第3章 处理器管理

一、系统的工作流程

1.程序及其特点

程序应具体两个基本特点：

- 顺序性
- 可再现性

2.系统的工作流程

- 顺序执行的工作方式及特征

  顺序执行指：处理器在开始执行一道程序后，只有在这道程序运行结束(程序指令运行完成，或程序运行过程出现错误而无法继续运行)，才能开始执行下一道程序。

  这种工作流程的外在表现就是单任务。特征：**封闭性可再现性**

- 并发执行的工作方式

  并发执行是指：在多道程序设计环境下，处理器在开始执行一道程序的第1条指令后，在这道程序完成之前，处理器可以开始执行下一道程序，同样地，更多其他的程序也可以开始运行。

  这种工作流程的外在表现就是多任务。

- 3. 并发执行的理解

➢ 宏观：多道程序“同时”在运行，表现为多任务

➢ 微观：多道程序又是轮流交替地在处理器上执行

- 4. 并发执行的特征

并发执行可以发挥硬件的并行能力，为任务协作提供可能，但并发执行具有复杂性：

➢ 随机性/不确定的：为操作系统管理提供可能

➢ 不可再现性：程序丢失了可再现性

➢ 相互制约

例3-1 已知两道程序PA和PB，它们对同一个变量count进行操作，PA程序每次运行时对变量count进行加1操作，而PB程序每次运行时对变量count进行减1操作。我们用C语言语法描述PA和PB程序如下：

PA(){	PB(){
int x;	int y;
① x=count;	④ y=count;
② x=x+1;	⑤ y=y-1;
③ count=x;	⑥ count=y;
}	}

可以假定：变量count为int类型，在count=100时PA()和PB()各运行一次，那么，在并发执行方式下，它们运行后count的值是多少？

◆①②③④⑤⑥执行： count=100

◆①④⑤⑥②③执行： count=101

◆④①②③⑤⑥执行 ： count=99

二、进程概念

1.进程定义

一道程序在一个数据集上的一次执行过程，称为一个进程(Process)

2.进程的主要特征

- 动态性
- 并发性
- 独立性
- 结构性
- 异步性

三、进程的动态性

1.进程的基本状态

运行(Running)

就绪(Ready)

阻塞(Blocked)

2.进程的状态转换

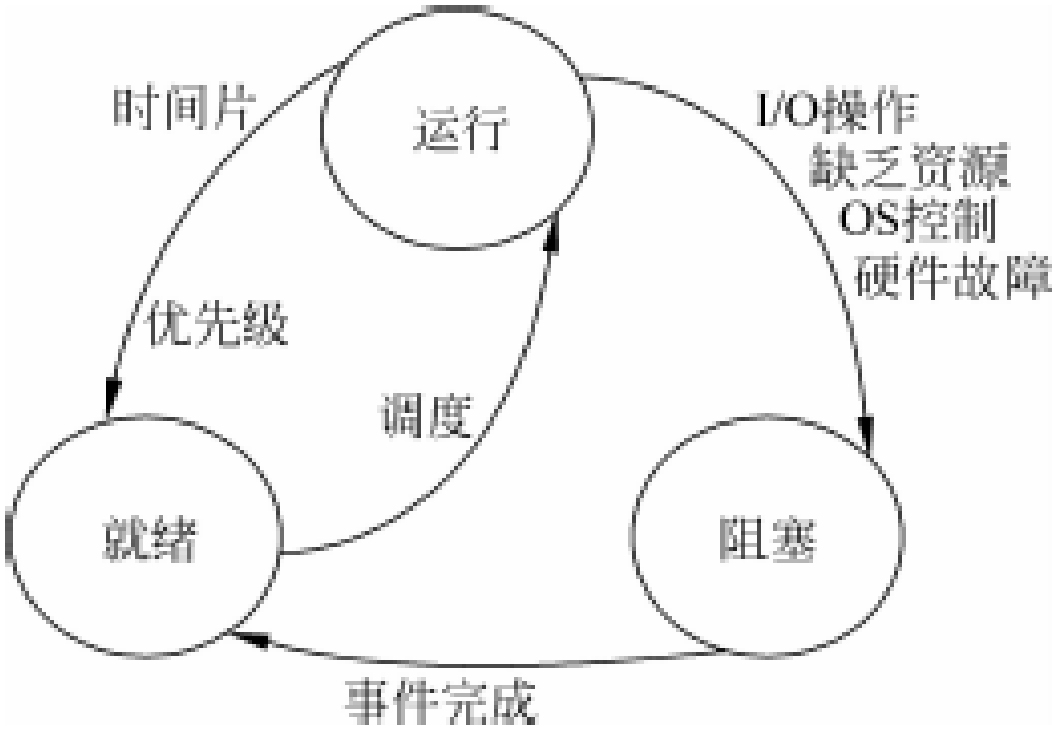

图 3-1　进程状态转换关系图

四、进程管理的主要功能

1.进程控制块PCB及组成

2. PCB队列

3.进程管理的主要功能

对处理器的管理转化为对进程的管理

- 控制
- 同步
- 通信
- 调度
- 死锁

五、进程控制

1.原子性(All or Nothing)

2. 原语(Primitive)

3.进程控制的含义

4.进程创建原语(Create)

- 创建进程的时机
- 创建原语的主要操作
  - 建立一个PCB
  - 生成pid
  - 初始化PCB各项内容(进程状态为就绪状态)
  - 加入合适的就绪队列
- 进程树

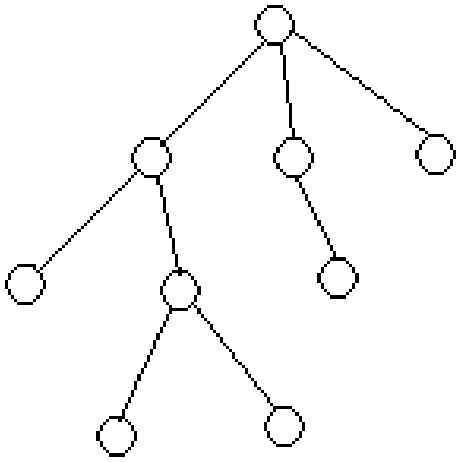

图3-2 进程树

4.进程撤销原语(Destroy)

- 进程撤销的时机
- 撤销原语的主要操作

5.进程阻塞原语(Blocked)

- 进程阻塞意义：减少CPU等待时间
- 阻塞原语的主要操作

6.进程唤醒原语(Wakeup)

- 唤醒原语的主要操作
  - 从等待队列中移出进程
  - 修改PCB的进程状态为就绪状态
  - 进程加入合适的就绪队列

在一个进程被唤醒时，它的阻塞状态直接改为运行状态，这种作法合理吗？

六、进程同步

1.并发进程的制约关系(例1)

P1程序	P2程序
…	…
打印第1行A1	打印第1行B1
打印第2行A2	打印第2行B2
…	…
打印第n行An	打印第m行Bm
…	…

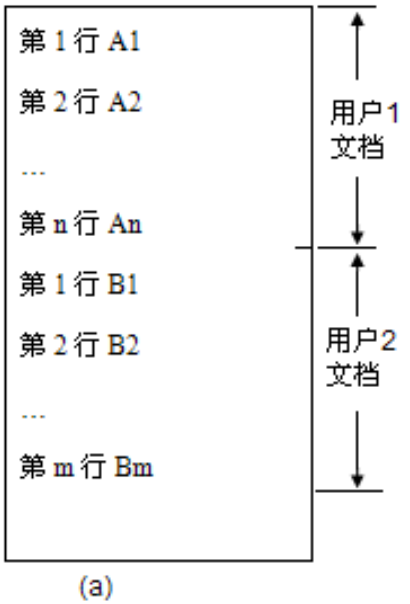

图3-3 例1进程顺序执行的结果

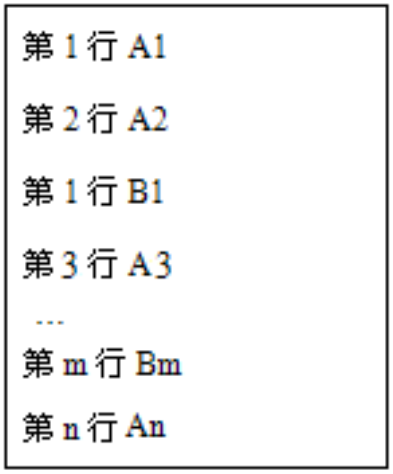

图3-4 例1进程并发执行的一种结果

间接制约关系 --资源共享引起

例2

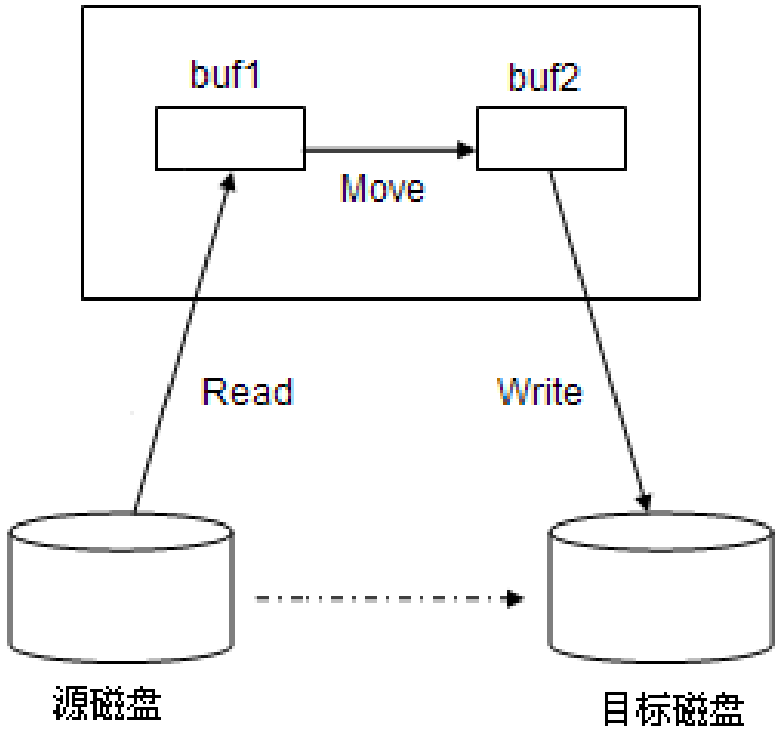

图3-5 三个进程的任务协作

直接制约关系--由任务协作引起的

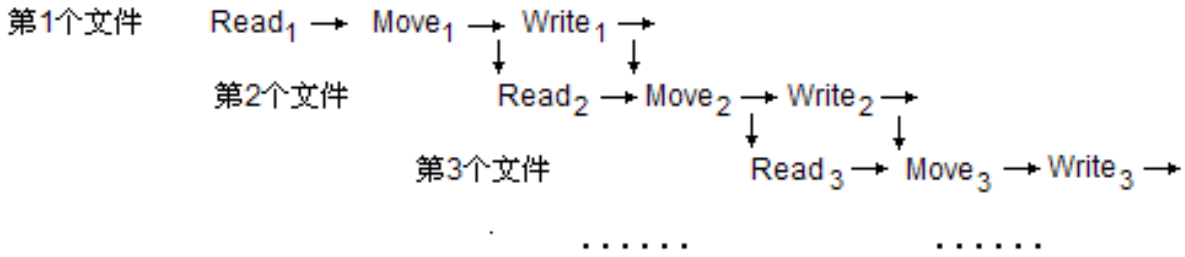

图3-10 Read、Move、Write的并行执行

2.间接制约与互斥关系

- 资源的使用步骤
- 临界资源与间接制约
- 临界区与互斥关系

临界区(Critical Section，或Critical Region)是指进程对应的程序中访问临界资源的一段程序代码，就是进程在资源的一次使用过程中，从申请开始至归还为止的一段程序代码。

图3-7 资源使用步骤

两个或两个以上的一组并发进程，称它们具有**互斥关系**，是指这组进程至少共享一类临界资源，当一个进程在临界资源对应的临界区内执行时，其他要求进入相关临界区执行的进程必须等待。

3.直接制约与同步关系

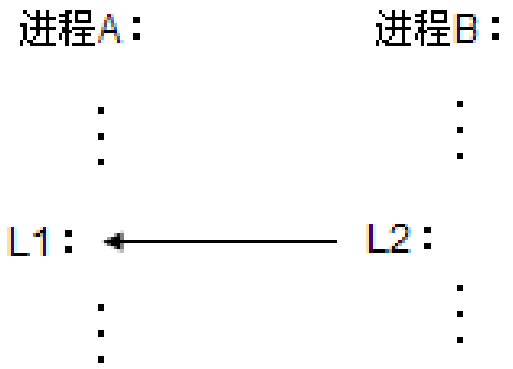

图3-8 单向依赖关系

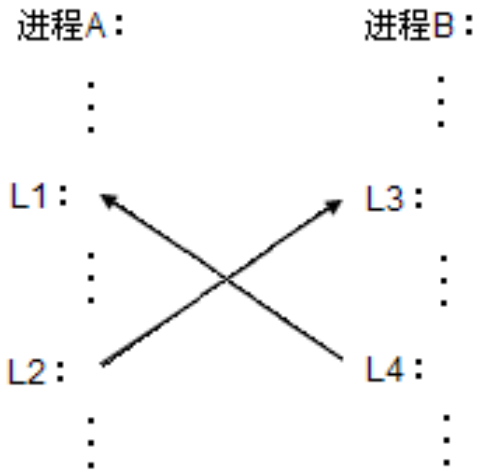

图3-9 相互依赖关系

在一组并发进程中，如果每个进程至少与同组中另一个进程存在单向或相互依赖关系，称这组进程具有**同步关系**，简称同步进程

4.进程同步机制

- 常用的进程同步机制

  加锁机制、标志位机制、信号量机制和管程机制

- 临界区管理准则
  - 空闲让进
  - 忙则等待
  - 有限等待
  - 让权等待

5.互斥关系与加锁机制

- 加锁机制原理：

  锁变量key、加锁操作lock(key)和解锁操作unlock(key)。

lock(key)
{
while(key==1);
key=1;
}
	unlock(key)
	{
	key=0;
	}

规定key=0时表示对应的锁是开的，允许进程进入对应的临界区执行；key=1表示应对的锁是关的，禁止进程进入对应的临界区

- 加锁机制应用

  假定p1、p2、…、pn是一组互斥关系的进程，对应的锁变量为key，那么，加锁机制的应用方法是：

  置锁变量初值key=0，对于进程pi，i=1,n，其加锁机制的控制方法描述如下

```
…
lock(key);
临界区;
unlock(key);
…
```

- 加锁机制分析
  - 普通的加锁机制不能实现互斥关系

```
tsl:
    mov ax,1
    xchg ax,key
    cmp     ax,0
    jne     tsl
```

  - 存在“忙等待”现象，浪费了处理器时间
  - 存在“饥饿”(Starvation)现象
  - 多个锁变量的加锁操作可能造成进程死锁

6.信号量机制与互斥关系

- 信号量机制原理：

  信号量(semaphore)、p()和v()操作.

struct semaphore {
int value;
PCB *bq;
}
	p(s)
	{
	s.value = s.value – 1 ;
	if (s.value＜0)
	blocked(s) ;
	}
		v(s)
		{
		s.value = s.value + 1 ;
		if (s.value≤0)
		wakeup(s);
		}

p()和v()操作定义为原语， p()和v()操作的作用？

信号量机制实现互斥关系

假定进程p1、p2、…、pn共享某一个临界资源，定义一个信号量s，初值为1，那么，应用信号量机制实现p1、p2、…、pn互斥关系的模型如下：

对于进程pi，i=1,n，其信号量机制的控制描述如下：

…

p(s);

临界区;

v(s);

…

7.信号量机制与同步关系

简单同步关系

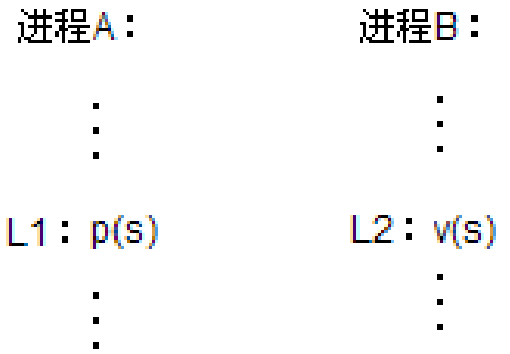

图3-10 简单同步关系

一般同步关系

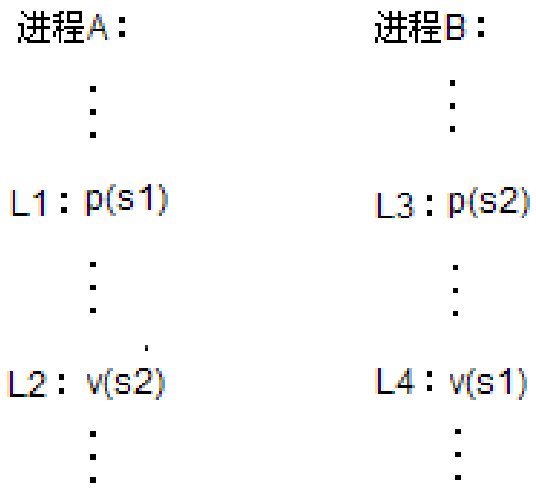

图3-11 一般同步关系

并发程序设计 对于给定的一组进程，应用同步机制实现它们的并发执行

例3-4 两个进程P1和P2共享一个缓冲区 buf，进程P1反复地计算，并把计算结果存入缓冲区buf，进程P2每次从缓冲区中取出计算结果并送向打印机。规定：P1把结果存入缓冲区buf后，P2才能打印，P1的一次计算的结果只能打印一次，只有在结果被打印后，P1新的计算结果才能存入缓冲区。试用信号量机制实现P1和P2的并发执行。

7.经典同步问题-生产者/消费者问题

- PC问题的描述
- PC问题的分类

  生产者进程数为n，消费者进程数为m，缓冲区容量为k

  - 简单PC问题：n=1,m=1,k=1
  - 一般PC问题：n=1,m=1,k>1
  - 复杂PC问题：n>1,m>1,k>1
  - 特殊PC问题：其他，主要有n+m=3或4，k=1或者2；条件消费或重复消费等

在并发程序设计中，如果有两个或多个连续的p操作，就必须认真分析，合理安排它们的执行顺序。

7.经典同步问题-读者/写者问题

- 问题描述

假设有一个写者进程Writer和若干个读者进程Reader，他们共享一组数据，写者进程Writer对数据进行写操作(如修改、删除、添加等)，读者进程Reader对数据进行读操作，规定：

(1)写操作与任一读操作之间，必须互斥执行；

(2)多个读操作可以同时进行。

如何用信号量机制实现他们的并发执行？

七、进程通信

1.进程通信的概念

- 什么是进程通信
- 进程通信类型：

  低级通信

  高级通信：没有特别说明，进程通信是指高级通信，即应用程序之间的数据交换
- 为什么需要进程通信

  任务协作

  进程的独立性
- 进程通信的可行性

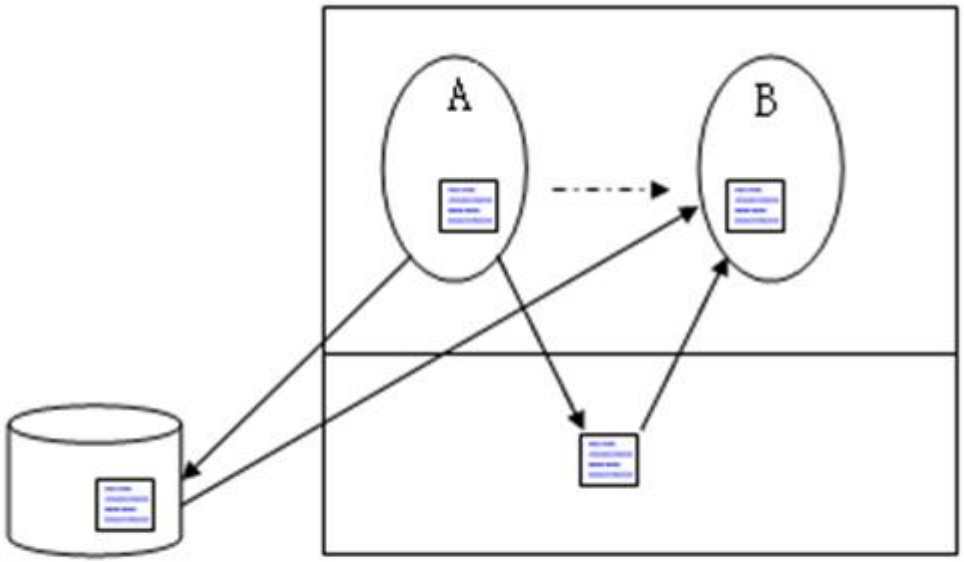

图3-13 进程通信方案

2.进程通信方式

- 共享存储区通信
- 消息缓冲通信
- 信箱通信
- 管道通信

3.消息缓冲通信的设计和实现

✓ 基本思想

✓ 设计

- ◆ 消息缓冲区结构
  - 发送进程标识(pid)
  - 正文大小(size)
  - 正文(data)
  - 向下指针(Next)
- ◆ PCB的通信参数结构
  - 消息缓冲区队列(mq)
  - 互斥信号量(mutex)
  - 同步信号量(msg)
- ◆ 发送操作和接收操作
  - send(dest，&mptr)
  - receive(&mptr)

✓ 实现-典型的简单同步问题

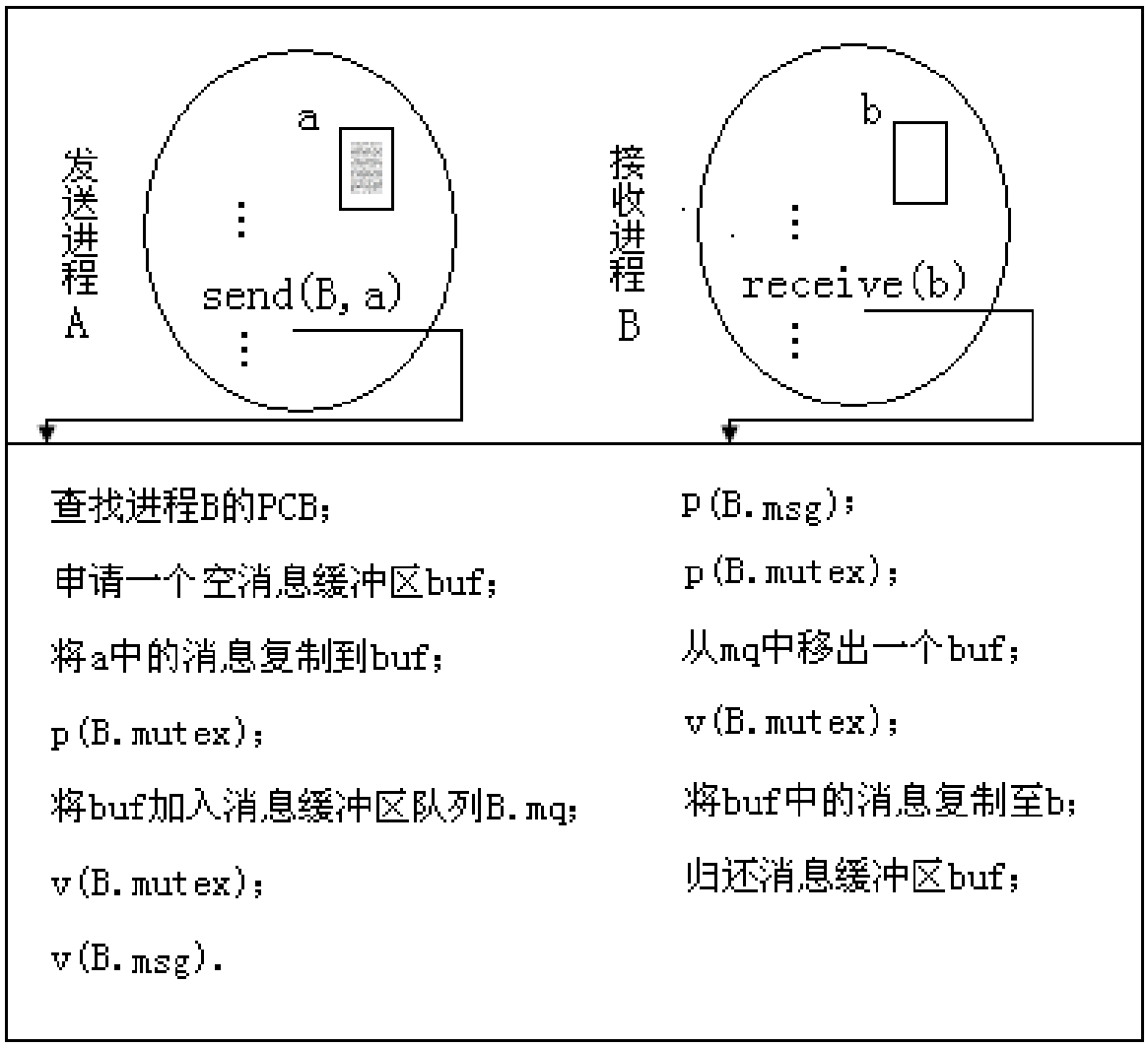

图3-14 消息缓冲通信的实现

✓分析

- 直接通信
- IPC

3.信箱通信的设计和实现

- ✓ 信箱结构
  - ◆ 信箱头基本结构
    - 信箱名(boxname)
    - 信箱标识符(bid)
    - 信箱大小(size)
    - 同步信号量(mailnum)
    - 同步信号量(freenum)
    - 读互信号量(rmutex)
    - 写互信号量(wmutex)
    - 读信件指针(out)
    - 存信件指针(in)
  - ◆ 信箱体结构
    - 由若干个信格组成，一个信格存放一个信件，要交换的数据组织成信件
  - ◆ 发送操作和接收操作
    - send(dest，&mptr)
    - receive(addr，&mptr)

✓ 实现-复杂PC同步问题

send(dest，&mptr)
{
p(dest.freenum)；
p(dest.wmutex)；
dest.buf[dest.in] ← mptr；
dest.in = (dest.in+1) % dest.size；
v(dest.wmutex)；
v(dest.mailnum)；
}
	receive(addr，&mptr)
	{
	p(addr.mailnum)；
	p(addr.rmutex)；
	mptr ← addr.buf[addr.out]；
	addr.out = (addr.out +1) % addr.size；
	v(addr.rmutex)；
	v(addr.freenum)；
	}

八、线程

1.线程的引入

- 什么是线程

  把进程细化成若干个可以独立运行的实体，每一个实体称为一个线程(Thread)
- 引入线程的目的--引入线程可以减小系统的基本工作单位粒度

  实现进程内部的并发执行，提高并行程度

  减少处理器切换带来的开销

  简化进程通信方式

2.线程与进程的关系

- 同一进程的线程之间共享的该进程地址空间
- 与进程一样，线程具有动态性和并发性
- 线程是处理器分配调度的基本单位
- 进程是其他资源(除处理器之外)分配的基本单位
- 进程之间在处理器切换时现场的保护/恢复的开销比较大
- 同一进程的线程之间在处理器切换时现场的保护/恢复的开销比较小

3.线程包(Threads Package)、线程类型

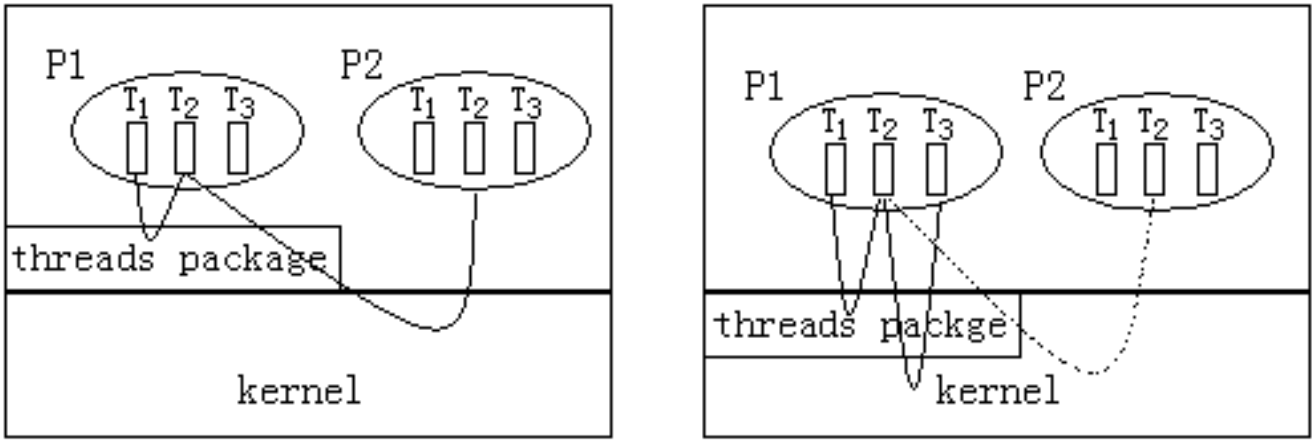
P1
P2
T1 T2 T3
T1 T2 T3
threads package
kernel
(a)用户级线程
P1
P2
T1 T2 T3
T1 T2 T3
threads packge
kernel
(b)系统级线程

图3-15 线程的类型

4.线程的常用细化方法

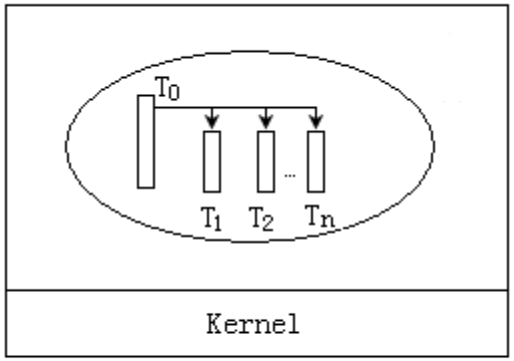

图3-16 分派/处理模型

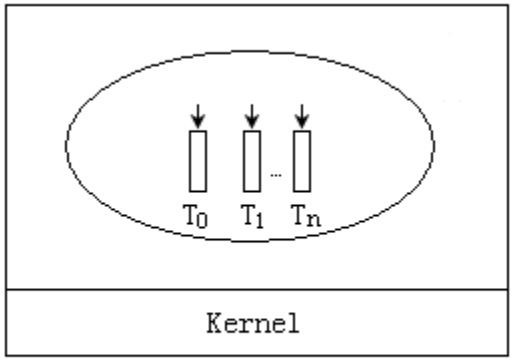

图3-17 队列模型

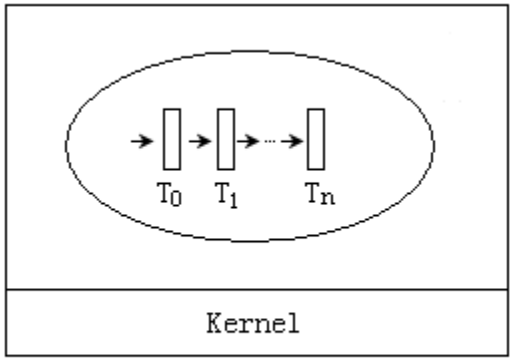

图3-18 管道模型