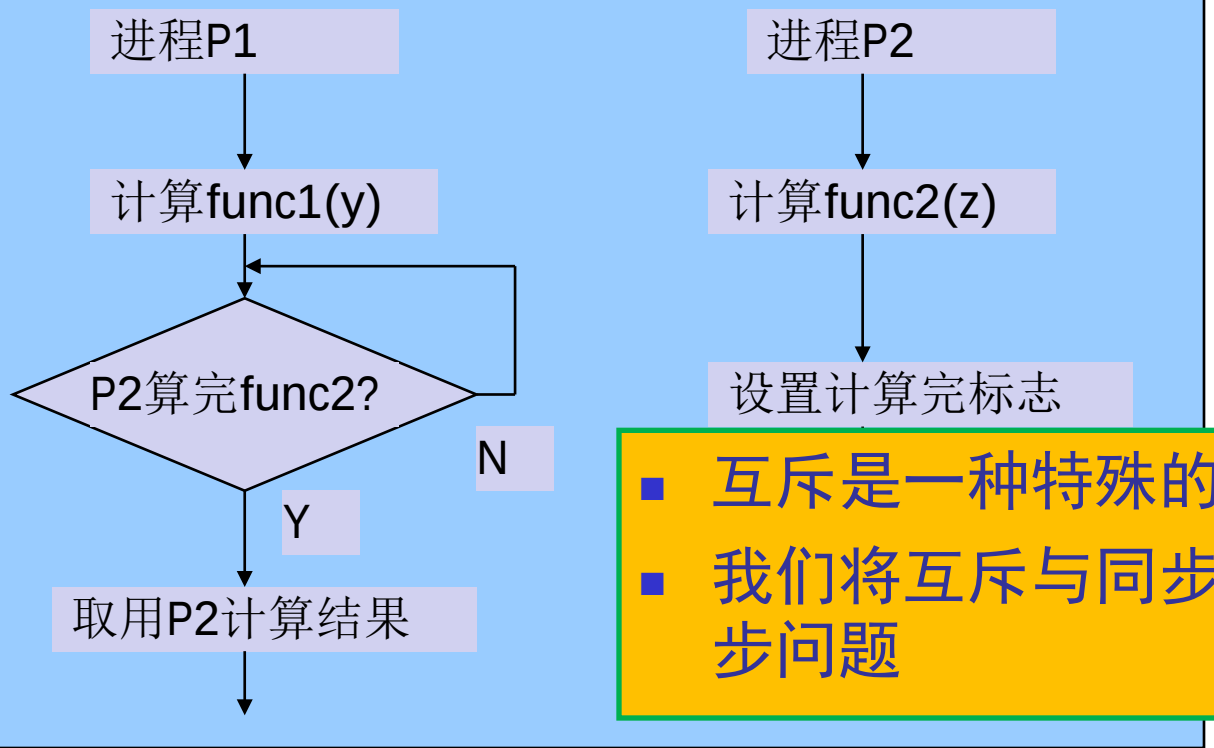
错误：这是与时间有关的错误，执行临界区的若干个进程必须互斥进入

2.2.4 进程互斥与同步

2.同步概念与同步问题

- 同步(synchronism)是指有协作关系的进程之间需要调整它们之间的相对速度。

x=func1(y)*func2(z),func1与func2由两个进程完成



- 互斥是一种特殊的同步关系
- 我们将互斥与同步面临的问题统称为同步问题

2.2.4 进程互斥与同步

- 操作系统设置同步机制应遵循的规则
  - 空闲让进：
    - 当无进程进入临界区，应允许一个请求进入临界区的进程立即进入临界区
  - 忙则等待：
    - 当已有进程进入临界区，其它请求进入临界区的进程必须等待，以保证对临界资源的互斥访问
  - 有限等待：
    - 对请求进入临界区的进程，应保证在有限时间内能够进入临界区，避免“死等”
  - 让权等待：
    - 当进程不能进入自己的临界区时，应立即释放已占用资源，以免产生死锁

45

2.2.4 进程互斥与同步

repeat

entry section

Critical section

exit section

remainder section

Until false

```
Repeat
        Non_CS;
        临界区入口控制代码CS-inCode ;
         CS ;
        临界区出口控制代码CS-outCode ;
Until false
```

第二章　操作系统及进程管理

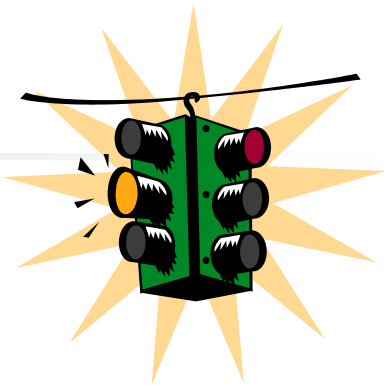
2.2进程及进程通信

- 2.2.1 进程的引入
- 2.2.2 进程描述及进程状态
- 2.2.3 进程控制
- 2.2.4 进程的同步与互斥
- 2.2.5 信号量机制
- 2.2.6 经典进程同步
- 2.2.7 进程通信

- 信号量概念
- 信号量机制控制进程互斥与同步

47

2.2.5 信号量机制



针对进程互斥问题，1965年，荷兰人Dijkstra首先提出信号量机制

2.2.5 信号量（灯）机制

- 信号量（Semaphores）
  - 信号量是一个数据结构，它由一个信号量变量以及对该变量进行的原语操作组成。
- 信号量机制
  - 操作系统利用信号量实现进程同步与互斥的机制
- 基本原理是：
  - 两个或多个进程可以通过简单的信号进行合作，一个进程可以被迫在某一位置停止，直到它接收到一个特定的信号
- 是现代操作系统在进程之间实现互斥与同步的基本工具

2.2.5 信号量（灯）机制

- 信号量的类型：
  - 整型：*S*为初值非负的整型变量，通常描述资源的状态或可用资源的数量。
  - 记录型：二元组(*S*,*Q*)，Q初始状态为空的队列。
  - AND型：一次需要多个共享资源，改进P-V操作。
  - 信号量集：一次需要N个多类资源，改进P-V操作。

50

2.2.5 信号量（灯）机制

1.整型信号量

- 最初信号量变量定义为整数值变量，在它上面定义三个原语操作：

  1）一个信号量可以初始化成非负整数。

  2）原语操作P（P操作）：

  判断信号量值，

  如果为0，忙等待(判断——等待)

  否则将信号量值减1；

  3）原语操作V（V操作）：

  将信号量值加1；

```
p(S):
   while S<=0 do no-op
   S:=S-1
v(S):
   S:=S+1;
```

2.2.5 信号量（灯）机制

- 使用P、V操作作为进入临界区的入口控制代码和出口控制代码，可以有效实现临界资源的互斥使用。

```
Repeat
        Non_CS;
        临界区入口控制代码CS-inCode ;
         CS ;
        临界区出口控制代码CS-outCode ;
Until false
```

```
p(S):
   while S<=0 do no-op
   S:=S-1
v(S):
   S:=S+1;
```

2.2.5 信号量（灯）机制

- 例.两个进程P1、P2共享临界资源CR，同类临界区为CS_1、CS_2
- 使用P、V操作作为CS_1、CS_2的入口和出口控制码，二者共享信号量mutex

Cobegin Semophore ***mutex=1***;	
P1: Repeat P(mutex); CS_1; V(mutex); non_CS_1; Until false	P2: Repeat P(mutex); CS_2; V(mutex); non_CS_2; Until false
coend	

两个临界区的一种推进顺序

```
P1:
Repeat
   P(mutex);
   CS1;
   V(mutex);
   non_CS1;
Until false
```

```
P2:
Repeat
   P(mutex);
   CS2;
   V(mutex);
   non_CS2;
Until false
```

	t0	t1	t2	t3	t4	t5	t5
进程P1执行	P操作：判断mutex值为1；mutex --；进入临界区	临界区工作，访问临界资源		退出临界区；执行V操作 mutex++；		non_CS1；	non_CS1；
进程P2执行			P操作：判断mutex值为0；在P操作中等待		P操作：判断mutex值为1；mutex --进入临界区	临界区工作，访问临界资源	退出临界区；执行V操作：mutex++；

2.2.5 信号量（灯）机制

- 整型信号量机制成功的控制了进程对临界资源的互斥访问
- 但是当S<=0时，P操作仍然参与CPU分配，执行while操作，判断S值
- 这个语句每次需要占用处理机，这就浪费了宝贵的处理机资源
- 产生了记录型信号量机制。

```
p(S):
    while S<=0 do no-op
    S:=S-1
v(S):
    S:=S+1;
```

2.2.5 信号量（灯）机制

2. 记录型信号量

- 记录型信号量数据结构中，除了一个整数变量外，还加了一个指针队列，用以记录阻塞进程
- P操作又称wait操作
- v操作又称signal操作

```
Struct Semaphore {

int value;

List_of_process L;

}S;
```

2.2.5 信号量（灯）机制

- 记录型信号量的几点说明
  - 信号量初值，一定是一个非负的整数
  - 调用wait的进程如果对信号量当前值减1后，信号量值大于等于0，则该进程继续运行下去
  - 否则它就被阻塞，直到有别的进程通过做signal（S）来唤醒它
  - 调用signal的进程状态不会改变

```
wait(S) {
        S.value--;
        if S.value<0 then
        block(S,L);
      }
```

```
signal(S) {
        S.value++;
        if S.value≦0 then
        wakeup(S,L);
    }
```

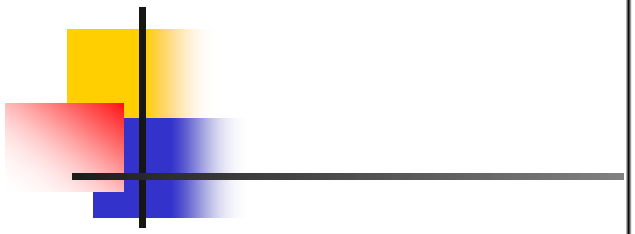
2.2.5 信号量机制

3. 利用信号量实现进程互斥

- 使用信号量互斥访问临界资源，需要设置一个互斥信号量，其初值必须为1
- 以Wait操作作为临界区入口控制码，Signal操作作为临界区出口控制码

```
Repeat
    Wait(mutex);
    CSi;
    Signal(mutex);
    non_Csi
Until false
```


58



```
Cobegin
Semophore    mutex=1;
```

P1:	P2:

Repeat
wait(mutex);
CS_1;
signal(mutex);
non_CS_1;
Until false
	Repeat
	wait(mutex);
	CS_2;
	signal(mutex);
	non_CS_2;
	Until false

```
coend
```

	t0	t1	t2	t3	t4	t5
进程P1执行	wait操作：判断mutex值为1；mutex --；进入临界区	临界区工作CS1，访问临界资源		退出临界区执行signal操作：mutex++ mutex=0 唤醒		
进程P2执行			wait操作：判断mutex值为0；阻塞		临界区工作CS2，访问临界资源	退出临界区执行signal操作：mutex++

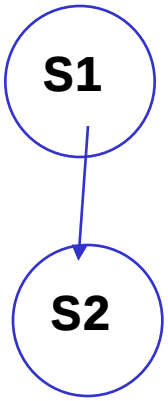
2.2.5 信号量（灯）机制

4. 利用信号量实现进程同步

例如：有两个并发进程P1具有语句s1,P2具有语句s2，而s2的执行必须等待s1的完成。

设：P1、P2共用信号量synch，初始化为0，则：

```
P1:s1;
     Signal(synch);
P2:Wait(synch);
     s2;
```



60

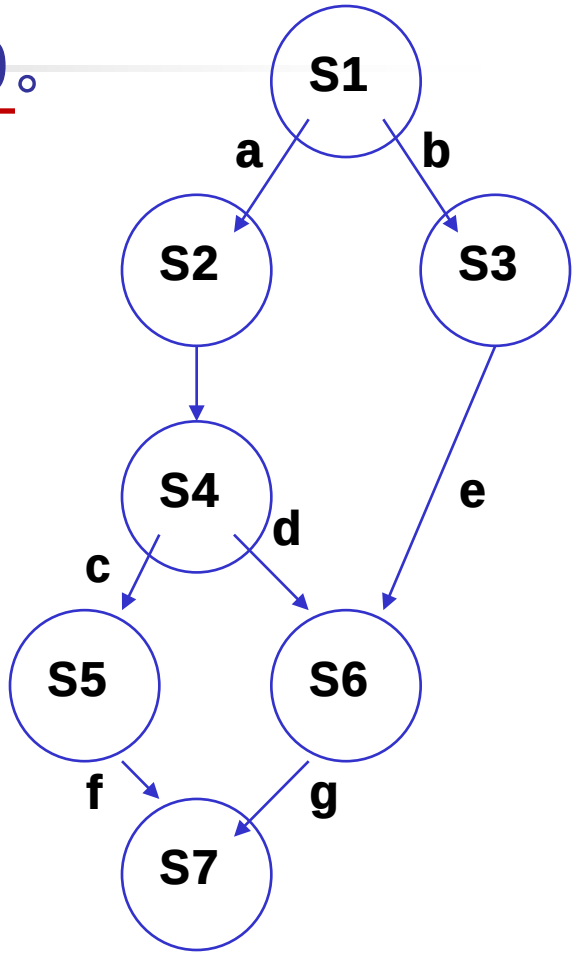
2.2.5 信号量（灯）机制

例：用P、V操作控制如图的程序同步

解：设7个信号量，其初值均设为0。

Var a,b,c,d,e,f,g:semapheres;

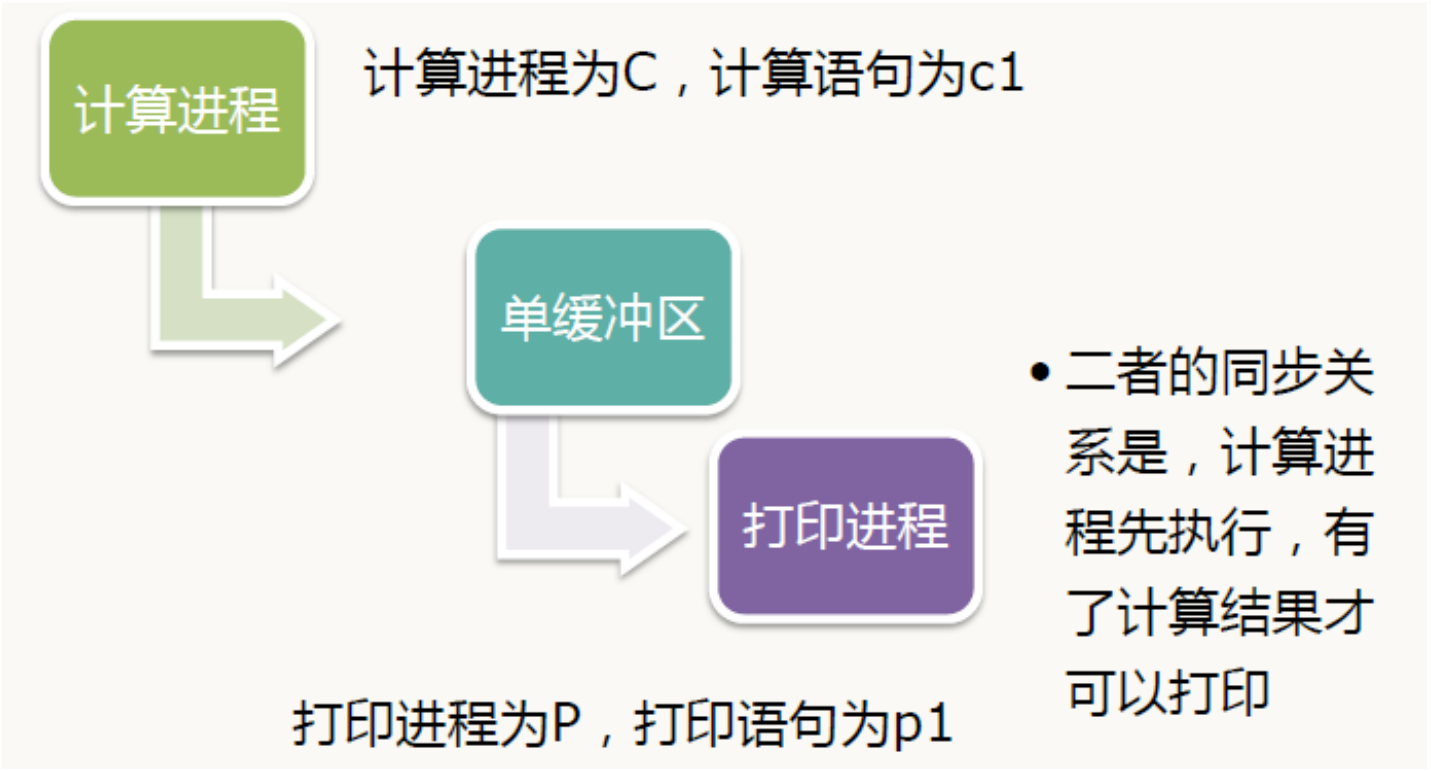
```
Begin
  cobegin
     begin s1;V(a);V(b);end
     begin P(a);s2;s4;V(c);V(d);end
     begin P(b);s3;V(e);end
     begin P(c);s5;V(f);end
     begin P(d); P(e); s6;V(g);end
     begin P(f); P(g); s7; end
  coend;
End;
```



61

2.2.5 信号量（灯）机制

- 设计一个同步方案解决计算进程与打印进程的同步



2.2.5 信号量（灯）机制

- 如何实现？一旦打印进程先于计算进程获得处理机，会发生什么？

同步信号量sm初值设置为0

值得注意的是，Wait操作和Signal操作使用不当，仍然会出现与时间有关的错误

```
Cobegin
Semophore  sm=0;
计算进程 C:                    打印进程 P:
Repeat                         Repeat
      c1;                          Wait（sm）;
      signal(sm);                    p1;
Until false                        signal(sm);
                               Until false
coend
```

信号量的应用

- 实现进程互斥
  - 例如：售票系统
- 实现进程同步
  - 例如：下棋问题

64

第二章　操作系统及进程管理

2.2进程及进程通信

- 2.2.1 进程的引入
- 2.2.2 进程描述及进程状态
- 2.2.3 进程控制
- 2.2.4 进程的同步与互斥
- 2.2.5 信号量机制
- 2.2.6 经典进程同步问题
- 2.2.7 进程通信

- 生产者-消费者问题
- 读者-写者问题
- 哲学家问题

2.2.6 经典进程同步问题

- 生产者-消费者问题
  - 生产者与消费者互斥访问公用数据缓冲区
  - 生产“数据”，消费“数据”
- 读者-写者问题
  - 数据文件或记录被多个进程共享并互斥访问的问题
  - 允许多个Reader同时访问，但不允许一个Writer和其它Reader或任何两个以上的Writer同时访问
- 哲学家就餐问题
  - 多资源共享及互斥访问
  - 五个哲学家的思考与互斥共享五根筷子就餐的问题

66

2.2.6 经典进程同步问题—生产者—消费者问题

- 代表两类对象共享资源，间接制约的关系
- 问题描述：一组生产者和一组消费者通过缓冲区进行通信。生产者不断（循环）将产品送入缓冲区，消费者不断（循环）从中取用产品。二类为并发进程：
- 同步问题：
  - 若缓冲区满，则生产者不能将产品送入
  - 若缓冲区已空，则消费者不能取得产品
- 互斥问题：
  - 同一时刻只能有一个人在仓库（缓冲区）

67

2.2.6 经典进程同步问题—生产者—消费者问题

- 问题解决
  - 两类并发进程：
    - 生产者，消费者
  - 共享资源：
    - 仓库→缓冲池
    - 临界资源，两类进程互斥访问
    - 两类进程同步推进
  - 设置同步信号量、互斥信号量

2.2.6 经典进程同步问题

信号量：
mutex = 1 控制互斥访问缓冲区
empty = n, full = 0 空/满缓冲区个数

生产者	消费者
repeat 生产出一个产品 **P(empty);** **P(mutex);** 把新产品放入缓冲区 **V(mutex);** **V(full);** **Until false**	**repeat** **P(full);** **P(mutex);** 从缓冲区取出一个产品 **V(mutex);** **V(empty);** 消费该产品 **Until false**

2.2.6 经典进程同步问题

```
Var mutex, empty, full : semaphore : = 1, n, 0;
buffer : array[0,…,n-1] of item;
in, out : integer : = 0, 0;
```

```
producer :
begin
  repeat
    producer an item nextp;
     P(empty);
     P(mutex);
     buffer(in):=nextp;
     in:=(in+1) mod n;
     V(mutex);
     V(full);
  until false;
end
```

```
consumer :
begin
  repeat
    P(full);
    P(mutex);
    nextp= buffer(out);
    out:=(out+1) mod n;
    V(mutex);
    V(empty);
    consume the item in nextc;
    until false;
end
```

2.2.6 经典进程同步问题

- P操作顺序不能改变！否则出现死锁！

```
producer :
begin
  repeat
    producer an item nextp;
    P(empty);
    P(mutex);
    buffer(in):=nextp;
    in:=(in+1) mod n;
    V(mutex);
    V(full);
  until false;
end
```

```
consumer :
begin
  repeat
    P(full);
    P(mutex);
    nextp= buffer(out);
    out:=(out+1) mod n;
    V(mutex);
    V(empty);
    consume the item in nextc;
    until false;
end
```

P(mutex);
P(empty);

死锁概念

- 并发进程竞争资源的最大问题——死锁
- 死锁概念的提出
  - 1965年，Dijkstra研究银行家算法时提出。
- 死锁（Deadlock）：
  - 是指两个或两个以上的进程在运行过程中，因争夺资源而造成的一种互相等待（谁也无法再继续推进）的现象，若无外力作用，它们都将无法推进下去。

产生死所原因

- 竞争资源
  - 系统中配备的非剥夺性资源的数量不能满足诸进程运行的需要时，会使进程因争夺资源而陷入僵局。
  - 系统资源
    - 可剥夺性资源：CPU、RAM等；
    - 非剥夺性资源：打印机、磁带机等；
    - 临时性资源：通信数据。
- 进程间推进顺序不当
  - 进程推进顺序合法——不会导致死锁
  - 进程推进顺序非法——可能会导致死锁

产生死锁的必要条件

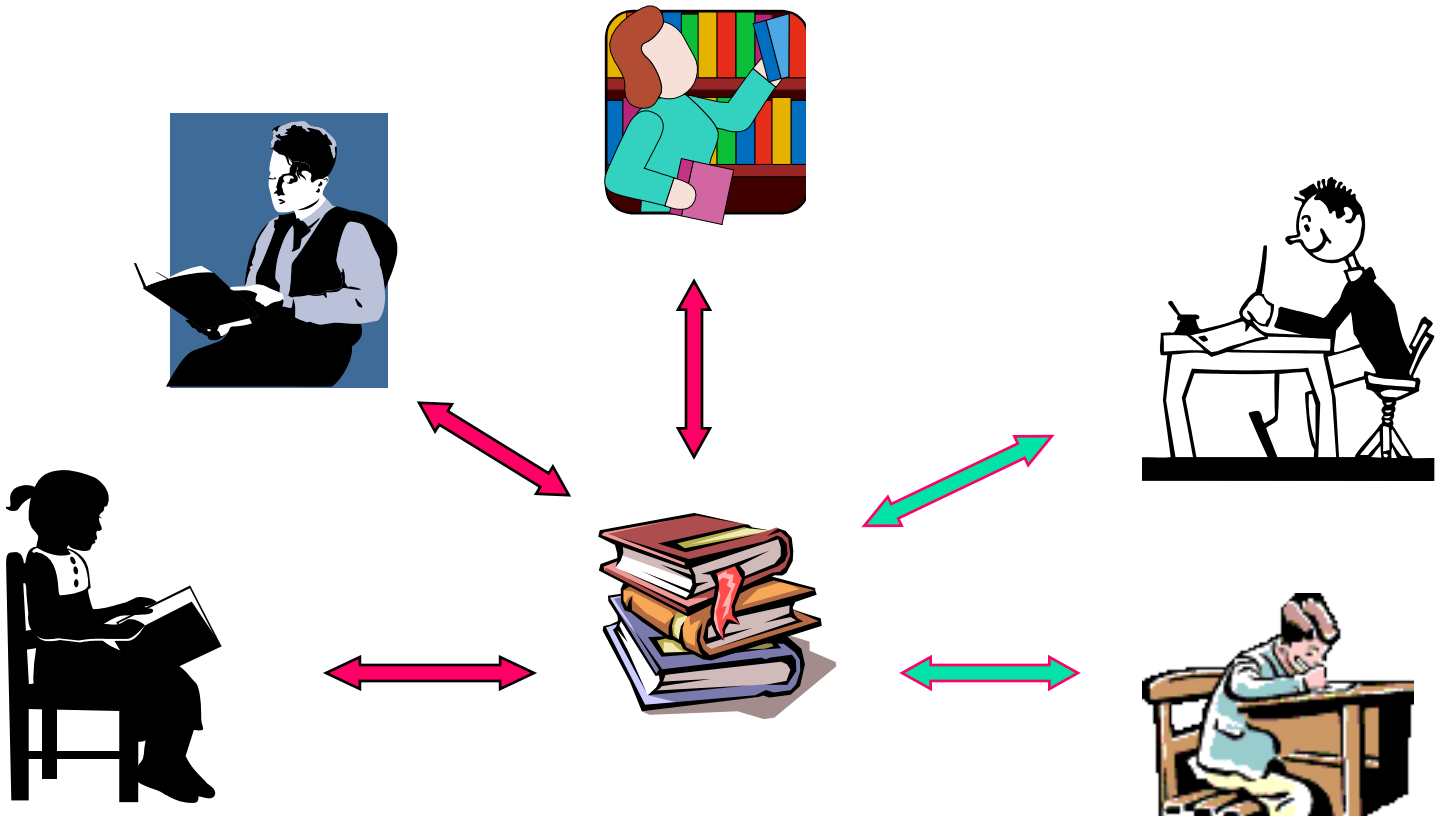
- 互斥条件
  - 一个资源一次只能被一个进程使用。
- 占有并请求条件（部分分配）
  - 保留已经得到的资源，还要求其它的资源。
- 不可剥夺条件（不可抢占）
  - 资源只能被占有者释放，不能被其它进程强行抢占
- 环路等待条件（循环等待）
  - 系统中的进程形成了环形的资源请求链。

2.2.6 经典进程同步问题—读者-写者问题

- 问题描述：
  - 一个数据文件或记录，可被多个进程共享
  - 只要求读该文件的进程称为“Reader进程”
  - 其它进程则称为“Writer进程”。
  - 允许多个进程同时读一个共享对象，但不允许一个Writer进程和其它Reader进程或Writer进程同时访问共享对象。
- “读者—写者问题”
  - 保证一个Writer进程必须与其它进程互斥地访问共享对象的同步问题。

75

2.2.6 经典进程同步问题—读者-写者问题



76

2.2.6 经典进程同步问题—读者-写者问题

- 问题分析
  - 读、写要互斥——设共享资源互斥信号量
  - 只要有一个进程读，则不允许写进程进入——设计数变量
  - 多个读进程都对计数变量操作——设计数变量互斥信号量

- Wmutex：互斥信号量，实现Reader与Writer进程间在读或写时的互斥
- RC：正在读的进程数
- Rmutex：互斥信号量，实现对RC访问的互斥。（RC是临界资源）

77

读者-写者临界区示意

信号量：wmutex,rmutex = 1 ,1
RC:integer=0;

Reader
repeat
wait(rmutex);
若 RC = 0, 则wait(wmutex);
RC加1;
signal(rmutex);
读数据对象
wait(rmutex);
RC减1;
若 RC = 0, 则signal(wmutex);
signal (rmutex);
until false;

Writer
repeat
wait(wmutex);
对数据对象写
signal(wmutex);
until false;

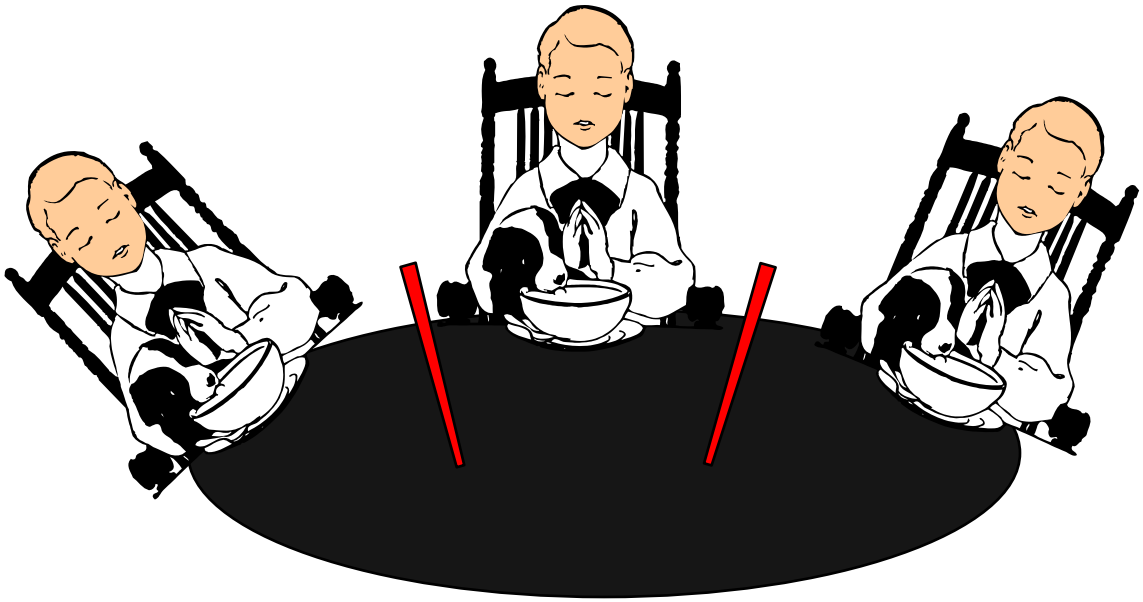
是否有人对RC操作？

无人在读，有人在写吗？

写完，释放文件

读完，无人读，释放文件

2.2.6 经典进程同步问题—哲学家就餐问题



79

2.2.6 经典进程同步问题—哲学家就餐问题

- 问题描述：
  - 有五个哲学家共用一张圆桌，分别坐在周围的五张椅子上，在圆桌上有五个碗和五只筷子，他们的生活方式是交替地进行思考和进餐。平时，一个哲学家进行思考，饥饿时便试图取用其左右最靠近他的筷子，只有在他拿到两只筷子时才能进餐。
- 问题分析
  - 五个进程：思考和就餐并发
  - 筷子是临界资源，在一段时间内只允许一位哲学家使用

80

2.2.6 经典进程同步问题—哲学家就餐问题

信号量定义：var chopstick :array[0,...,4] of semaphore;

第i位哲学家的活动描述：

```
repeat
        P(chopstick[i]);
        P(chopstick[(i+1) mod 5);
        ......
        eating;
        ......
        V(chopstick[i]);
        V (chopstick[(i+1) mod 5);
        ......
        thinking;
until false;
```


81

2.2.6 经典进程同步问题—哲学家就餐问题

遗留问题：

五位哲学家同时拿起左边的筷子，则因五位哲学家均在等待右边的筷子而使他们饥饿而“死”。

解决的办法如下：

1. 至多只允许四位哲学家同时去拿左边的筷子；
2. 仅当哲学家左右两边的筷子均可用时才允许他拿起筷子；
3. 规定奇数号哲学家先拿起他左边的筷子，而偶数号哲学家先拿起他右边的筷子。

其他进程同步问题分析

二人下棋、超市购物、单线轨道

82

第二章　操作系统及进程管理

2.2进程及进程通信

- 2.2.1 进程的引入
- 2.2.2 进程描述及进程状态
- 2.2.3 进程控制
- 2.2.4 进程的同步与互斥
- 2.2.5 信号量机制
- 2.2.6 经典进程同步问题
- 2.2.7 进程通信

- 共享存储器系统
- 消息传递系统

83

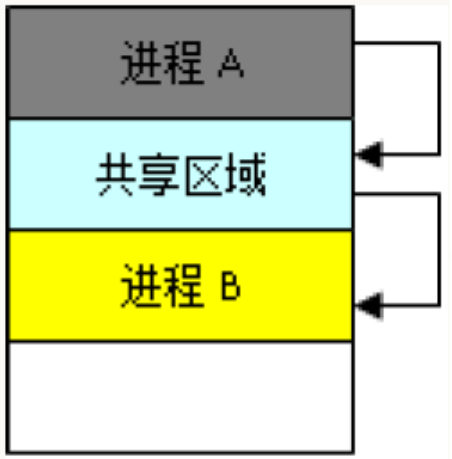
2.2.7 进程通信—

进程通信的概念和类型

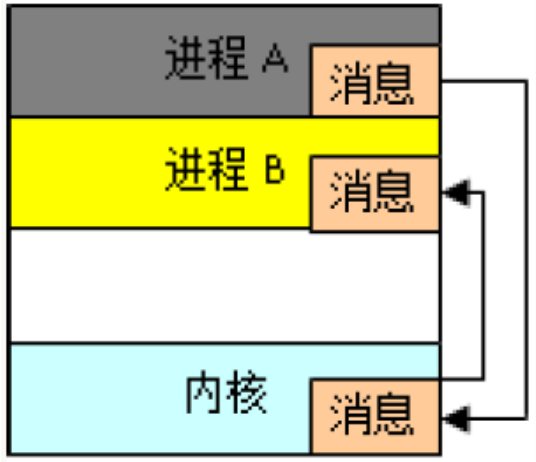
- 进程通信
  - 并发进程之间相互交换信息
- 两种级别：
  - 低级通信（互斥、同步）
    - 利用信号量机制实现进程间的数据传递。
    - 缺点：效率低；对用户不透明。
  - 高级通信（进程通信）
    - 进程之间利用OS提供的一组通信命令，高效地传送大量数据的信息交换方式。
    - 优点：高效；方便，简化了通信程序的设计。

2.2.7 进程通信

进程通信的类型



- 共享存储器系统
  - 基于共享数据结构的通信方式——低级
  - 基于共享存储区的通信方式——高级
- 消息传递系统
  - 消息（Message 报文）及相关的一组命令
  - 直接通信方式和间接通信方式



- 管道通信系统
  - 管道（Pipe文件）：用于连结一个读进程和一个写进程以实现他们之间通信的一个共享文件。
  - 双方进程的协调：互斥、同步、确定对方存在

85

2.2.7 进程通信—

消息传递通信的实现方法

- 直接通信方式
  - 源进程利用OS提供的发送命令原语，直接把消息发送给目标。
  - Send(Receiver,message)、Receive(Sender,message)
- 间接通信方式
  - 进程之间通过一个作为共享数据结构的中间实体——信箱，以消息暂存方式实现的通信。
  - 操作原语：信箱的创建、撤消；消息的发送、接收。
  - 信箱的创建和拥有者：OS或用户（通信）进程。
  - 信箱的种类：
    - 私用信箱、公用信箱、共享信箱
  - 利用信箱通信的进程之间的关系
    - 一对一、多对一、一对多、多对多

86

2.2节作业

- 1.试说明进程在三个基本状态之间转换的典型原因。
- 2.试说明PCB的作用。
- 3.进程在运行时，存在哪两种形式的制约？举例说明。
- 4.举例说明生产者-消费者的问题。