第3章 进程管理

进程的概念是操作系统中最重要的概念，它主要用来描述系统中并发程序的执行行为。进程是系统资源分配的基本单位。

教学要求

◆牢固掌握进程的概念，深入理解进程最基本的属性是动态性和并发性。

◆掌握进程与程序的主要区别。

◆掌握进程的基本状态及其转换条件，理解进程的一般组成，深入理解进程控制块的作用。

◆掌握进程同步与互斥的概念，掌握进程临界资源和临界区的概念，理解进入临界区的原则。

◆理解信号量概念，P、V操作执行的动作。能用信号量和P、V操作实现简单的进程互斥或同步。

教学内容

3.1 进程的引入

3.2 进程的结构

3.3 进程控制

3.4 进程的同步与互斥

3.5 进程间通信

3.6 进程调度

3.7 死锁

3.8 线程

3.1 进程的引入

3.1.1 顺序程序与并发程序

1. 顺序程序

是指严格按照写入的顺序执行的程序。顺序程序执行时具有以下特征：

- 顺序性
- 封闭性
- 可再现性

2. 并发程序

并发程序是指两道或两道以上程序同时装入内存中运行，这些程序的执行在时间上互相有重叠，即在一个程序执行结束之前，另一个程序已经开始执行。

并发程序的特征：

- 间断性
- 失去封闭性
- 不可再现性

引入进程原因总结

在多道环境下，一个程序活动不再能独占系统资源，不能单独决定这些资源的状态。程序和程序活动之间也不再有一一对应关系。程序活动不再处于一个封闭的系统中，而是和其它程序活动之间存在着相互依赖和制约的关系，因而呈现出并发、动态以及相互制约这些新的特征。在这种情况下，程序这个静态的概念已经不能如实地反映多道系统中程序的并发活动，故引入了进程的概念来描述系统和用户的程序活动。

3.1.2 进程的定义与特性

1、定义：进程是指可并发执行的程序，在一个数据集合上的一次运行过程。

2、特征

- 动态性：进程是程序执行的过程。
- 并发性：进程使程序能并发执行。
- 异步性：进程以各自独立的、不可预知的速度向前推进
- 独立性：进程是独立运行的基本单位，也是系统资源分配与调度的独立单位。
- 结构性：进程包括可执行的程序代码、程序的数据和堆栈、程序计数器、堆栈指针、有关寄存器、以及所有运行程序所必须的其它信息。

3.1.3 进程与程序的关系

- 动态性和静态性。进程是动态的，程序是静态的。
- 临时性和永久性。进程是临时的，程序是永久的；进程是一个状态变化的过程，而程序可长久保存。
- 并发性与顺序性。
- 组成上的不同。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（PCB）。
- 多对一的关系。通过多次执行，一个程序可对应多个进程；但一个进程只能对应一个程序。

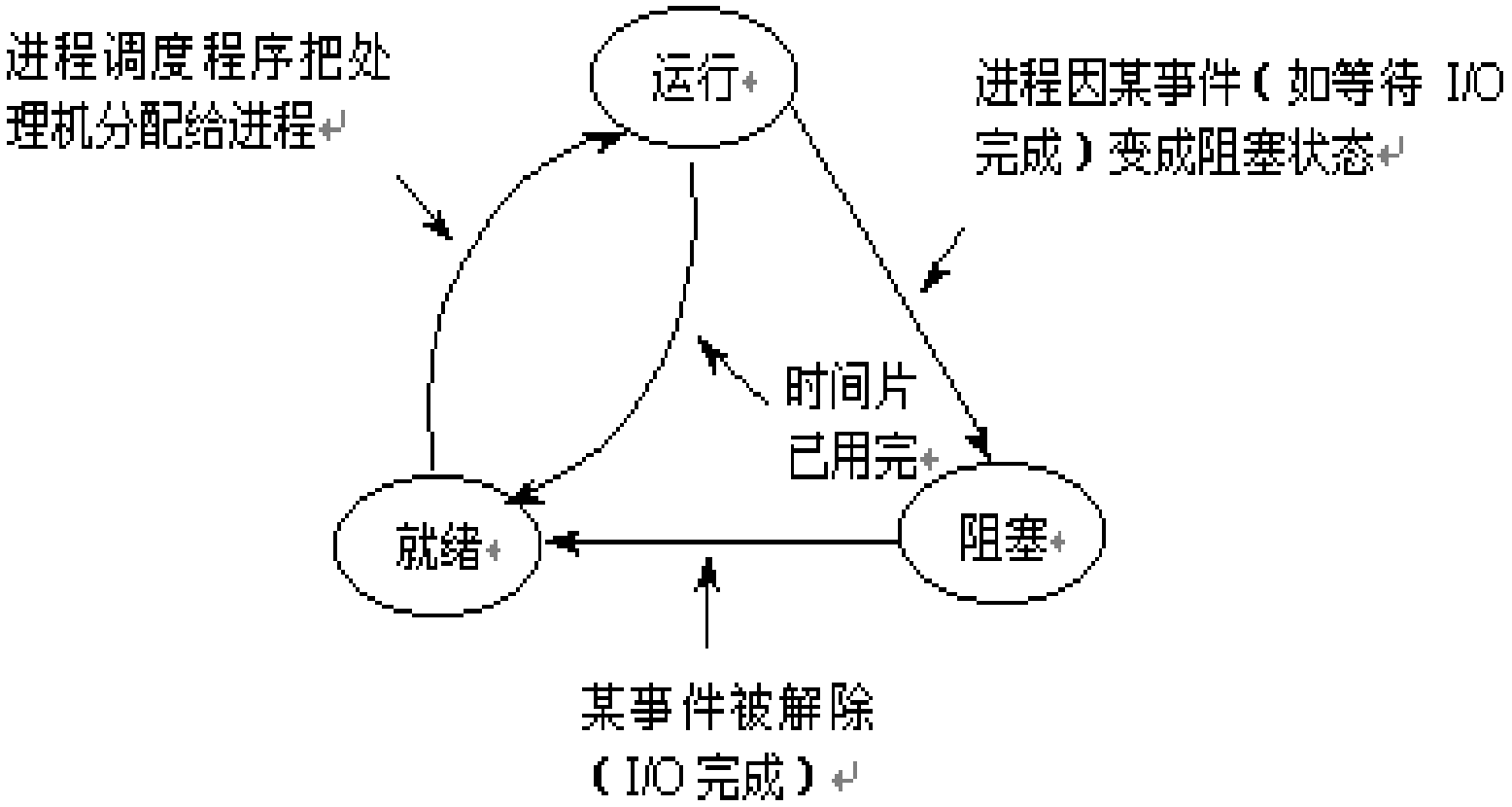
3.1.4 进程的状态及其转换

1、基本状态（三状态）

- 就绪态。
- 运行态。
- 阻塞态（等待态、睡眠态）

在五状态模型中，增加了创建态和终止态

2、 状态的转换


进程调度程序把处理机分配给进程
运行
进程因某事件（如等待 I/O 完成）变成阻塞状态
时间片已用完
就绪
阻塞
某事件被解除（I/O 完成）

3.2 进程的结构

3.2.1 进程的实体

在操作系统中，一个进程是通过其物理实体被感知的，进程的物理实体又称为进程的静态描述。进程的静态描述由三部分组成：

3.2 进程的结构（续）

- 程序：描述了进程所要完成的功能；
- 数据：进程运行所需要的数据和工作区；
- 进程控制块(PCB)：它包含了进程的描述信息、控制信息和资源信息，是进程动态特性的集中反映，是进程存在的唯一标志；

3.2.2 PCB的内容

1、进程描述信息：

- 进程标识符(process ID,PID)唯一，通常是一个整数
- 进程名，通常基于可执行文件名（不唯一）
- 用户标识符(user ID)；进程组关系

3.2.2 PCB的内容（续）

2、进程控制信息：

- 当前状态
- 优先级(priority)
- 代码执行入口地址
- 程序的外存地址
- 运行统计信息（执行时间、页面调度）
- 进程间同步和通信；阻塞原因
- 进程的队列指针
- 进程的消息队列指针

3.2.2 PCB的内容（续）

3、所拥有的资源和使用情况：

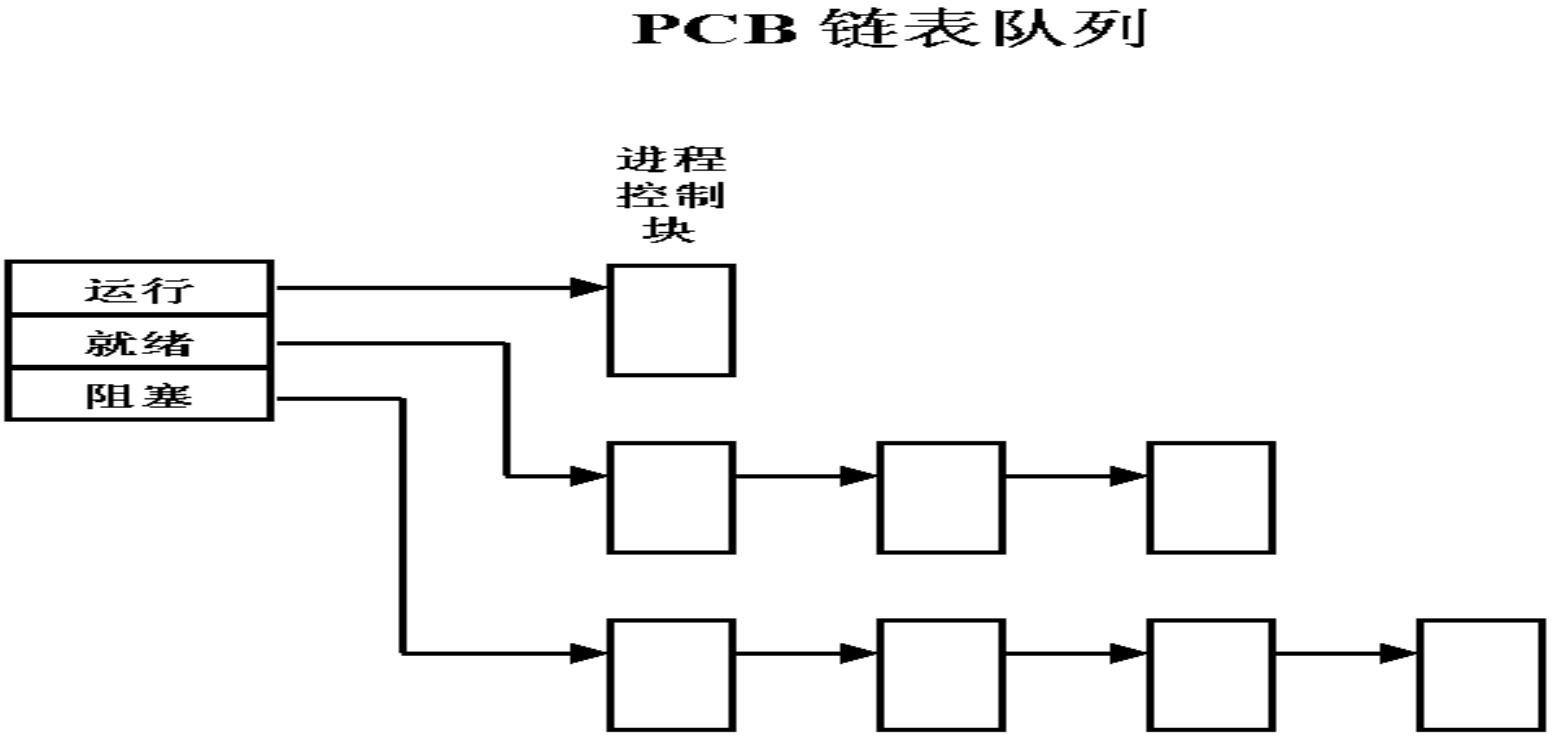
- 虚拟地址空间的现状
- 打开文件列表

4、CPU现场保护信息：

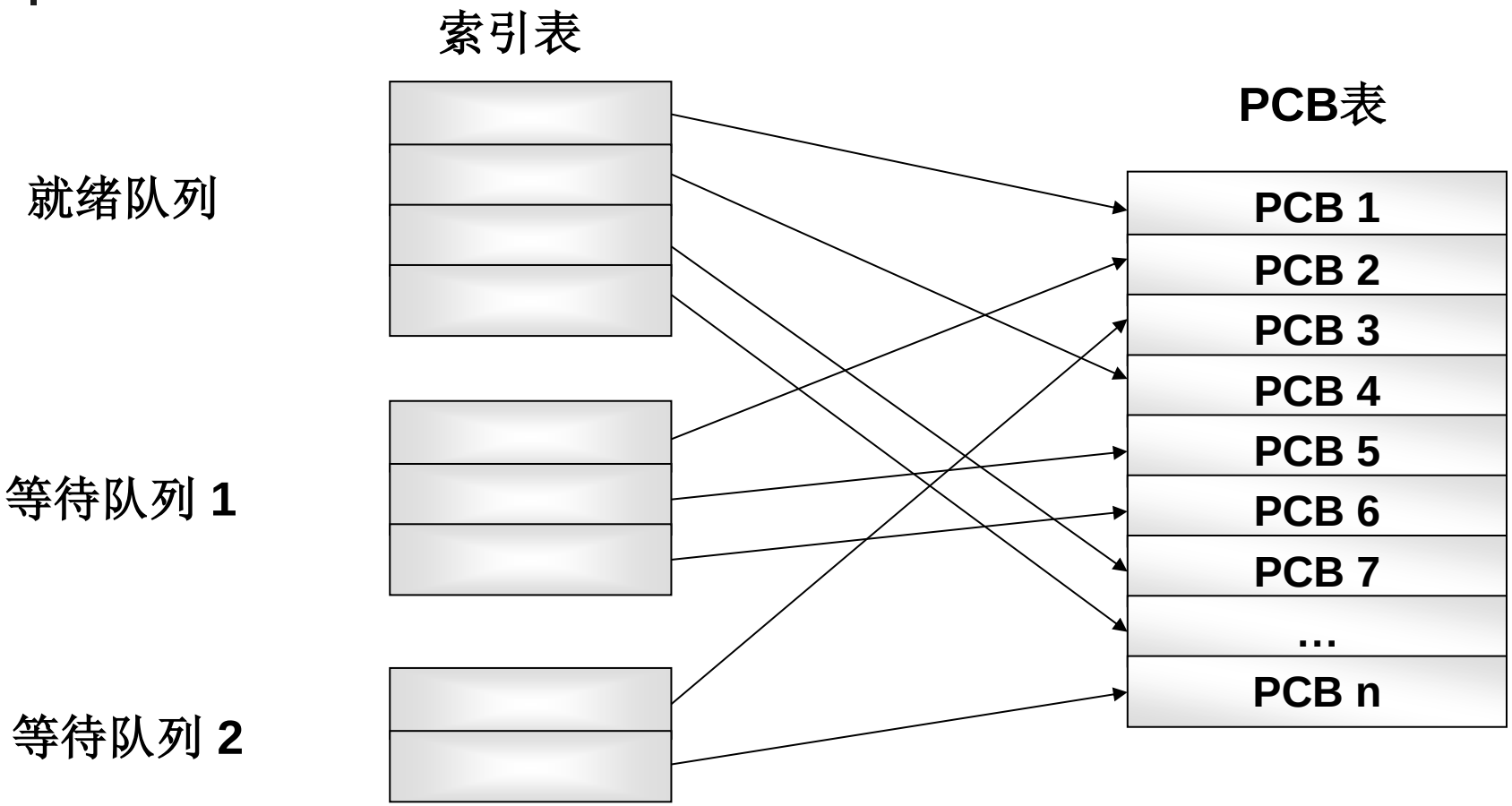
- 寄存器值（通用、程序计数器**PC**、状态**PSW**，地址包括栈指针）
- 指向赋予该进程的段/页表的指针

3.2.3 PCB的组织

1、链接结构



2、索引结构


索引表
PCB表
就绪队列
等待队列 1
等待队列 2
PCB 1
PCB 2
PCB 3
PCB 4
PCB 5
PCB 6
PCB 7
…
PCB n

3.3 进程控制

操作系统使用系统原语(primitive)控制进程状态改变，原语被认为是机器语言的延伸，是完成特定任务的一段内核基本代码。

特点：

- 原子操作性(atomic operate)
- 不能直接使用，需通过特殊的系统调用来使用原语。

3.3.1 进程的创建与撤销

1、引起创建进程的事件

- 由系统创建的事件
- 由存在的进程创建的事件

3.3.1 进程的创建与撤销（续）

父进程：创建子进程的进程

子进程：被创建的进程

进程树：反映进程家族关系的图

2、进程的创建

创建过程如下：

- 申请空白PCB
- 为新进程分配资源
- 初始化PCB
- 将新进程插入就绪队列

3、进程的撤销

- 进程撤销的原因
- 进程撤销的过程

3.3.2 进程的阻塞与唤醒

1、进程的阻塞

处于运行状态的进程，如果要等待某事件的发生，如等待资源、等待I/O完成等，无法继续执行，这时进程自己调用阻塞原语，把自己变成阻塞状态。

3.3.2 进程的阻塞与唤醒（续）

操作过程

(1) 停止进程执行，把CPU现场信息保存到该进程的PCB的相应表目中。

(2) 修改PCB的有关表目内容，如把进程状态由运行状态改为阻塞状态。

(3) 根据阻塞原因，把修改后的PCB插入到相应的阻塞队列中。

(4) 转入进程调度程序，从就绪队列中重新调度其他进程运行。

2、进程的唤醒

当处于阻塞状态的进程等待的事件已经结束，这时就由完成等待事件的进程（如释放资源的进程或完成I/O的进程）调用唤醒原语，将该阻塞进程唤醒，转换成就绪状态。具体步骤

3.3.2 进程的阻塞与唤醒（续）

唤醒过程如下：

- 把PCB从相应的等待队列中移出。
- 修改PCB的有关表目内容，如把进程状态由阻塞状态改为就绪状态。
- 将修改后的PCB插入到就绪队列中，等待调度。

3.3.3 Linux进程控制

- 进程的创建： pid=fork()

Pid的返回值有两个：

0：子进程占用CPU

正整数：父进程占用CPU

注意：除非子进程加载执行其他代码，否则子进程要继承父进程中从创建语句以下的所有代码。

Linux进程创建示例

```
Main()
{   int pid;
    Pid=fork();
    If (pid)
        Printf("it is parent process\n");
    Else
        Printf("it is child process\n");
        Printf("it is end\n");}
```

下面是输出的一种结果，为什么？

It is parent process

It is end

It is child process

It is end

3.3.3 Linux进程控制（续）

- 进程等待： **wait()**
- 进程执行： **exec()**
- 进程终止： **exit()**

3.4 进程的同步与互斥

3.4.1 基本概念

1、进程同步：对多个进程执行操作的时间顺序所加的某种限制，如：操作A应在操作B之前进行。

它主要源于进程的合作，是进程之间因直接制约而形成的互相合作，互相等待的关系。

如：公交运输中司机驾驶和售票员开、关车门；教学管理中的学生考试与教师改卷

3.4.1 基本概念（续）

2、进程互斥：同步的特例，进程之间的多个操作决不能同时执行，如：操作A和操作B不能同时执行，但没有规定谁先谁后的问题。

它主要源于对资源的竞争，是进程之间因间接制约而排它性使用资源的关系。

如：学生和教师都去借阅图书；多个乘客购买车票

3.4.1 基本概念（续）

3、临界资源(critical resource)：一次只允许一个进程使用的资源。

临界资源可能是硬件，如打印机；也可能是软件：变量，数据，表格，队列等。

4、临界区：包含临界资源的代码段。

3.4.1 基本概念（续）

并发进程对临界区的访问必须作某种限制，否则就可能出现与时间有关的错误。

【例】假设一个民航售票系统有两个终端，分别运行进程**P1**和**P2**。该系统的公共数据区中的某个单元**x**存放某次航班的余票数，**R1**和**R2**表示进程**P1**和**P2**执行时各自所用的工作单元。

进程P1的算法：

```
P1()
    {int R1;
    {按旅客订票要求找到x};
       R1=x;
    if R1>=1 then
   {R1=R1-1; x=R1; 输出一张票; }
    else {输出票已售完; }}
```

进程P2的算法：

```
P2()
{int R2;
{按旅客订票要求找到x};
        R2=x;
if R2>=1 then
  {R2=R2-1; x=R2; 输出一张票; }
else {输出票已售完; }}
```

假设单元x的当前值为u，则可能出现如下所示的运行情况：

P1：R1=u；

P2：R2=u；

P2：R2=R2-1；x=R2；输出一张票；x=u-1；

P1：R1=R1-1；x=R1；输出一张票；x=u-1；

两次售票后，票的余数为u-1

访问临界区的准则

为了禁止两个进程同时进入临界区内,可以采用软件办法或系统提供的同步机构来协调它们的关系。但必须遵循下述准则：

- 空闲则进
- 忙则等待
- 有限等待
- 让权等待

【注意】临界区是对某一临界资源而言的，对于不同临界资源的临界区，它们之间不存在互斥。如有程序段A、B是关于变量X的临界区，而C、D是关于变量Y的临界区，那么，A、B之间需要互斥执行，C、D之间也要互斥执行，而A与C、B与D之间不用互斥执行。

3.4.2 实现进程互斥的硬件方法

1、“开关中断”指令

关中断—临界区—开中断

2、测试与设置指令TS

测试布尔变量**lock**的状态表示临界区是否可用，具体通过**lock**和**unlock**原语实现

3、交换指令Swap

设置全局布尔变量**lock**和局部布尔变量**key**，当**lock＝true and key＝true**时进入临界区。

3.4.3 信号量与P、V操作

1、什么是信号量

来自于信号灯的概念，信号量有多种类型，常用的是记录型信号量，可定义如下：

```
struct   semaphore
{ int value;    /*value的初值为非负变量*/
  int *Q;       /*Q是初始为空的等待队列*/
 }S;
```

2、P、V操作(wait、signal)

P(S)

```
    S.value=S.value-1;
if (S.value<0)
    {
当前进程进入等待队列Q;
阻塞当前进程;
    }
else
当前进程继续;
```

3.4.3 信号量（续）

V(S)

```
    S.value=S.value+1;
if (S.value<=0)
    {从等待队列Q中取出一个进程P,进程
  P进入就绪队列;
    当前进程继续;
}
else
当前进程继续;
```

信号量与P、V操作的物理意义

当S.value>0，表示可用资源个数；

当S.value<0，|S.value|表示等待该信号量的进程个数。

P(S)：表示申请一个资源，常用于阻塞进程

V(S)：表示释放一个资源，常用于唤醒进程

信号量的初值应该大于等于0

信号量物理含义举例

设有m个学生准备到图书馆去借书，每人一次借一本，该图书馆共有n本书，且n<m。此时，可用信号量book来表示图书馆中的图书，其初值为n。

- 借书:P(book)，若book>=0，有书可借；反之，该学生就不能借书，需要等其他学生还书后才能借。此时|book|=需等待借书的学生人数。
- 还书:V(book)，表示可以借出的书的数量多了1本，若此时book<=0，有学生正在等待借书，应允许一个学生借书。

3、用信号量实现互斥模型

为临界资源设置一个互斥信号量S，其初值为1；在每个进程中将临界区代码置于P(S)和V(S)原语之间。必须成对使用P和V原语。

互斥模型如下：

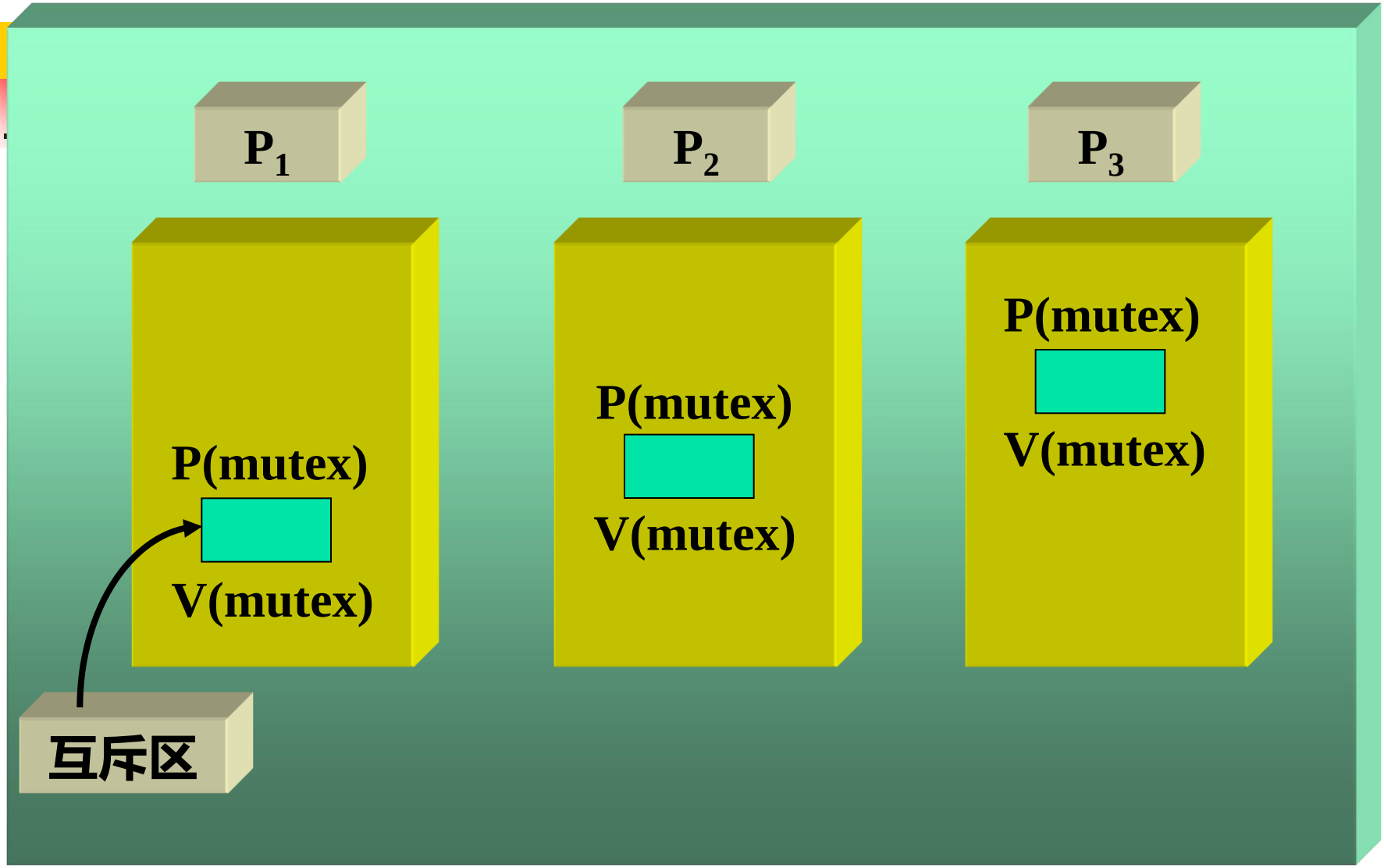
P（S）

临界段；

V（S）

剩余代码；

用信号量实现互斥模型



4、用信号量实现同步模型

为进程设置一个同步信号量S，其初值为0；在进程需要同步的地方分别插入P(S)和V(S)原语。一个进程使用P原语时，则另一进程往往使用V原语与之对应，具体怎么使用要看实际情况来定。

3.4.3 信号量（续）

例如，有两个进程P1、P2，P1的功能是计算x=a+b的值，a和b是常量，在P1的前面代码中能得到；P2的功能是计算y=x+1的值。

P1	P2
…	…
X=a+b;	P(S)
V(S)	Y=X+1;
…	…

【思考题1】

桌上有一空盘，最多允许存放一只水果。父亲每次可向盘中放一个苹果，母亲每次可向盘中放一个桔子，儿子专等吃盘中的桔子，女儿专等吃苹果。

试用**P**、**V**操作实现父亲、母亲、儿子和女儿四个并发进程的同步。

【解答】

设置三个信号量S，So，Sa ，初值分别为1，0，0。其中，S用于父母向盘中放水果时互斥，So用于母子同步，Sa用于父女同步。

四个进程的同步过程如下：

Father()	Mather()	Son()	Daughter()
p(S)；	p(S)；	p(So)；	p(Sa)；
放苹果；	放桔子；	取桔子；	取苹果；
v(Sa)；	v(So)；	v(S)；	v(S)；

【思考题2】

对于N个进程实现互斥的模型中，信号量的取值范围是什么，有什么含义。

【解答】

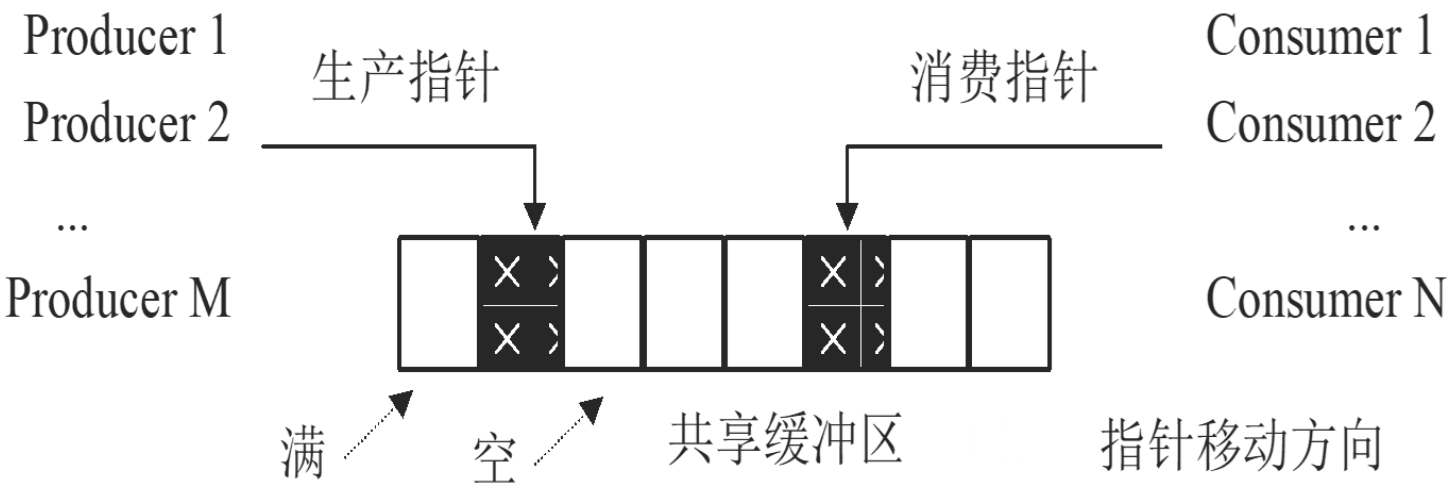
- 范围是：－(N－1)～1间的整数
- 若S＝1，表示没有进程进入临界区
- 若S＝0，表示有一个进程进入临界区,无等待
- 若S＝-t，表示一个进程进入临界区，另外t个进程等待进入。

3.4.4 经典同步互斥问题

1、生产者－消费者问题

生产者-消费者问题是最著名的同步问题，它描述一组生产者进程向一组消费者进程提供消息。它们共享一个有界缓冲池,生产者向其中投放消息，消费者从中取得消息。

生产者—消费者问题


Producer 1
Producer 2
...
Producer M
生产指针
消费指针
Consumer 1
Consumer 2
...
Consumer N
满
空
共享缓冲区
指针移动方向

生产者—消费者问题（续）

生产者和消费者之间应满足下列条件：

- 任何两个进程必须互斥使用缓冲池；
- 只有在缓冲池中至少有一个缓冲区已存入消息后，消费者才能从中提取消息，否则消费者必须等待。
- 只有缓冲池中至少有一个缓冲区是空时，生产者才能把消息放入缓冲区，否则生产者必须等待。

【解答分析】

- 根据题意，能套模型的首先考虑模型问题；
- 定义信号量时一定要与资源对应；因此，需要确定每个进程的资源类型；
- 本题中，生产者的资源是空缓冲区，用**empty**表示；消费者的资源是存入缓冲池中的消息，用**full**表示；用**mutex**用于进程间互斥。

生产者—消费者问题（续）

empty=n, full=0, mutex=1

生产者	消费者
P(empty);	P(full);
P(mutex);	P(mutex);
往buffer中放入一条消息;	从buffer中取出一条消息;
V(mutex);	V(mutex);
V(full);	V(empty);

【思考题3】在前面介绍的水果问题中，如果盘子中允许存放2个水果时，该怎么处理？

【提示】考虑两个生产者-消费者问题。信息量怎么去定义？需要4个信号量：empty=2,fulla=0,fullo=0,metux=1

2、读者—写者问题

【问题描述】有两组并发进程：读者和写者，共享一组数据区。

要求：

1、允许多个读者同时读

2、读者与写者要互斥

3、写者与写者要互斥

【分析】

- 由于读者与写者、写者之间互斥，可定义一个互斥信号量**wrt**，初值为**1**；
- 只要有读者在时，写者总被阻塞，仅当所有的读者都离开后，才能允许写者写；因此，必须用一个变量**readcount**来统计当时的读者个数；
- 由于读者的到来或离开都要改变**readcount**的值，为确保**readcount**的正确性，因此，对它的计算也要互斥，用信号量**mutex**来实现，初值为**1**；

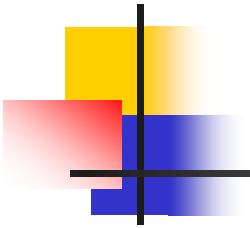
读者—写者问题（续）

写者

P(wrt)

写入数据；

V(wrt)



读者

```
P(mutex)
readcount=readcount+1
If  readcount=1 then  P (wrt)
V(mutex)
  读数据；
P(mutex)
 readcount=readcount-1
if readcount=0 then  V(wrt)
  V (mutex)
```

读者—写者问题（续）

【思考题4】修改以上读者写者问题的算法，使之对写者优先，即一旦有写者到达，后续的读者必须等待，无论是否有读者在读。

【提示】增加一个信号量S，用于在写者到达后封锁后续的读者。

读者：

P(S)

P(mutex)

readcount=readcount+1

if (readcount==1) P (wrt)

V(mutex)

V(S)

读数据；

P(mutex)

readcount=readcount-1

if (readcount==0) V(wrt)

V(mutex)

写者：

P(S)

P(wrt)

写数据；

V(wrt)

V(S)

读者—写者问题（续）

【讨论题】若规定读者能同时读的数量不超过6个，请修改读者写者问题的算法。

【提示】增加一个信号量S，初值为6，用于阻塞超过6个之后的读者。

读者：

```
P(S)
P(mutex)
    readcount=readcount+1
 if (readcount==1)   P (wrt)
V(mutex)
    读数据;
P(mutex)
    readcount=readcount-1
    if (readcount==0) V(wrt)
V(mutex)
 V(S)
```

【思考题5】有条AB段的单车道，在AB段上没有车辆时，先到达AB段的同方向车辆可以通过。当反方向出现等待车辆时，未到达AB段的同方向车辆禁止通行。同样，反方向的车辆通过时，也按照上述规则进行。请用信号量和P,V操作实现对AB路段的交通管制。

3.5 进程通信

3.5.1 进程通信类型

- 低级通信：交换信息量少，如P、V原语，软中断信号；
- 高级通信：交换信息量大，有消息缓冲、信箱方式、共享存储区、管道通信；

3.5.2 消息缓冲

1、基本原理：

进程间的数据交换以消息为单位，消息存放在缓冲区中。

- 发送进程利用发送原语把载有消息的缓冲区挂接到接收进程的消息缓冲队列上。
- 接收进程利用接收原语直接从消息缓冲队列中取得消息。

消息缓冲区结构

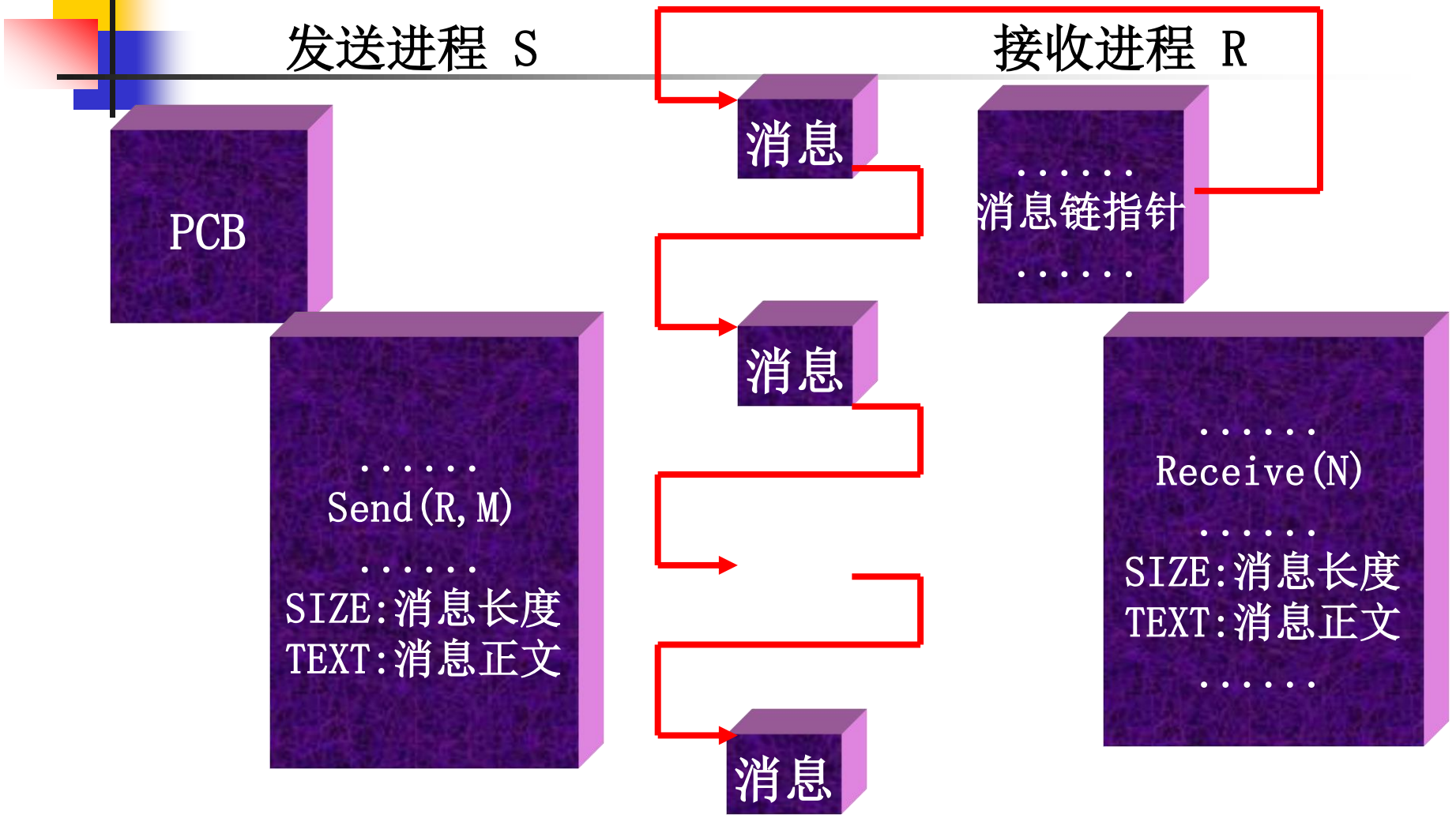
```
struct msg_buf
       { int sender;    //发送者ID
         int size;      //消息长度
       char text[30]; //消息正文
       struct msg_buf *next;//消息队列指针
  }
```

3.5.2 消息缓冲（续）

2、发送与接收原语

- 发送原语：Send(Receiver,message)
- 接收原语：Receive(Sender，message)

3、消息缓冲示例


发送进程 S
接收进程 R
PCB
消息
消息
消息
.....
消息链指针
.....
.....
Send(R,M)
.....
SIZE:消息长度
TEXT:消息正文
.....
Receive(N)
.....
SIZE:消息长度
TEXT:消息正文
.....

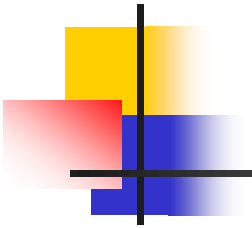
3.5.3 信箱方式

- 信箱方式是一种间接通信方式，通过信箱实现

  （消息缓冲是直接通信方式，要求通信双方都在场，同时处于通信状态）

- 通信原语

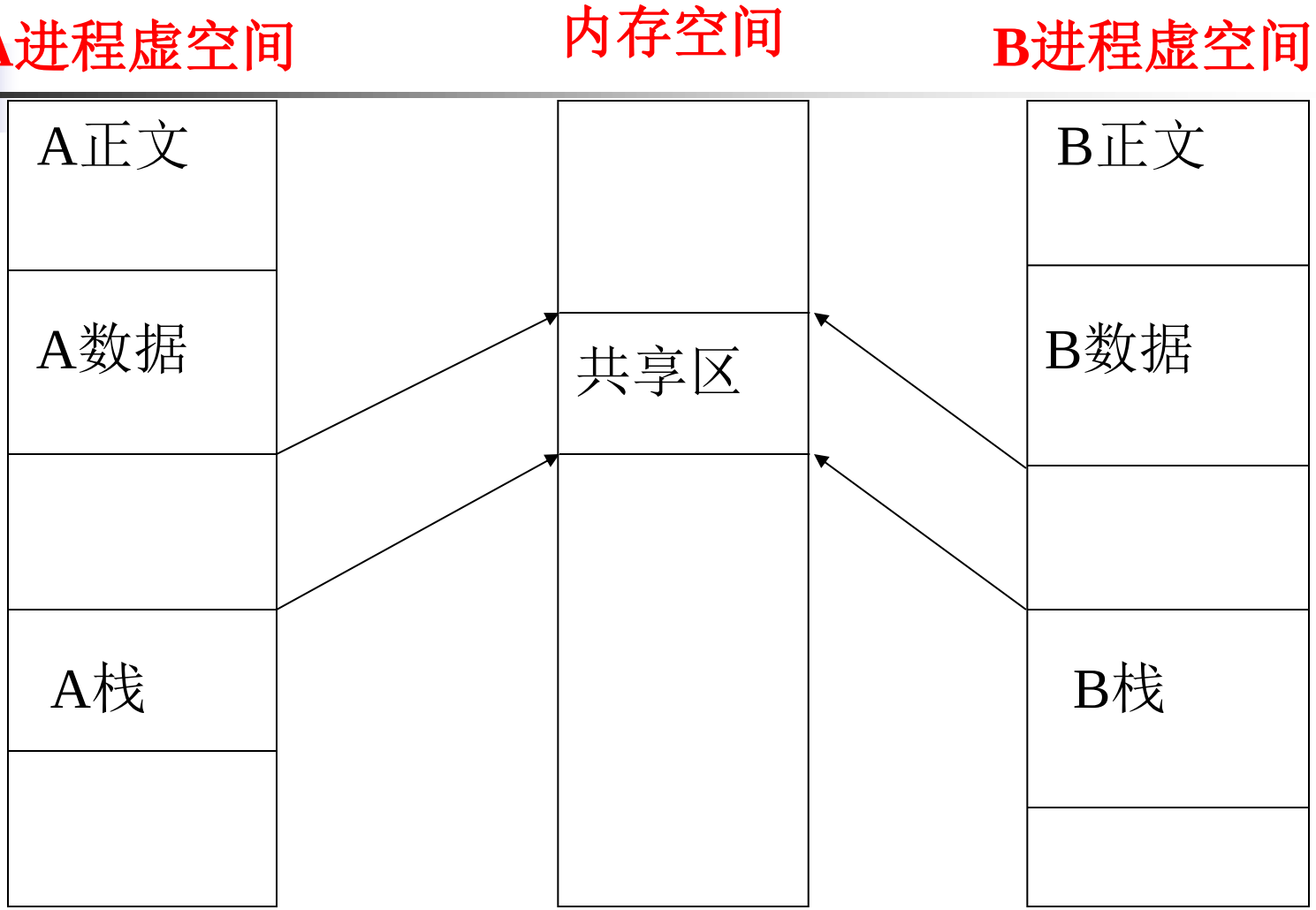
  发送原语：Send(Mailbox, message)；
  接收原语：Receive(Mailbox, message)；



- 信箱是存放信件的存储区域，每个信箱可分成信箱头和信箱体两部分。
- 信箱头指出信箱容量、信件格式、指针等；信箱体用来存放信件

- 发送信件：
  如果指定信箱未满，则将信件送入信箱中由指针所指示的位置，并释放等待该信箱中信件的等待者；否则发送信件者被置成等待信箱状态。
- 接收信件：
  如果指定信箱中有信，则取出一封信件，并释放等待信箱的等待者，否则接收信件者被置成等待信箱中信件的状态。

3.5.4 共享存储区



3.5.5 管道通信

管道是指用于连接一个读进程和一个写进程，以实现它们之间通信的共享文件。向管道提供输入的发送进程（即写进程），以字符形式将大量的数据送入管道，而接收管道输出的接收进程（即读进程）可从管道中接收数据。管道通信是基于文件系统形式的一种通信方式。

3.5.5 管道通信（续）

字符流方式写入读出
先进先出顺序

3.5.5 管道通信（续）

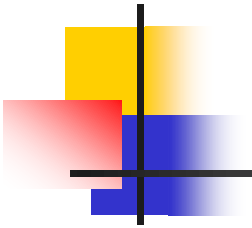
- 管道分为无名管道和有名管道两种类型。
- 无名管道是一个只存在于打开文件机构中的一个临时文件，从结构上没有文件路径名，不占用文件目录项。

3.5.5 管道通信（续）

- 在UNIX中，无名管道利用系统调用pipe(fd)创建，它返回两个文件描述符:写文件描述符fd[1]和读文件描述符fd[0]。无名管道也称为pipe文件。

```
int fd[2];
pipe(fd);
```

- 读管道：read(fd[0],buf1,n)
- 写管道：write(fd[1],buf2,n)



【例】建立一个管道，同时父进程生成一个子进程，父进程向管道中写一字符串，子进程从中读出该字符串。

```
#include<stdio.h>
#define msgsize 15
char *msg=”hello,China”;
  main( )
  {
       char buf[msgsize];
      int fd[2],j,pid;
      /*open pipe*/
 if(pipe(fd)<0) {
  perror(“管道创建失败！”); exit(1);}
      if((pid=fork( )<0) {
     perror(“进程创建失败！”); exit(2);
      }
```

```
/*父进程把信息写入管道*/
        if(pid>0) {
        close(fd[0]);
        write(fd[1],msg,msgsize);
        wait((int*)0);}
/*子进程从管道中读出信息*/
     if (pid==0)
   {  close(fd[1]);
      read(fd[0],buf,msgsize);
      printf("%s\n,buf"); } }
      exit(0);}
```

3.6 进程调度

3.6.1 进程调度的职能

进程调度的任务是控制协调进程对CPU的竞争，即按一定的调度算法从就绪队列中选中一个进程，把CPU的使用权交给被选中的进程。

3.6.2 进程调度方式

- 剥夺方式（抢占式调度）

  当有比正在运行的进程优先级更高的进程就绪时，系统可强行剥夺正在运行进程的CPU，提供给具有更高优先级的进程使用；

- 非剥夺方式（非抢占式调度）

  某一进程被调度运行后，除非由于它自身的原因不能运行，否则一直运行下去；

3.6.3 进程调度时机

- 当一个进程运行完毕，或由于某种错误而终止运行
- 当一个进程在运行时变为等待状态（等待I/O）
- 分时系统中时间片到
- 当有一个优先级更高的进程就绪（可抢占式）

  例如：新创建一个进程；一个等待进程变成就绪
- 在进程通信中，执行中的进程执行了某种原语操作（P操作，阻塞原语）

3.6.4 进程调度算法

- 先来先服务调度算法
- 短进程优先调度算法
- 优先级调度算法
- 时间片轮转调度算法
- 多级反馈队列调度算法

3.6.4 进程调度算法（续）

【例】一个具有两道作业的批处理系统，作业调度采用短作业优先，进程调度采用基于优先数的抢占式调度算法。在下表所示的作业序列，优先数为进程优先数，优先数越小优先级越高。

(1)列出所有作业进入内存时间及结束时间。

(2)计算平均周转时间。

作业名	到达时间	运行时间	优先数
J1	9：00	40分	4
J2	9：20	30分	2
J3	9：30	50分	3
J4	9：50	20分	5

解答：关键时间点：9:50（作业调度）

作业名	进入内存	完成时间	周转时间
J1	9:00	10:10	70
J2	9:20	9:50	30
J3	10:10	11:00	90
J4	9:50	11:20	90

3.7 死锁

本节主要介绍死锁（**deadlock**）的概念及其产生的原因，提出死锁的预防、避免和检测方法，以及在死锁发生后该如何解除和进行系统恢复。

3.7.1 死锁的概念

1、死锁的定义

死锁是指多个进程因竞争使用资源而形成的一个僵局(deadly embrace)。在这个僵局中，每个进程都占有一定资源，都申请使用已被另一进程占用且不能剥夺的资源，所有这些进程都进入阻塞状态而不能继续运行。如交通阻塞现象。

关于死锁的一些结论

- 参与死锁的进程最少是两个
  （两个以上进程才会出现死锁）
- 参与死锁的进程至少有两个已经占有资源
- 参与死锁的所有进程都在等待资源
- 参与死锁的进程是当前系统中所有进程的子集

3.7.1 死锁的概念（续）

2、死锁产生的原因

- 系统资源不足
- 进程推进的顺序不合适
- 资源分配不当

3、产生死锁的必要条件

- 互斥条件(mutual exclusion)

  资源互斥访问；

- 不剥夺条件(no preemption)

  不能强行剥夺进程拥有的资源；

- 请求和保持条件(request and hold)

  进程等待新资源时继续占有已分配的资源；

- 循环等待条件(circular wait)

  存在一种进程的循环链，链中的每一个进程已获得的资源同时被链中的下一个进程所请求；

4、死锁的处理方法

- **预防**：破坏产生死锁的四个必要条件之一。
- **避免**：在资源的动态分配过程中，用某种方法去防止系统进入死锁状态，从而避免死锁的发生。
- **检测**：通过检测机构及时检测出死锁的发生，并精确确定与死锁有关的进程和资源。
- **解除**：与检测死锁相配套，用于将进程从死锁状态解脱出来。

3.7.2 死锁的预防

死锁的预防一般是从破坏导致发生死锁的必要条件着手。

- 破坏互斥条件：把独享资源变为共享资源；
- 破坏不剥夺条件：主动释放资源策略和强行剥夺策略；
- 破坏请求和保持条件：采用静态分配策略；
- 破坏循环等待条件：采用资源的有序申请策略；

3.7.3 死锁的避免

要从根本上预防死锁是不现实的，因为这会导致资源使用和进程执行的低效。事实上，死锁的产生跟每个进程动态申请资源的过程息息相关。如果能掌握并发进程中与每个进程有关的资源动态申请情况，同样可以避免死锁的发生。通过定义两种状态：安全状态与不安全状态，分别对应于可以避免死锁和产生死锁的情况。

1、安全与不安全状态

安全状态是指在某一时刻，系统至少存在一个安全序列<P_1，P_2，P_3，…，P_n>，按照此序列来为每个进程分配所需资源，直到满足其最大需求，使每个进程都能顺利地完成，则称此时系统处于安全状态。反之，称之为不安全状态。

安全状态　安全序列：〈P2，P1，P3〉

进程号	总共需求	已分配	还需	剩余
P1	10	5	5	3
P2	4	2	2	
P3	9	2	7	

不安全状态　无安全序列

进程号	总共需求	已分配	还需	剩余
P1	10	5	5	2
P2	4	2	2	
P3	9	3	6	

【注意】：系统处于安全状态并不表示就一定不存在死锁。如果存在不安全序列且按不安全序列分配，就会发生死锁。处于不安全状态会产生死锁。

【思考题6】系统中有3个进程，每个进程最多需要3个资源，问系统中不存在死锁的最低资源数是多少？

2、银行家算法

- 银行家拥有一笔周转资金
- 客户要求分期贷款，如果客户能够得到各期贷款，就一定能够归还贷款，否则就一定不能归还贷款
- 银行家应谨慎的贷款，防止出现坏帐

具体过程如下：

- 顾客的借款操作依次顺序进行，直到全部操作完成；
- 银行家对顾客的借款操作进行判断，确定其安全性（能否支持顾客借款，直到全部归还）；
- 如果是安全的，则借款给顾客，否则暂不借款。

【例】判断银行家是否安全

总资金10万元		
用户	已借款（万）	还要借款（万）
P	4	4
Q	2	2
R	2	7

剩余资金：2万

用银行家算法避免死锁

- 操作系统（银行家）
- 操作系统管理的资源(周转资金)
- 进程（要求贷款的客户）

3、银行家算法中的数据结构
（N个进程使用M类资源）

- 可利用资源向量Available,为有m个元素的一维数组，每个元素代表一类资源
- 最大需求矩阵Max[n*m]
- 分配矩阵Allocation[n*m]
- 需求矩阵 Need[n*m]
- 进程的请求向量 Request,为有m个元素的一维数组，每个元素代表一类资源

当进程i申请资源时，执行下列步骤：

①若Request[i]≤Need[i],转②；
　　否则错误返回

②若Request[i]≤Available,转③；
否则等待

③假设系统分配了资源，则有：

Available=Available-Request[i];

Allocation[i]=Allocation[i]+Request[i];

Need[i]=Need[i]-Request[i]

④执行安全性算法检查。若系统新状态是安全的，则分配完成,若系统新状态是不安全的，则恢复原状态，进程等待。

安全性算法描述如下

（1）设置两个向量

- Work 它表示系统可提供给进程继续运行的各类资源数目，初始时，Work=Available。
- Finish 它表示系统是否有足够的资源分配给进程并完成运行。初始时，

Finish（i）=False；

当可以分配给进程i时，

Finish（i）=True。

安全性算法（续）

（2）从剩余进程集合中找到一个能满足下列条件的进程。

Finish(i)=False；

Need[i]≤Work；

安全性算法（续）

（3）当进程i获得资源后，可以顺利执行完毕，并释放出分配给它的资源，所以需执行：

```
Work= Work+Allocation[i];
Finish(i)=True;
Goto （2）；
```

安全性算法（续）

（4）如果所有进程的Finish(i)=true，则表示系统处于安全状态；反之，系统处于不安全状态。

【例】设系统有五个进程和三类资源，每类资源分别有10、5、7。在T0时刻资源分配情况如下图

进程 \ 资源情况	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	7	5	3	0	1	0	7	4	3	3	3	2
P_1	3	2	2	2	0	0	1	2	2			
P_2	9	0	2	3	0	2	6	0	0			
P_3	2	2	2	2	1	1	0	1	1			
P_4	4	3	3	0	0	2	4	3	1			

T_0 时刻的资源分配表

T0时刻可以找到一个安全序列{p1,p3,p4,p2,p0}， 系统是安全的。

进程 \ 资源情况	Work			Need			Allocation			work+Allocation			Finish
	A	B	C	A	B	C	A	B	C	A	B	C	
P_1	3	3	2	1	2	2	2	0	0	5	3	2	true
P_3	5	3	2	0	1	1	2	1	1	7	4	3	true
P_4	7	4	3	4	3	1	0	0	2	7	4	5	true
P_2	7	4	5	6	0	0	3	0	2	10	4	7	true
P_0	10	4	7	7	4	3	0	1	0	10	5	7	true

T_0 时刻的安全序列

在T0时刻，如果P1发出请求Request(1,0,2)，系统能否响应？为什么？

进程 \ 资源情况	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	7	5	3	0	1	0	7	4	3	3	3	2
										(2	3	0)
P_1	3	2	2	2	0	0	1	2	2			
				(3	0	2)	(0	2	0)			
P_2	9	0	2	3	0	2	6	0	0			
P_3	2	2	2	2	1	1	0	1	1			
P_4	4	3	3	0	0	2	4	3	1			

T_0 时刻的资源分配表

可以找到一个安全序列{p1,p3,p4,p0,p2}， 系统是安全的，可以满足P1的资源请求。

	Work			Need			Allocation			Work+Alocation			Finish
	A	B	C	A	B	C	A	B	C	A	B	C	
P_1	2	3	0	0	2	0	3	0	2	5	3	2	true
P_3	5	3	2	0	1	1	2	1	1	7	4	3	true
P_4	7	4	3	4	3	1	0	0	2	7	4	5	true
P_0	7	4	5	7	4	3	0	1	0	7	5	5	true
P_2	7	5	5	6	0	0	3	0	2	10	5	7	true

P_1 申请资源时的安全性检查

【思考题7】有三类资源A(17)、B(5)、C(20)。有5个进程P1-P5。T0时刻系统状态如下：

	最大需求	已分配
P1	5 5 9	2 1 2
P2	5 3 6	4 0 2
P3	4 0 11	4 0 5
P4	4 2 5	2 0 4
P5	4 2 4	3 1 4

问(1)、T0时刻是否为安全状态，给出安全系列。

(2)、T0时刻，P2: Request(0,3,4)，能否分配，为什么？

(3)、在(2)的基础上P4：Request(2,0,1)，能否分配，为什么？

解：(1) T0时刻的安全系列

先求出Need和Work

	最大需求	已分配	Need
P1	5 5 9	2 1 2	3 4 7
P2	5 3 6	4 0 2	1 3 4
P3	4 0 11	4 0 5	0 0 6
P4	4 2 5	2 0 4	2 2 1
P5	4 2 4	3 1 4	1 1 0

A(17)、B(5)、C(20)　　Work=(2, 3, 3)

Work=(2,3,3)

	Work	Allocation	Need	W+A	Finish
P5	2 3 3	3 1 4	1 1 0	5 4 7	T
P4	5 4 7	2 0 4	2 2 1	7 4 11	T
P3	7 4 11	4 0 5	0 0 6	11 4 16	T
P2	11 4 16	4 0 2	1 3 4	15 4 18	T
P1	15 4 18	2 1 2	3 4 7	17 5 20	T

(2) P2: Request(0,3,4)

因Available =(2, 3, 3)< Request(0,3,4),
所以不能分配

(3) P4：Request(2,0,1) Available =(2,3,3)

先进行预分配，得到下表

	Allocation	Need	Available
P1	2 1 2	3 4 7	0 3 2
P2	4 0 2	1 3 4	
P3	4 0 5	0 0 6	
P4	4 0 5	0 2 0	
P5	3 1 4	1 1 0	

有安全序列{P4 ,P5, P3, P2, P1}, 可以分配

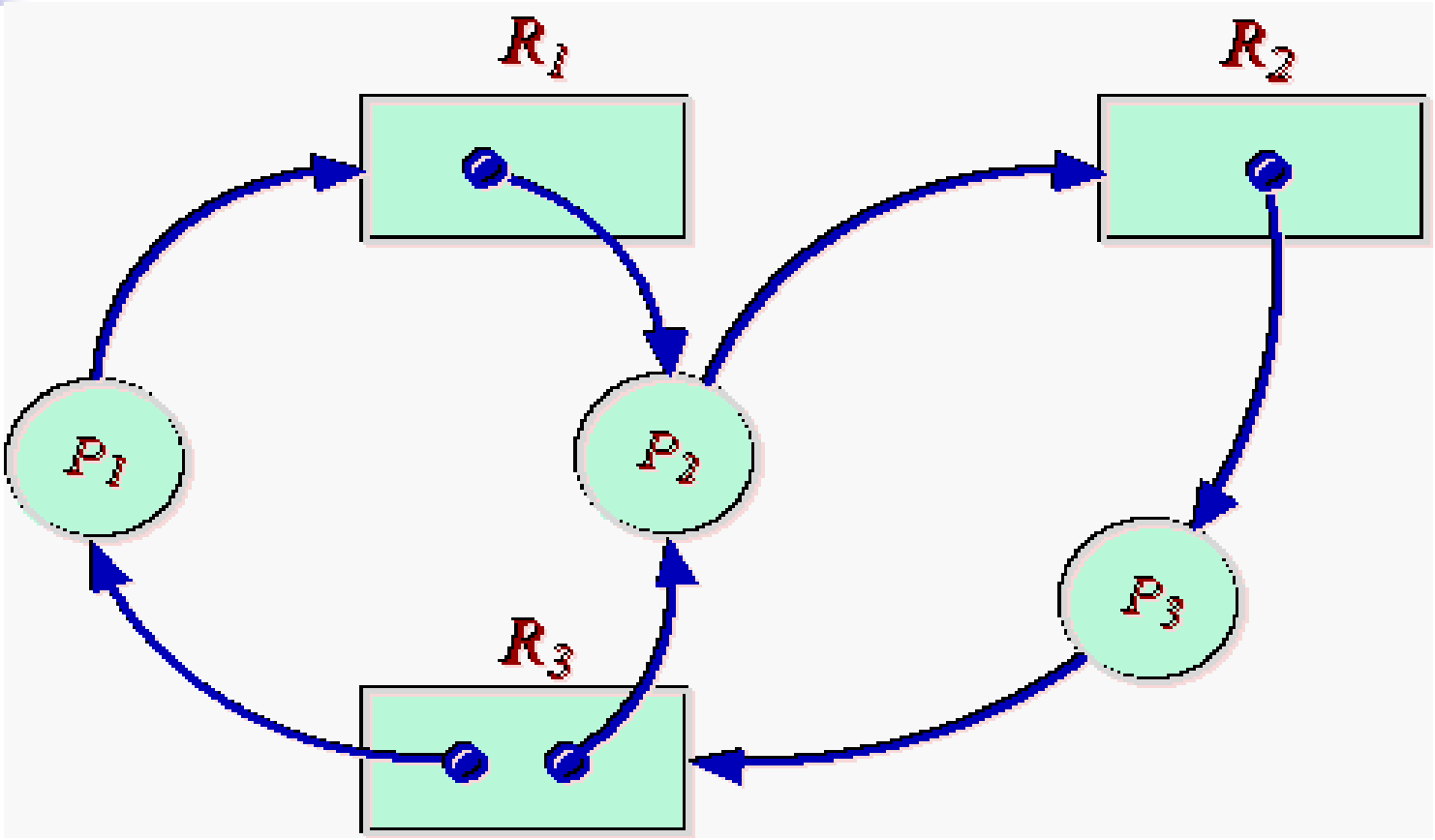
	Work	Allocation	Need	W+A	Finish
P4	0 3 2	4 0 5	0 2 0	4 3 7	T
P5	4 3 7	3 1 4	1 1 0	7 4 11	T
P3	7 4 11	4 0 5	0 0 6	11 4 16	T
P2	11 4 16	4 0 2	1 3 4	15 4 18	T
P1	15 4 18	2 1 2	3 4 7	17 5 20	T

3.7.4 死锁的检测

允许死锁发生，操作系统不断监视系统进展情况，判断死锁是否发生。

通过化简资源分配图实现

资源分配图示例



死锁定理

如果资源分配图中没有环路，则系统中没有死锁，如果图中存在环路则系统中可能存在死锁。

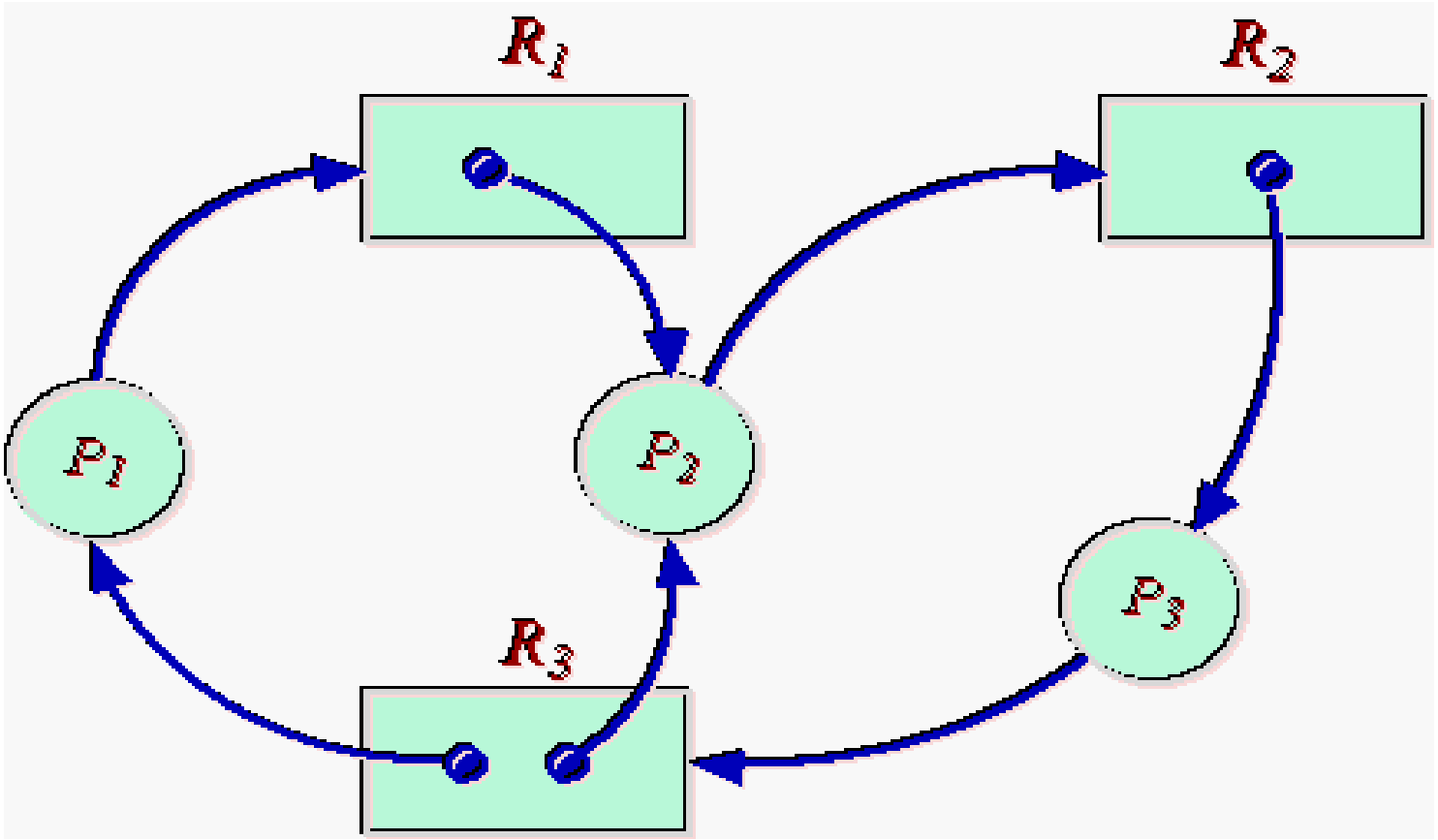
如果每个资源类中只包含一个资源实例，则环路是死锁存在的充分必要条件。

资源分配图化简

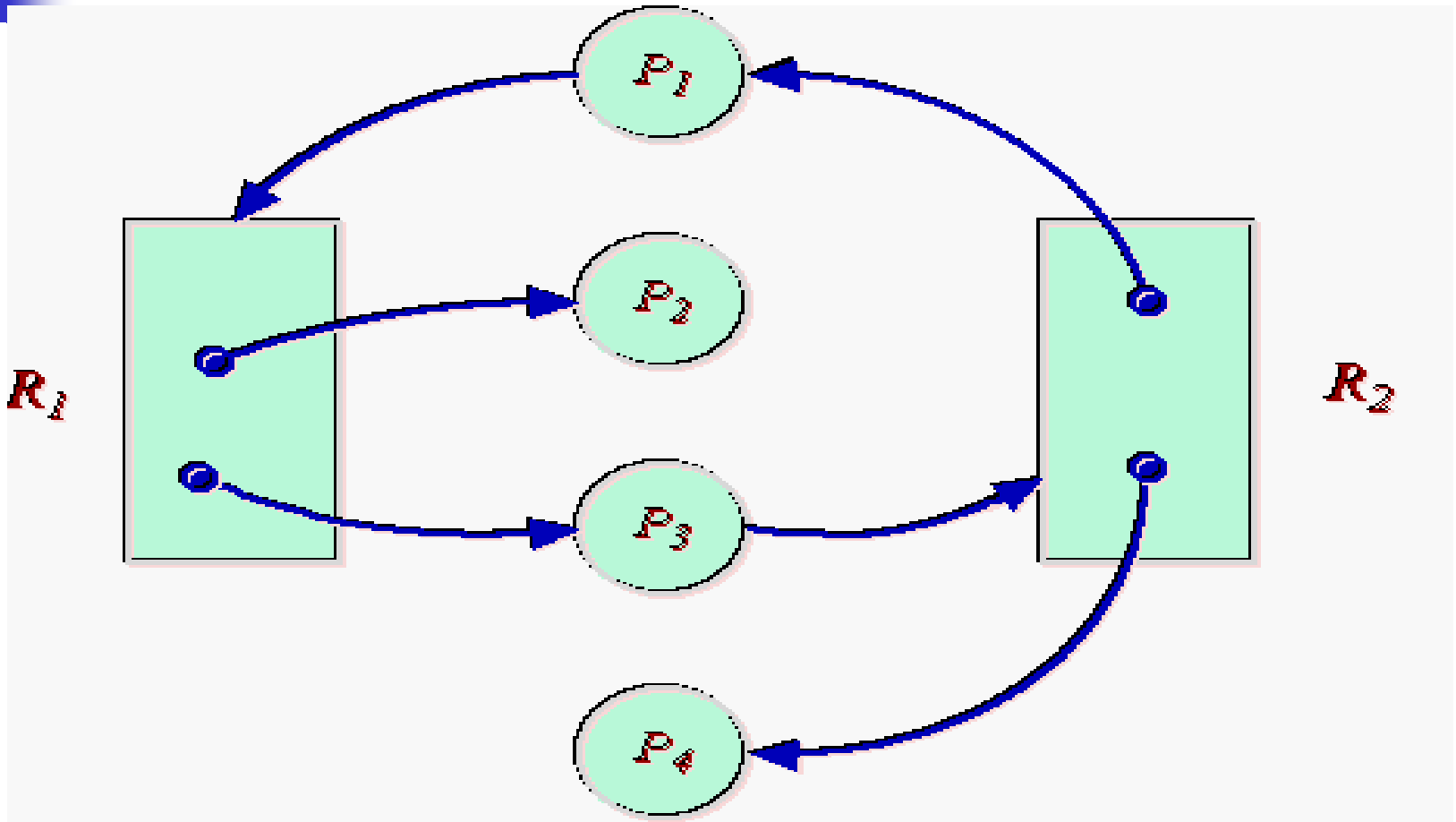
- 找一个非孤立点进程结点且只有分配边，去掉分配边，将其变为孤立结点
- 再把相应的资源分配给一个等待该资源的进程，即将某进程的申请边变为分配边
- 重复以上步骤，若所有进程成为孤立结点，称该图是可完全简化的，否则称该图是不可完全简化的

死锁状态的充分条件是：当且仅当资源分配图是不可完全简化的。

有环有死锁



有环无死锁



3.7.5 死锁的解除

1、进程撤销

可选择部分或全部撤销

2、进程回退

3、资源剥夺

避免进程“饥饿”

3.8 线程

3.8.1 线程的引入

引入进程是为了使多个程序并发执行，以改善资源利用率、提高系统吞吐量。

引入线程则是为了减少程序并发执行时的所付出的时空开销。尤其要解决进程在处理同数据集上的多请求时效率低下的问题。

【例如】

- LAN中的一个文件服务器，在一段时间内需要处理几个文件请求；
- 联网售票系统中，每个终端上的查询或购票请求；

用进程解决上述问题的方法

- 对所有的请求用一个服务进程来顺序处理
- 每一个请求都由一个相互独立的服务进程处理
- 用一个服务进程并发处理所有请求，只要存在请求进程，该服务进程就不进入等待状态

问题所在：进程是系统调度的基本单位

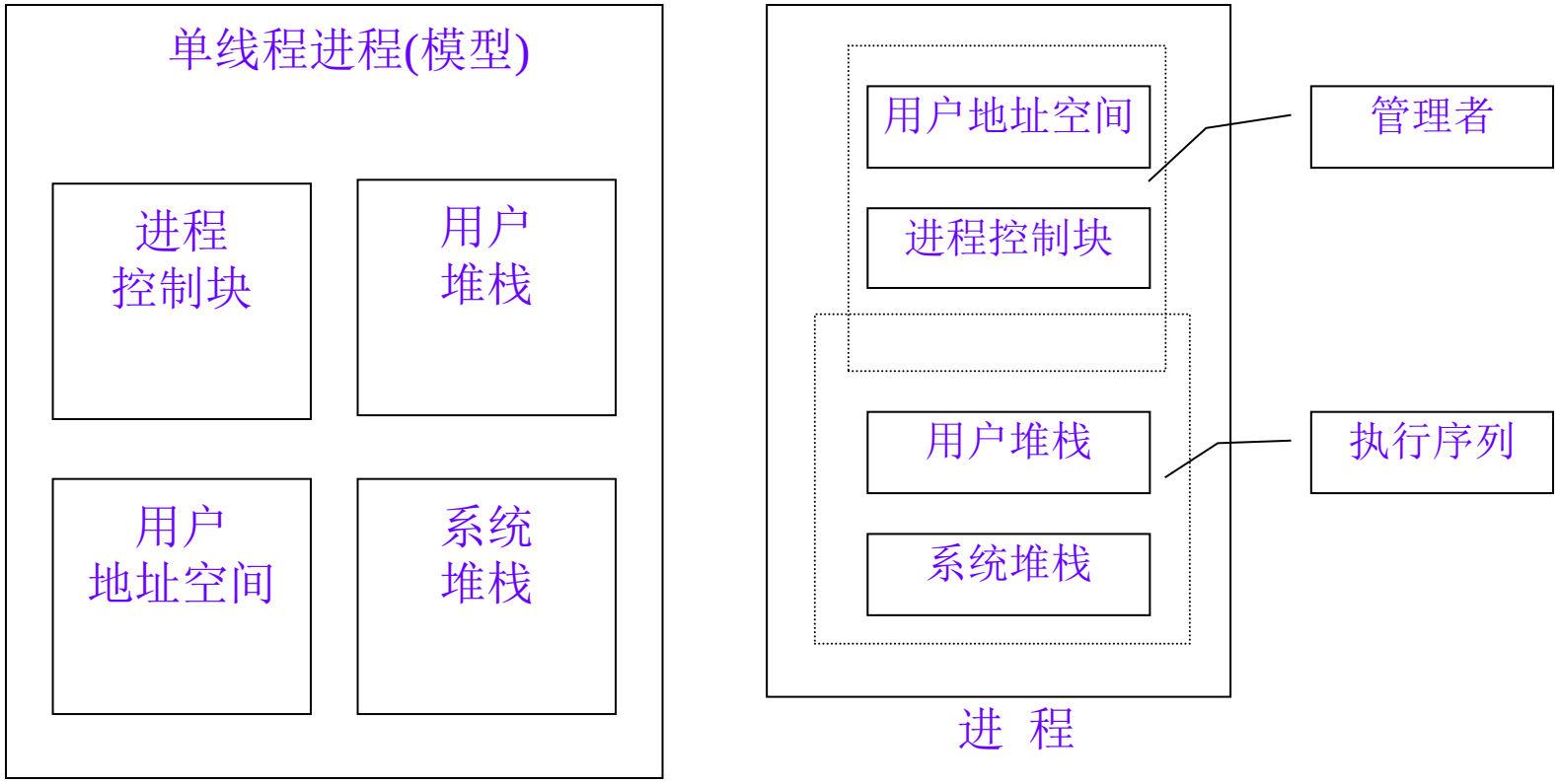
【解决方法】

- 把进程的两项功能—“独立分配资源”与“被调度分派执行”分离开来
- 进程作为系统资源分配和保护的独立单位，不需要频繁地切换
- 线程作为系统调度的基本单位，能轻装运行，会被频繁地调度和切换，在这种指导思想下，产生了线程和多线程进程(multiple threaded process)的概念。以前的进程称为单线程进程。

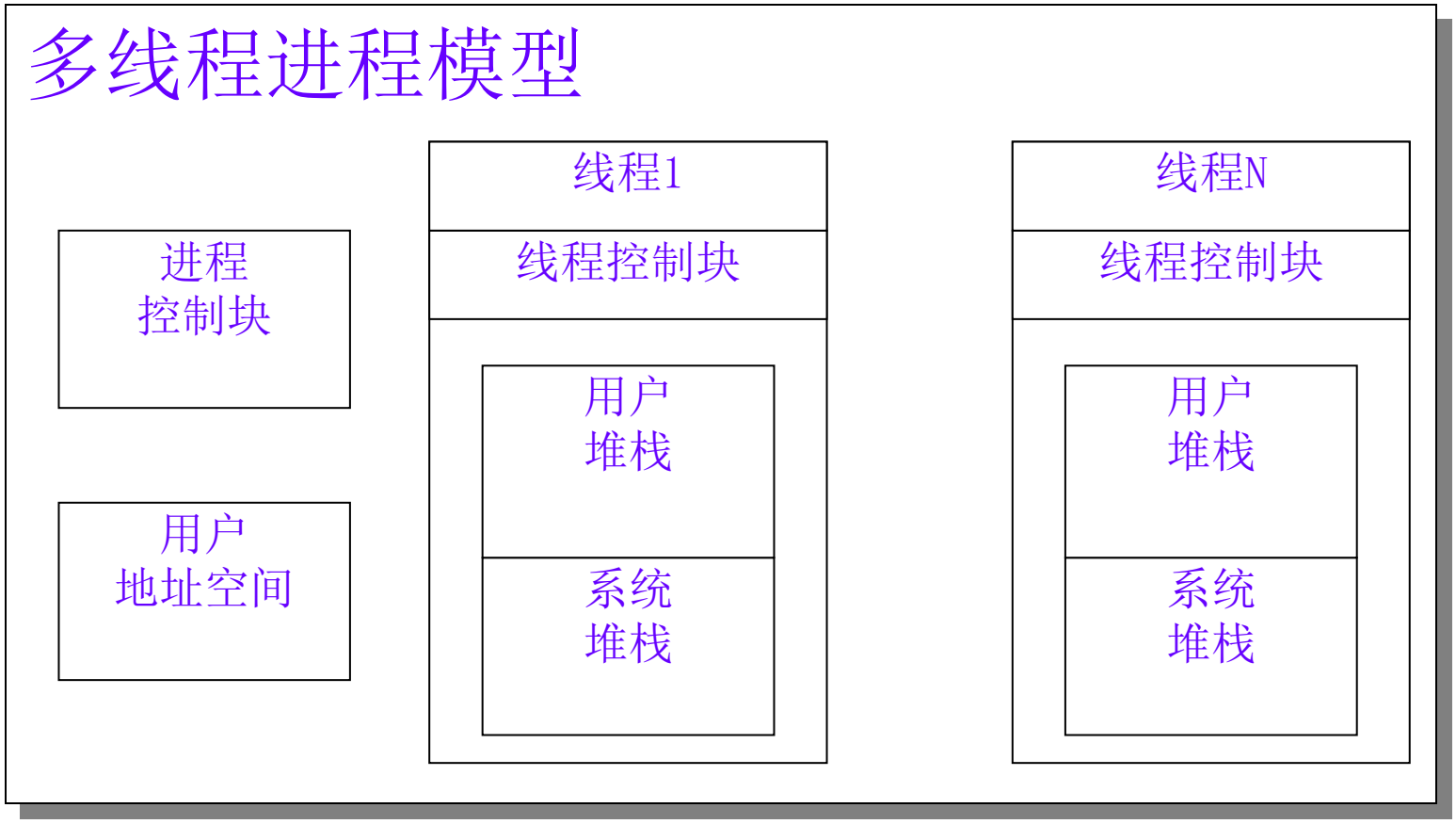
3.8.2 线程的概念

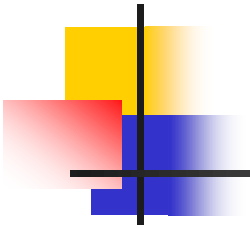
- 线程是进程中的一个实体（控制流），是被系统独立调度和分派的基本单位。
- 线程自己基本不拥有系统资源，只拥有少量必不可少的资源：程序计数器、一组寄存器、栈。
- 线程可与同属一个进程的其它线程共享进程所拥有的全部资源。
- 一个线程可以创建和撤消另一个线程；同一进程中的多个线程之间可以并发执行。

单线程进程的内存布局


单线程进程(模型)
进程
控制块
用户
堆栈
用户
地址空间
系统
堆栈
用户地址空间
进程控制块
用户堆栈
系统堆栈
管理者
执行序列
进 程

多线程进程的内存布局


多线程进程模型
进程
控制块
用户
地址空间
线程1
线程控制块
用户
堆栈
系统
堆栈
线程N
线程控制块
用户
堆栈
系统
堆栈



在多线程进程的系统中，进程可以看作是企业中的一条生产线（或者一个车间），而线程则是生产线（或车间）里干活的工人，这些工人可以做相同的工作，也可以做不同的工作。

多线程技术应用

- 前台和后台工作
- C/S应用模式
- 异步处理
- 加快执行速度
- 设计用户接口

3.8.3 线程与进程的比较

1.调度与切换：在多进程系统中，进程既是资源分配的基本单位，又是系统调度与切换的独立单位。而在多线程系统中，进程只是系统资源分配的基本单位，而线程是系统调度和切换的基本单位。

线程与进程的比较(续)

2.并发性：不仅进程之间可以并发执行，而且在一个进程中的多个线程之间亦可并发执行。

3.拥有资源：进程拥有的资源较多，而线程拥有的资源很少。

线程与进程的比较(续)

4.系统开销： 由于在创建或撤消进程时，系统都要为之分配或回收资源，如内存空间、I/O设备等。因此，操作系统所付出的开销将明显地大于在创建或撤消线程时的开销。

线程与进程的比较(续)

5.通信： 进程间通信可通过多种方式进行，而线程间通信主要通过共享内存方式，直接读写进程的数据段来进行通信。

3.8.4 线程的实现方式

1、内核级线程

依赖于内核的，即无论是用户进程中的线程，还是系统进程中的线程，它们的创建、撤消、切换都由内核实现。

内核级线程的优点和缺点

优点：

- 对多处理器，核心可以同时调度同一进程的多个线程
- 阻塞是在线程一级完成

缺点：

- 在同一进程内的线程切换调用内核，导致速度下降

2、用户级线程

- 由应用程序完成所有线程的管理

  通过线程库(用户空间)：一组管理线程的函数

  线程库提供一个线程运行管理系统（运行系统）
- 核心不知道线程的存在
- 线程切换不需要核心态特权
- 调度是应用特定的

用户级线程的优点和缺点

优点：

- 线程切换不调用核心
- 调度是应用程序特定的：可以选择最好的算法
- ULT可运行在任何操作系统上（只需要线程库），可以在一个不支持线程的OS上实现

缺点：

- 大多数系统调用是阻塞的，因此核心阻塞进程，故进程中所有线程将被阻塞
- 核心只将处理器分配给进程，同一进程中的两个线程不能同时运行于两个处理器上