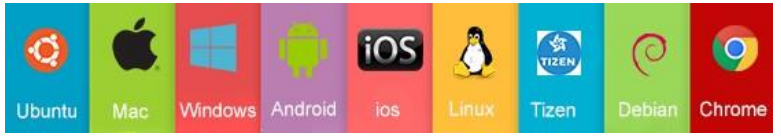
操作系统
Operating System

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



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

回顾——进程同步机制

- 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

回顾——进程同步机制

◆临界资源：进程采取互斥方式实现对资源的共享

- 著名的进程同步问题：生产者与消费者（producer-consumer）问题
- 临界区(critical section)：每个进程中访问临界资源的那段代码，存取操作区域

◆进程同步和互斥

- **进程同步（直接相互制约关系）**：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。
- **进程互斥（间接相互制约关系）**：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。

回顾——进程同步机制

◆ 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

回顾——进程同步机制

- **硬件同步机制**
  - 关中断
  - 利用test-and-set指令实现互斥
  - 利用swap指令实现进程互斥
- **信号量机制（PV操作）**
  - 整型信号量
  - 记录型信号量（结构型信号量）
  - AND型信号量
  - 信号量集

回顾——进程同步机制

✓ 空闲让进　✓ 忙则等待
✓ 有限等待　× 让权等待

◆ 整型信号量

- **P操作**：表示同步进程发出的检测信号量操作，检测是否能够使用临界资源
- **V操作**：表示访问完临界资源的进程通知等待进程已经完成了临界资源的访问，将临界资源释放
- **P操作**

```
wait(S){
    while S≤0；  // do no-op
    S: = S - 1;}
```

- **V操作**

```
signal(S){
     S: = S + 1; }
```

其中，S为一个需要初始化值的正整型量。对S的访问只能通过P、V操作。

- S>0，表示临界资源可以访问，P(S)中的检测语句通过，调用P(S)的进程可以访问临界资源。
- S≤0，表示有进程在访问临界资源，此时临界资源处于忙，调用P(S)的进程只能等待，直到S的值大于0，才可以访问临界资源。

回顾——进程同步机制

✓ 空闲让进　✓ 忙则等待
✓ 有限等待　✓ 让权等待

◆ 记录型信号量（结构型信号量）

➢记录型信号量在整型信号量的基础上进行了改进，让不能进入临界区的进程**“让权等待”**，即进程状态由运行转换为阻塞状态，进程进入阻塞队列中等待。

➢P操作

```
wait(S){
    S.value = S.value - 1
    if S.value<0 then block(S,L)
}
```

若信号量S的S.value的值为正数，该正数表示可对信号量可进行的P操作的次数，即实际可以使用的临界资源数

➢V操作

```
signal(S){
    S.value = S.value + 1
    if S.value≤0 then wakeup(S,L)
}
```

若信号量S的S.value的值为负值，表示有多个进程申请临界资源，而又不能得到，在阻塞队列等待。S.value的绝对值表示在阻塞队列等待临界资源的进程个数

回顾——进程同步机制

◆ AND型信号量

- 指同时需要**多种资源**且每种占用一个时的信号量操作；
- AND型信号量集的基本思想：在一个原语中申请整段代码需要的**多个临界资源**，要么**全部分配给它**，要么一**个都不分配**；
- AND型信号量集**V原语**为**Ssignal：**

```
Swait(S1, S2, …, Sn){
    While(TRUE){
        if (S1 >=1 and … and Sn>=1 ){
            for( i=1;i<=n; i++)    Si--;
            break；  }else{
            waiting；  }}}
```

```
Ssignal(S1, S2, …, Sn){
    while(TRUE){
        for (i=1; i<=n; i++)
        {
            Si++ ;    }}}
```

回顾——进程同步机制

◆ 信号量集

- 当进程需要**申请的临界资源种类较多**，**每类临界资源个数较多**时，如果用记录型信号量，进程每次只能一次申请或释放一个临界资源，非常麻烦。**因此，引入信号量集。**
- 在每次分配之前，测试**资源数量是否大于可分配的下限值**

  Swait(S1, t1, d1; ...; Sn, tn, dn);　（t：分配下限；d：需求值）

  Ssignal(S1, d1; ...; Sn, dn);
- 一般“信号量集”可以用于各种情况的资源分配和释放。下面是几种特殊的情况：

  1) **Swait(S, d, d)**表示每次申请d个资源，当资源数量少于d个时，便不予分配。

  2) **Swait(S, 1, 1)**表示互斥信号量。

  3) **Swait(S, 1, 0)**可作为一个可控开关(当S≥1时，允许多个进程进入临界区；当S=0时禁止任何进程进入临界区)。
- 由于一般信号量在使用时的灵活性，因此通常并不成对使用Swait和Ssignal。为了避免死锁可一起申请所有需要的资源，但**不一起释放**

回顾——进程同步机制

每个访问临界资源的进程都必备同步操作P、V，使得大量的同步操作分散在各个进程中，给系统管理带来麻烦，还可能导致系统死锁。因此，引入新的进程同步工具——管程（Monitors）

- **管程——一个操作系统的资源管理模块**
  - 代表共享资源的数据结构
  - 对该共享数据结构实施的一组过程所组成的资源管理程序
- **管程的定义（Hansan）**
  - 一个数据结构和能为并发进程所执行（在该数据结构上）的一组操作，这组操作能同步进程和改变管程中的数据
- **管程的组成**
  - 管程的名称
  - 局部于管程的共享数据结构说明
  - 对该数据结构的进行操作的过程
  - 对局部于管程的共享数据设置初始值的语句

回顾——经典进程的同步问题

回顾——生产者和消费者问题

```
mutex,full,empty:semaphore
mutex :=1;  full:=0;   empty:=n;
```

- **生产者P:**

```
While(True){
    P(        );
    P(        );
    Buffer(in)=nextp;
    in:=(in+1) mod n;
    V(        );
    V(        );
}
```

- **消费者C:**

```
While(Ture){
    P(        )
    P(        );
    netxc=buffer(out);
    out:=(out+1) mod n;
    V(        );
    V(        );
}
```

回顾——生产者和消费者问题

```
mutex,full,empty:semaphore
mutex :=1;  full:=0;   empty:=n;
```

- **生产者P：**

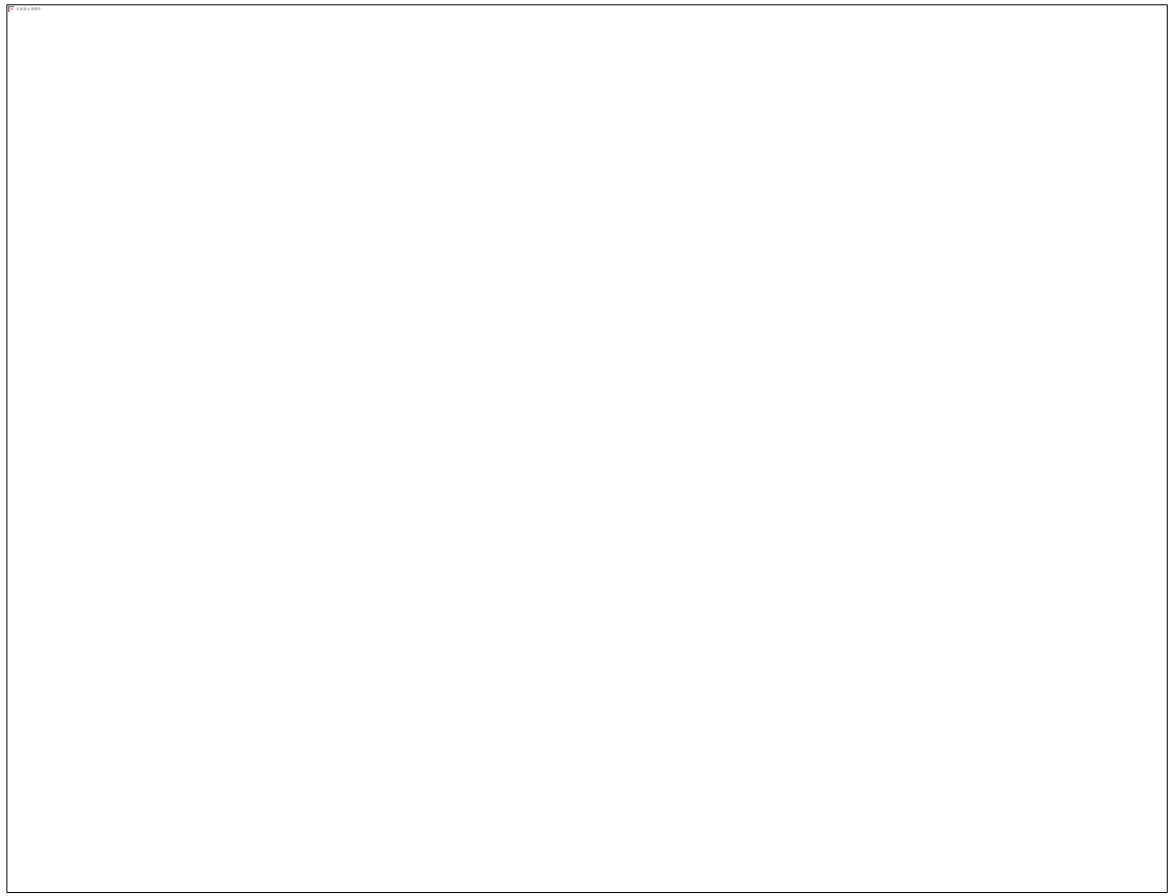
```
While(True){
    P(empty);
    P(mutex);
    Buffer(in)=nextp;
    in:=(in+1) mod n;
    V(mutex);
    V(full);
}
```

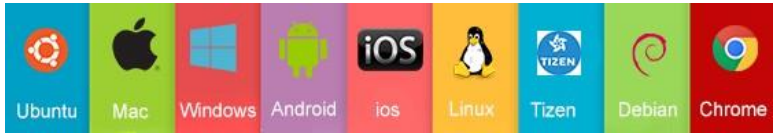
- **消费者C：**

```
While(Ture){
    P(full)
    P(mutex);
    netxc=buffer(out);
    out:=(out+1) mod n;
    V(mutex);
    V(empty);
}
```

回顾——哲学家就餐问题

- 方案2：仅当一个哲学家左右两边的筷子都可用时才允许他拿筷子（AND信号量机制）



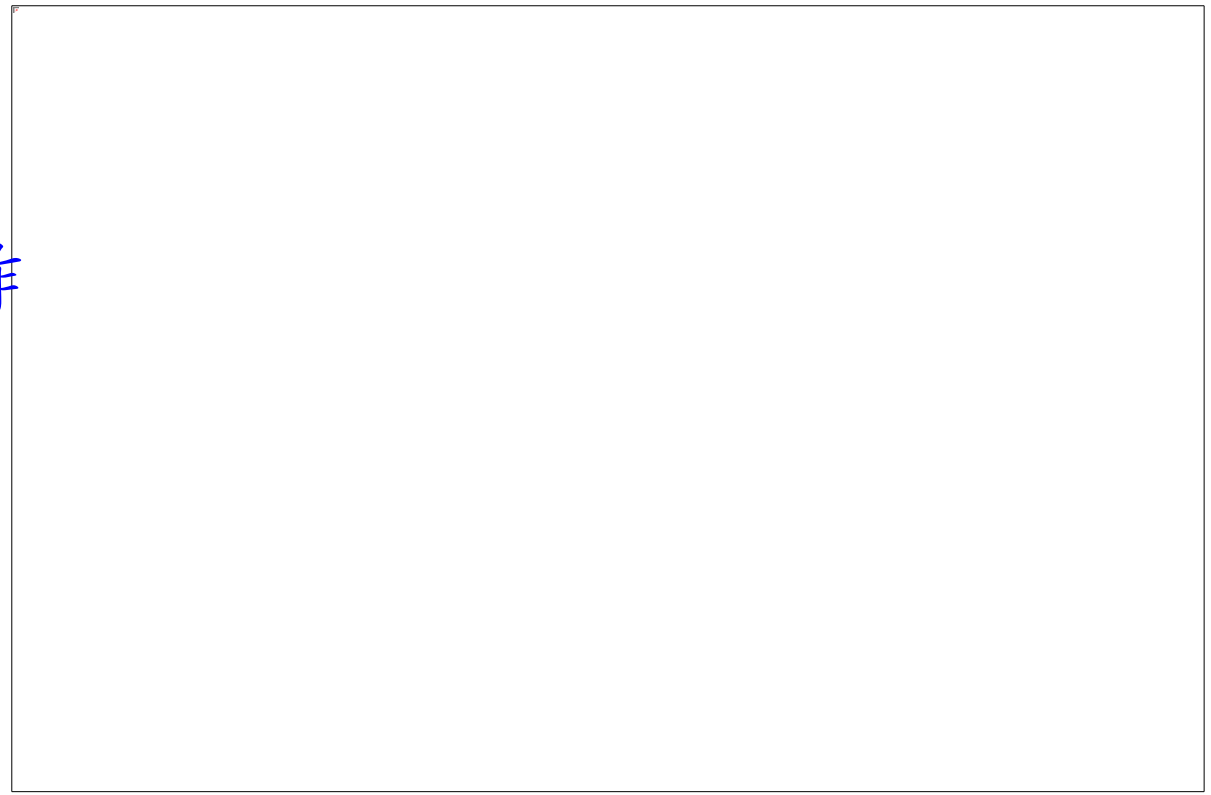


进程的描述和控制

- 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

2.5.3 读者和写者问题

- 有两组并发进程:
  - 读者（reader进程）和写者（writer进程），共享一组数据区
- 要求:
  - 允许多个读者同时执行读操作
  - 不允许读者、写者同时操作
  - 不允许多个写者同时操作



2.5.3 读者和写者问题

- 没有读者的时候，才可以写，读者比较优先

- 两个进程：
  - Reader、Writer
  - 读者与写者间的互斥信号量：Wmutex＝1
  - 多个读者间的互斥信号量：Rmutex＝1

- Readcount：正在读取的进程数目
  - Readcount＝0时允许写

2.5.3 读者和写者问题

读者Reader

```
While(True){
    P(rmutex)
    readcount++;
    if(readcount==1)
          P(wmutex);
    V(rmutex)
    执行读操作
    P(rmutex)
    readcount--;
    if(readcount==0)
        V(wmutex);
    V(rmutex)}
```

写者Writers

```
While(True){



 P(wmutex);
  执行写操作
 V(wmutex);




}
```

2.5.3 读者和写者问题

- 增加限制条件，即同时读取的读者数不能超过**RN**
- 信号量集:
  - Swait(S, t, d);　Ssignal(S,d)
  - S为信号量，t为下限值, d为需求量，
- **L, mx:=RN, 1**
  - 通过执行wait(L,1,1)操作来控制读者的数目,每当有一个读者进入时,就要先执行wait(L,1,1)操作,使L的值减1。
  - 当有RN个读者进入读后L便减为0，第RN+1个读者要进入读时，必然会因wait(L,1,1)操作失败而阻塞。
  - mx表示写资源。

2.5.3 读者和写者问题

读者Reader

```
int RN;
semaphore L=RN, mx=1;
void reader(){
    do{
        Swait(L,1,1; mx,1,0);
        ...
        进行读操作;
        Ssignal(L,1);
    }while(TRUE);
}
```

写者Writers

```
void writer(){
    do {
        Swait(mx,1,1;L,RN,0);
        ...
        进行写操作;
        Ssignal(mx,1);
    }while(TRUE);
}
```

思考题（2013年考研题）

某博物馆最多可容纳500人同时参观，有一个出入口，该出入口一次仅允许一个人通过。参观者的活动描述如下：

```
参观者进程i：
{       ……
        进门
        ……
        参观
        ……
        出门
        ……
}
```

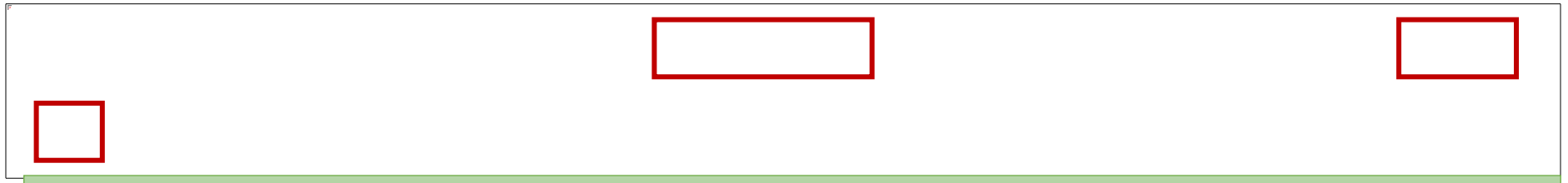
思考题（2013年考研题）

某博物馆最多可容纳500人同时参观，有一个出入口，该出入口一次仅允许一个人通过。参观者的活动描述如下：

```
参观者进程i：
{       ……
        进门
        ……
        参观
        ……
        出门
        ……
}
```

```
mutex, empty: semaphore;
mutex =1; empty=500;
参观者进程i：
{       P(empty);
        P(mutex);
        进门;
        V(mutex);
        参观;
        P(mutex);
        出门;
        V(mutex);
        V(empty);
}
```

拓展



semaphore in = 2; // 有两个入口　semaphore out = 1; semaphore empty = 500;

```
游客进程:
 while(1){
        P(      );
        P(      );
        进门
        V(      );
        参观
        P(      );
        出门
        V(      );
        V(      );
}
```

semaphore in = 2; // 有两个入口　semaphore out = 1;
semaphore empty = 500;　　semaphore full = 0;

```
游客进入进程:
 while(1){
        P(      );
        P(      );
        进门
        V(      );
        参观
        V(      );
}
```

```
游客出门进程:
 while(1){
        P(      );
        P(      );
        出门
        V(      );
        V(      );
}
```

拓展

1. 某工厂有两个生产车间和一个装配车间，两个生产车间分别生产 A、B 两种零件，装配车间的任务是把 A、B 两种零件组装成产品。两个生产车间每生产一个零件后都要分别把它们送到装配车间的货架 F1、F2 上，F1 存放零件 A，F2 存放零件 B，F1 和 F2 的容量均为可以存放 10 个零件。装配工人每次从货架上取一个 A 零件和一个 B 零件然后组装成产品。请用 PV 操作进行正确管理。(15 分)

```
        P(empty);
        P(in);
        进门
        V(in);
        参观
        P(out);
        出门
        V(out);
        V(empty);
}
```

游客进入进程：

```
while(1){
        P(empty);
        P(in);
        进门
        V(in);
        参观
        V(full);
}
```

游客出门进程：

```
while(1){
        P(full);
        P(out);
        出门
        V(out);
        V(empty);
}
```

拓展

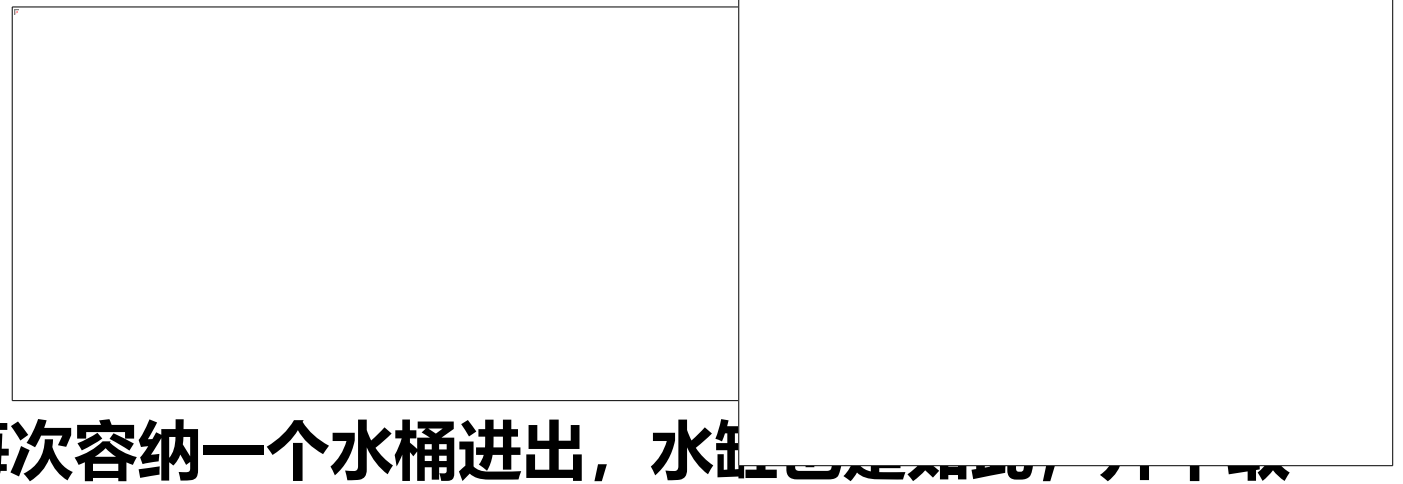
```
semaphore F1_full = 0;
semaphore F1_empty = 10;
semaphore F1_mutex = 1;
A:
while(1)
{
        生产零件 A;
        P(F1_empty);
        P(F1_mutex);
        放入货架
        V(F1_mutex);
        V(F1_full);
 }
```

```
semaphore F2_full = 0;
semaphore F2_empty = 10;
semaphore F2_mutex = 1;
B:
while(1)
{
        生产零件 B;
        P(F2_empty);
        P(F2_mutex);
        放入货架
        V(F2_mutex);
        V(F2_full);
 }
```

```
Assemble:
 while(1) {
        P(F1_full);
        P(F1_mutex);
        拿取 A;
        V(F1_mutex);
        V(F1_empty);
        P(F2_full);
        P(F2_mutex);
        拿取 B;
        V(F2_mutex);
        V(F2_empty);
        装配; }
```

2.5 进程同步应用示例-和尚喝水问题

- **题目：**
  - 小和尚从井中取水入缸供老和尚饮用。水缸可容纳10桶水。水井很窄，每次只能容一个桶取水。水桶总数为3个。每次入、取缸水仅为1桶，且不可同时进行。请给出取水、入水的算法描述。
  - 取水：从井中取水，从缸中取水
  - 入水：水倒入缸中



- **分析：**
  - 水缸和水井互斥，即水井每次容纳一个水桶进出，水缸也是如此，并不取水、倒水入缸、取水出缸，每次用水桶1个；水缸中可以装水10桶。

2.5 进程同步应用示例-和尚喝水问题

- 变量设置：
  - mutex1：用于实现对水井的互斥使用，初值=1；
  - mutex2：用于实现对水缸的互斥使用，初值=1；
  - empty：用于记录水缸还可以装入的桶数，初值=10；
  - full：用于记录水缸中已装入的桶数，初值=0；
  - count：用于记录可用水桶数，初值=3。

- 进程描述：
  - Get：从井中取水入缸；
  - Use：从缸中取水引用。

2.5 进程同步应用示例-和尚喝水问题

Get:（小和尚）

P (　　);

P (　　);

P (　　);

井中取水;

V (　　);

P (　　);

倒水入缸;

V (　　);

V (　　);

V (　　);

mutex1：对水井的互斥使用，1
mutex2：对水缸的互斥使用，1
empty：缸可以装入的桶数，10
full：缸中已装入的桶数，0
count：可用水桶数，3

Use:（老和尚）

P (　　);

P (　　);

P (　　);

缸中取水;

V (　　);

V (　　);

V (　　);

2.5 进程同步应用示例-和尚喝水问题

Get:（小和尚）

P (empty);

P (count);

P (mutex1);

井中取水；

V (mutex1);

P (mutex2);

倒水入缸；

V (mutex2);

V (count);

V (full);

mutex1：对水井的互斥使用，1
mutex2：对水缸的互斥使用，1
empty：缸可以装入的桶数，10
full：缸中已装入的桶数，0
count：可用水桶数，3

Use:（老和尚）

P (full);

P (count);

P (mutex2);

缸中取水；

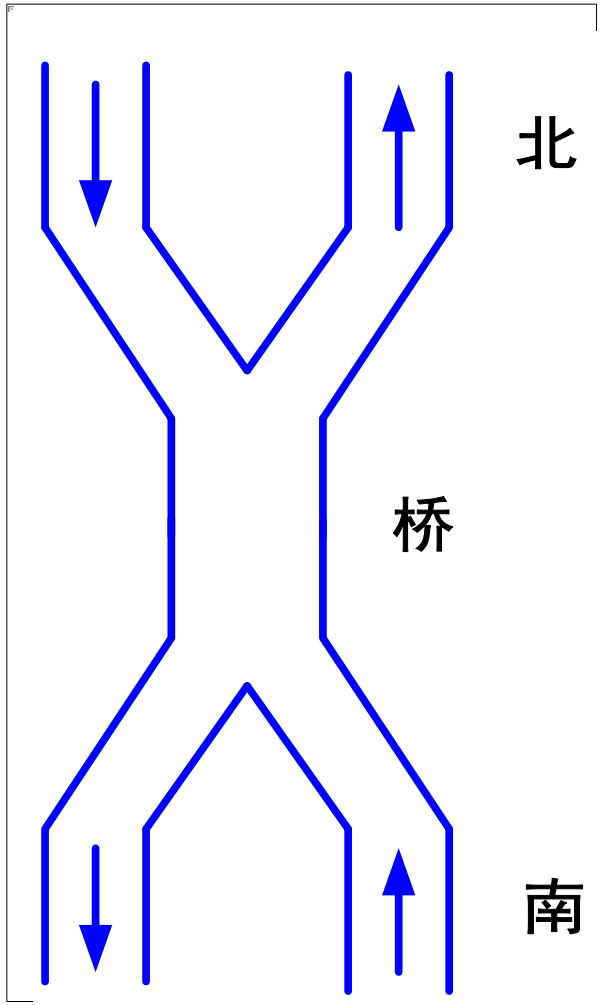
V (mutex2);

V (empty);

V (count);

2.5 进程同步应用示例-过桥问题

- **题目：**
  - 有一座桥如图所示，车流如图中箭头所示。桥上不允许两车交会，但允许同方向多辆车依次通行（即桥上可以有多个同方向的车）。
  - 用P、V操作实现交通管理，以防止桥上堵塞。



2.5 进程同步应用示例-过桥问题

- 变量设置：
  - mutexN：用于实现北方车辆互斥访问变量countN，初值=1；
  - mutexS：用于实现南方车辆互斥访问变量countS，初值=1；
  - wait：用于实现双方申请过桥车辆的排队，初值=1；
  - countN：用于记录当前北方正在过桥及已申请过桥的车辆数，初值=0；
  - countS：用于记录当前南方正在过桥及已申请过桥的车辆数，初值=0；
- 进程描述：
  - North、South：分别代表北方、南方车辆过桥的进程

North:

```
P ( wait );
P ( mutexN );//确保对countN的互斥访问
if (countN==0)
    P( mutexS)//本方向第一辆车，阻塞对方车辆过桥
countN ++;
V ( mutexN );//确保对countN的互斥访问
V ( wait );
  车辆过桥;
P ( mutexN );
countN --;
if (countN==0)
    V( mutexS)//本方向最后一辆车允许对方车辆过桥
V ( mutexN );
```

mutexN：北方车辆互斥访问变量countN，1

mutexS：南方车辆互斥访问变量countS，1

wait：双方申请过桥车辆的排队，1

countN：北方正在过桥及已申请过桥的车辆数，0

countS：南方正在过桥及已申请过桥的车辆数，0

South:

```
P ( wait );
P ( mutexS );
if (countS==0)
    P( mutexN)//本方向第一辆车，阻塞对方车辆过桥
countS++;
V ( mutexS);
V ( wait );
车辆过桥;
P ( mutexS );
countS--;
if (countS==0)
    V( mutexN)//本方向最后一辆车允许对方车辆过桥
V ( mutexS);
```

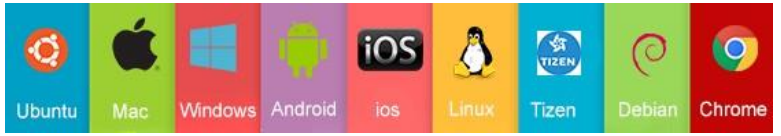
mutexN：北方车辆互斥访问变量countN，1

mutexS：南方车辆互斥访问变量countS，1

wait：双方申请过桥车辆的排队，1

countN：北方正在过桥及已申请过桥的车辆数，0

countS：南方正在过桥及已申请过桥的车辆数，0



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

2.6.1 进程通信的类型

◆ 进程通信——进程之间的信息交换

- 进程之间的互斥和同步，由于其所交换的信息量少而被归结为**低级通信**。
- 信号量及管程等**低级进程通信**存在不足之处
  - **效率低：**生产者每次只能向缓冲池投放一个产品(消息)，消费者每次只能从缓冲区中取得一个消息；
  - **通信对用户不透明可见：**用户要利用低级通信工具实现进程通信是非常不方便的。因为共享数据结构的设置、数据的传送、进程的互斥与同步等，都必须由程序员去实现，操作系统只能提供共享存储器。
  - **不适用多处理器的情况**

2.6.1 进程通信的类型

◆ 进程通信——进程之间的信息交换

➢ 高级通信工具的特点

- 使用方便：向用户提供了一组用于实现高级通信的命令（**原语**），用户可以直接利用它实现进程之间的通信。
- **高效**的传送大量数据：用户可以利用高级通信命令高效地传输大量数据

2.6.1 进程通信的类型

- **进程通信的类型**
  - **共享存储器系统（shared-memory system）**
  - **管道（piple）通信系统**
  - **消息传递系统（message passing system）**
  - **客户机-服务器系统（client-server system ）**

2.6.1 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

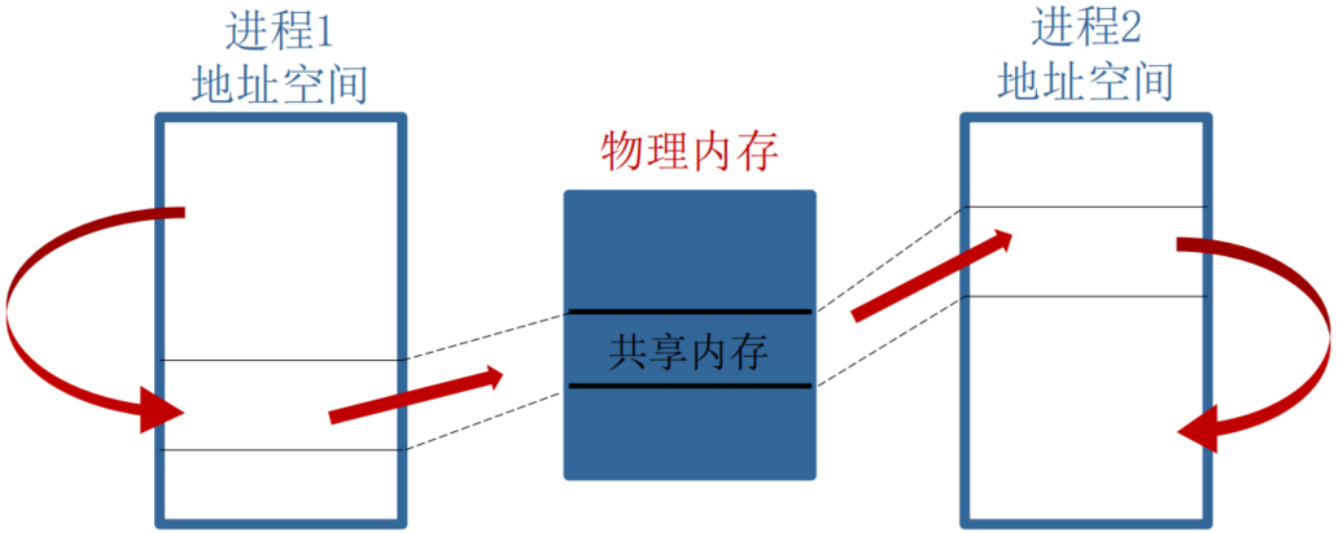
- 要求诸进程公用某些数据结构，借以实现诸进程间的信息交换。
- 在生产者—消费者问题中，就是用有界缓冲区这种数据结构来实现通信的。
- 通信方式是低效的，只适于传递相对少量的数据。

2.6.1 进程通信的类型

◆ 共享存储器系统

➢ 基于共享存储区的通信方式

- 为了传输大量数据，在存储器中划出了一块共享存储区，诸进程可通过对共享存储区中数据的读或写来实现通信。
- 高级通信，进程申请地址空间



2.6.1 进程通信的类型

◆ 管道通信系统

- 利用一个缓冲传输介质（内存或文件）连接两个相互通信的进程；字符流方式写入读出，先进先出顺序
- 管道通信机制必须提供的协调能力
  - **互斥：**即当一个进程正在对 pipe 执行读/写操作时，其它(另一)进程必须等待。
  - **同步：**指当写(输入)进程把一定数量(如 4 KB)的数据写入 pipe，便去睡眠等待，直到读(输出)进程取走数据后，再把它唤醒。当读进程读一空 pipe 时，也应睡眠等待，直至写进程将数据写入管道后，才将之唤醒。
  - **确定对方是否存在：**只有确定了对方已存在时，才能进行通信。

2.6.1 进程通信的类型

◆ 客户机-服务器系统

➢ 套接字

- 起源于Unix，是一个网络通信接口
- 包含通信目的地址，端口号，网络传输层协议，进程所在的网络地址，系统调用
- 适用于同一主机内部通信，也适用于不同主机间的进程通信

➢ 远程过程调用和远程方法调用

- 是一个通信协议，用于通过网络连接的系统
- 允许本地系统上的进程调用远程系统上的进程，对程序表现为常规的过程调用

2.6.1 进程通信的类型

◆ 消息传递系统

- 不借用共享存储区或数据结构，以格式化的消息（message）为单位在进程间进行消息传递
- 在进程之间通信时，源进程可以直接或间接地将消息传送给目标进程，进一步分类：
  - **直接通信方式：**直接消息传递系统(利用发送原语直接把消息发送给目标进程)
  - **间接通信方式：**信箱通信(通过共享中间实体进行通信)

2.6.1 消息传递通信的实现

◆ 直接消息传递系统

- ➢ 发送进程利用OS所提供的发送命令，直接把消息发送给目标进程。
- ➢ **对称寻址方式：**要求发送进程和接收进程都以显式方式提供对方的标识符，系统提供下述两条通信命令（原语）
  - Send(receiver，message)； 发送一个消息给接收进程；
  - Receive(sender，message)； 接收 Sender 发来的消息；
- ➢ **非对称寻址方式：**接收进程可与多个发送进程通信，因此，它不可能事先指定发送进程。
  - Send(P，message)； 发送一个消息给接收进程；
  - Receive(id，message)； 接收任何进程发来的消息；

2.6.1 消息传递通信的实现

◆ 直接消息传递系统

- ➢ **发送进程利用OS所提供的发送命令，直接把消息发送给目标进程。**
- ➢
- ➢

- **Receive(id，message)； 接收任何进程发来的消息；**

2.6.1 消息传递通信的实现

◆ 直接消息传递系统

➢ 消息的格式

- 在单机系统环境中，由于发送进程和接收进程处于同一台机器中，有着相同的环境，故其消息格式比较简单；
- 在计算机网络环境下，不仅源和目标进程所处的环境不同，而且信息的传输距离很远，可能要跨越若干个完全不同的网络，致使所用的消息格式比较复杂。
- 较短的**定长消息格式**，减少对消息的处理和存储开销；**变长的消息格式**，即进程所发送消息的长度是可变的，会付出更多的开销，但这方便了用户。这两种消息格式各有其优缺点，故在很多系统(包括计算机网络)中，是同时都用的。

2.6.1 消息传递通信的实现

◆ 直接消息传递系统

➢ 进程的同步方式

- **发送进程阻塞，接收进程阻塞：**进程之间紧密同步(tight synchronization)，发送进程和接收进程之间无缓冲时。这两个进程平时都处于阻塞状态，直到有消息传递时。这种同步方式称为**汇合(rendezrous)**。
- **发送进程不阻塞，接收进程阻塞：**平时，发送进程不阻塞，因而它可以尽快地把一个或多个消息发送给多个目标； 而接收进程平时则处于阻塞状态，直到发送进程发来消息时才被唤醒。
- **发送进程和接收进程均不阻塞：**平时，发送进程和接收进程都在忙于自己的事情，仅当发生某事件使它无法继续运行时，才把自己阻塞起来等待。

2.6.1 消息传递通信的实现

◆ 直接消息传递系统

➢ 通信链路

- 为使在发送进程和接收进程之间能进行通信，必须在两者之间建立一条**通信链路 (communication link)**
- 两种建立通信链路的方式
  - ✓ 由发送进程在通信之前用**显式的“建立连接”**命令(原语)请求系统为之建立一条通信链路；在链路使用完后，也用显式方式拆除链路。这种方式主要用于计算机网络中。
  - ✓ 第二种方式是发送进程**无须明确提出建立链路的请求**，只须利用系统提供的发送命令(原语)，系统会自动地为之建立一条链路。这种方式主要用于单机系统中。

2.6.1 消息传递通信的实现

◆ 直接消息传递系统

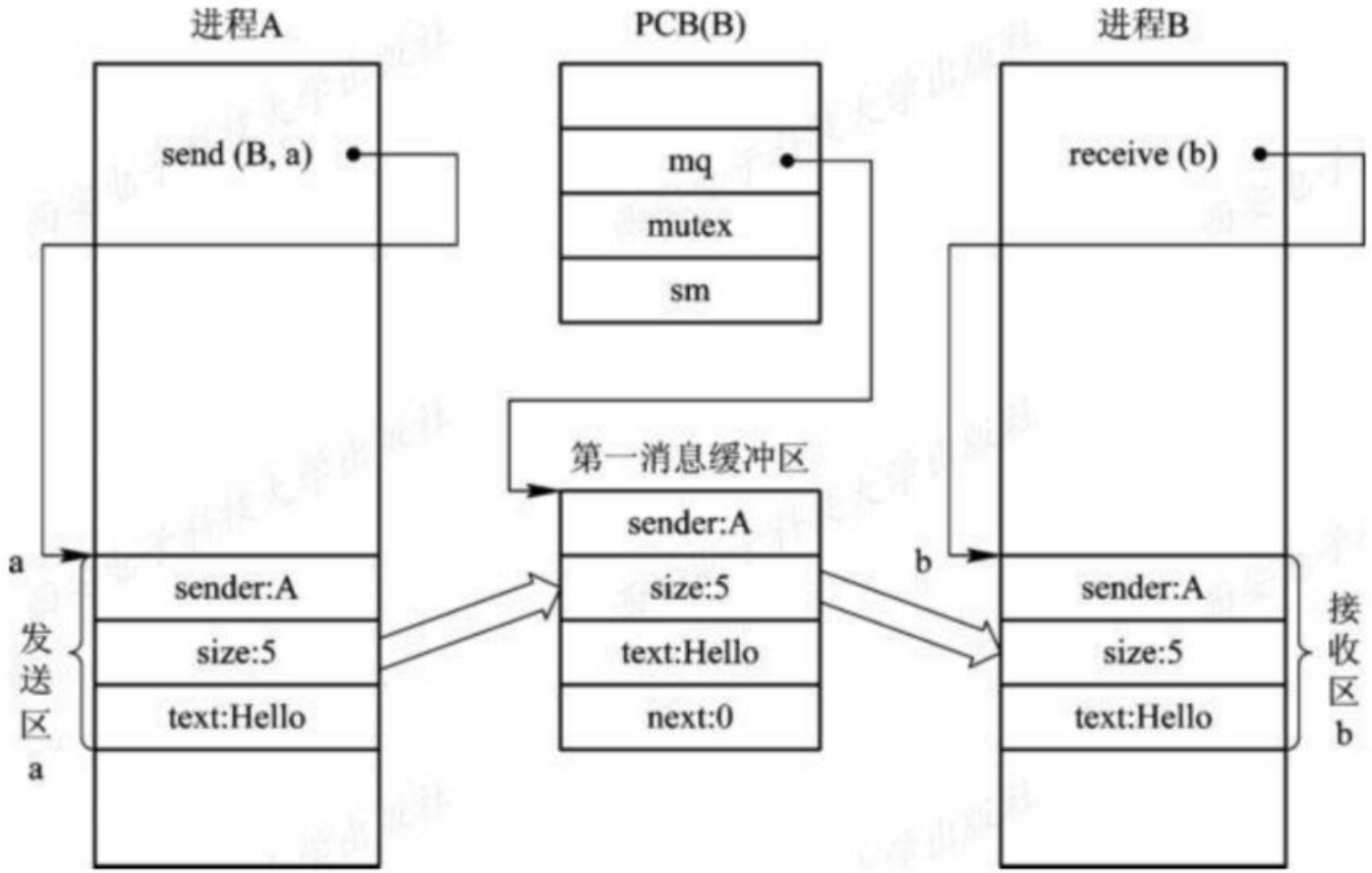
➢ 通信链路

- 而根据**通信方式**的不同，则又可把链路分成两种：
  - ✓ **单向链路**：只允许发送进程向接收进程发送消息，或者相反；
  - ✓ **双向链路**：既允许由进程 A 向进程 B 发送消息，也允许进程 B 同时向进程 A 发送消息。
- 还可根据**通信链路容量的**不同而把链路分成两类：
  - ✓ **无容量通信链路**：在这种通信链路上没有缓冲区，因而不能暂存任何消息；
  - ✓ **有容量通信链路**：指在通信链路中设置了缓冲区，因而能暂存消息。缓冲区数目愈多，通信链路的容量愈大。

2.6.1 消息传递通信的实现

◆ 消息传递系统

➢ 直接消息传递系统实例



2.6.1 消息传递通信的实现

◆ 直接消息传递系统实例

➢ 消息缓冲队列通信机制中的数据结构

- **消息缓冲区**

```
typedef struct message_buffer{
    int sender;    //发送者进程标识符
    int size;      //消息长度
    char *text;    //消息正文
    struct message_buffer *next; //指向下一个消息缓冲区的指针
}
```

- **PCB中有关通信的数据项**

```
typedef struct processcontrol_block{
    ...
    struct message_buffer *mq;  //消息队列队首指针
    semaphore mutex;  //消息队列互斥信号量
    semaphore sm;  //消息队列资源信号量
    ...
}
```

2.6.1 消息传递通信的实现

◆ 直接消息传递系统实例

➢ 发送原语Send

- 根据发送进程中的消息长度申请缓冲区
- 将发送进程中的信息复制到申请到的缓冲区中
- 把消息缓冲区挂到接收进程消息队列的末尾

```
void send(receiver, a){
    getbuf(a.size, i);
    i.sender = a.sender;
    i.size = a.size;
    copy(i.text, a.text);
    i.next = 0;
    getid(PCBset, receiver.j);
    wait(j.mutex);
    insert(&j.mq, i);
    signal(j.mutext);
    signal(j.sm);
}
```

2.6.1 消息传递通信的实现

◆ 直接消息传递系统实例

➢ 发送原语

```
procedure send(receiver， a)
  begin
    getbuf(a.size,i);              根据 a.size 申请缓冲区；
    i.sender:= a.sender;           将发送区 a 中的信息复制到消息缓冲区 i 中；
    i.size:=a.size;
    i.text:=a.text;
    i.next:=0;
    getid(PCB set，receiver.j);    获得接收进程内部标识符；
    wait(j.mutex);
    insert(j.mq，i);               将消息缓冲区插入消息队列；
    signal(j.mutex);
    signal(j.sm);
end
```

2.6.1 消息传递通信的实现

◆ 直接消息传递系统实例

➢ 接收原语

```
procedure receive(b)
   begin
      j:= internal name;          j 为接收进程内部的标识符;
      wait(j.sm);
      wait(j.mutex);
      remove(j.mq，i);            将消息队列中第一个消息移出;
      signal(j.mutex);
      b.sender:=i.sender;         将消息缓冲区 i 中的信息复制到接收区 b;
      b.size:=i.size;
      b.text:=i.text;
end
```

2.6.1 消息传递通信的实现

◆ 间接消息传递系统：信箱通信

- 间接通信方式指进程之间的通信需要通过作为**共享数据结构的实体**。
- 该实体用来暂存发送进程发送给目标进程的消息；接收进程则从该实体中取出对方发送给自己的消息。通常把这种中间实体称为**信箱**。
- 消息在信箱中可以安全地保存，只允许核准的目标用户随时读取。
- 利用信箱通信方式，既可实现**实时通信**，又可实现**非实时通信**。

2.6.1 消息传递通信的实现

◆ 间接消息传递系统：信箱通信

➢ 系统为信箱通信提供了若干条**原语**：

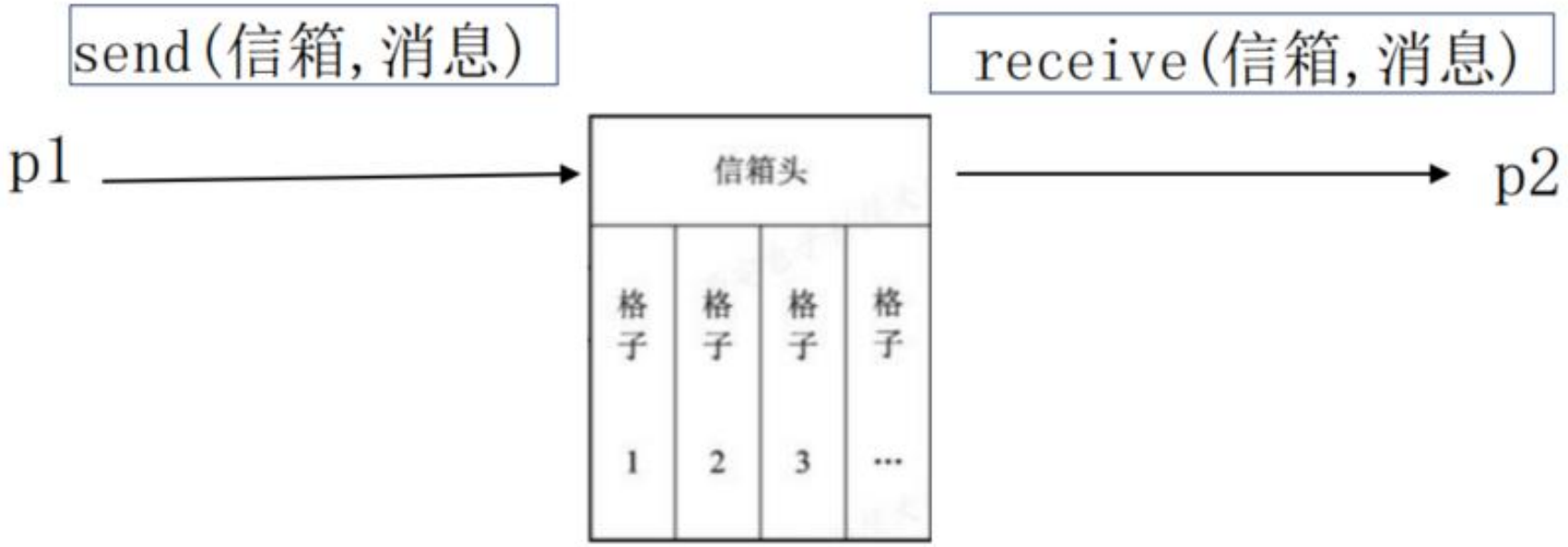
- **信箱的创建和撤消**：进程可利用信箱创建原语来建立一个新信箱。创建者进程应给出信箱名字、信箱属性(公用、私用或共享)；对于共享信箱，还应给出共享者的名字。当进程不再需要读信箱时，可用信箱撤消原语将之撤消。
- **消息的发送和接收**：当进程之间要利用信箱进行通信时，必须使用共享信箱，并利用系统提供的下述通信原语进行通信：
  - ✓ Send(mailbox，message)； 将一个消息发送到指定信箱；
  - ✓ Receive(mailbox，message)； 从指定信箱中接收一个消息；

2.6.1 消息传递通信的实现

◆ 间接消息传递系统实例

➢ 信箱通信

- 信箱头
- 信箱体



2.6.1 消息传递通信的实现

◆ 间接消息传递系统：信箱通信

- 信箱根据创建者的不同分为以下三类：
- **私用信箱：** 用户进程可为自己建立一个新信箱，并作为该进程的一部分。信箱的拥有者有权从信箱中读取消息，其他用户则只能将自己构成的消息发送到该信箱中。
- **公用信箱：** 它由操作系统创建，并提供给系统中的所有核准进程使用。核准进程既可把消息发送到该信箱中，也可从信箱中读取发送给自己的消息。
- **共享信箱：** 它由某进程创建，在创建时或创建后指明它是可共享的，同时须指出共享进程(用户)的名字。信箱的拥有者和共享者都有权从信箱中取走发送给自己的消息。

2.6.1 消息传递通信的实现

◆ 间接消息传递系统：信箱通信

- 在利用信箱通信时，在发送进程和接收进程之间存在以下四种关系：
- **一对一关系**：可为发送进程和接收进程建立一条两者专用的通信链路，使两者之间的交互不受其他进程的干扰。
- **多对一关系**：允许提供服务的进程与多个用户进程之间进行交互，也称为客户/服务器交互(client/server interaction)。
- **一对多关系**：允许一个发送进程与多个接收进程进行交互，使发送进程可用广播方式向接收者(多个)发送消息。
- **多对多关系**：允许建立一个公用信箱，让多个进程都能向信箱中投递消息；也可从信箱中取走属于自己的消息。


THEORY
PRACTICAL