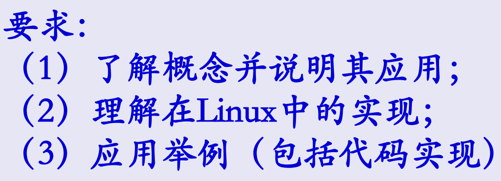
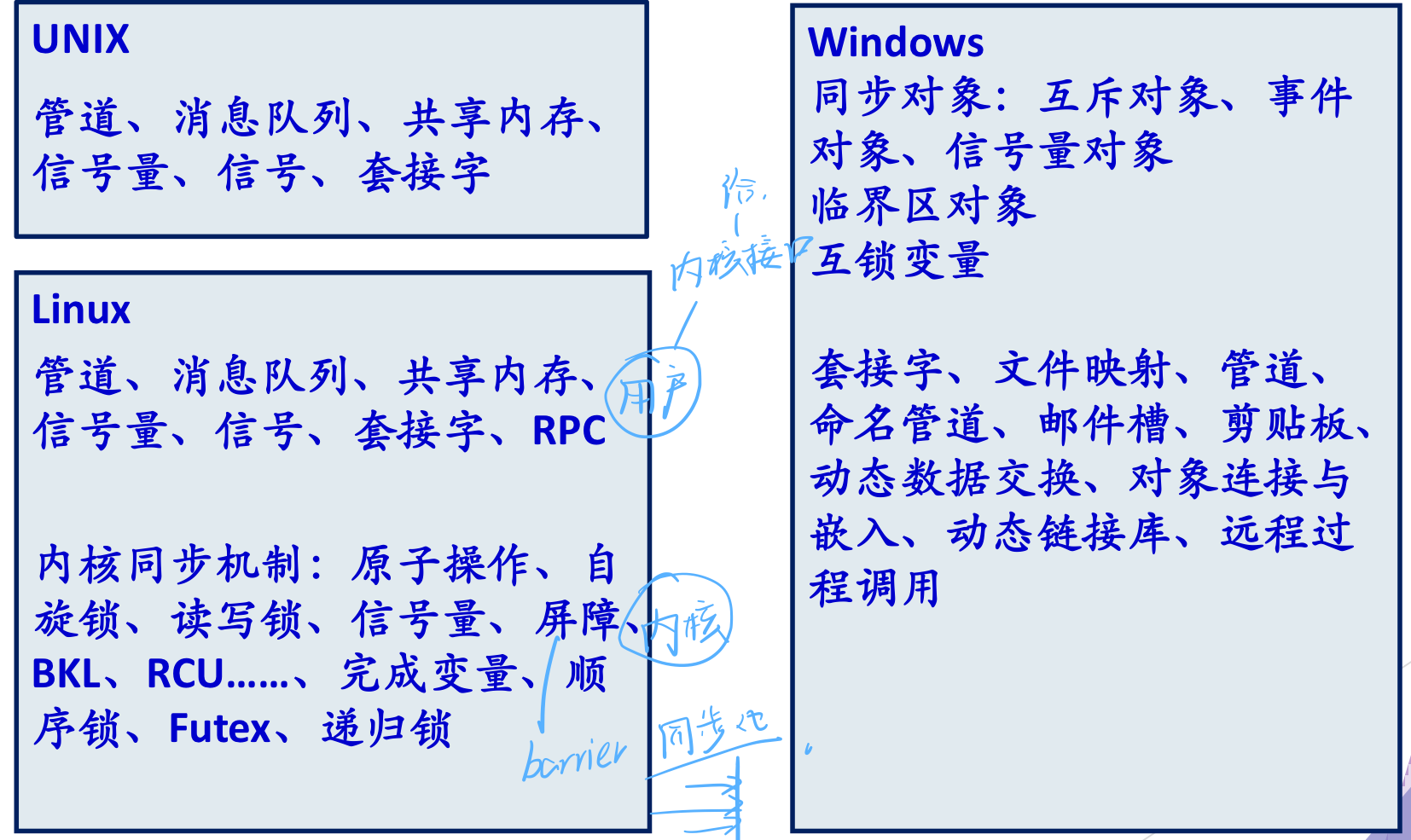
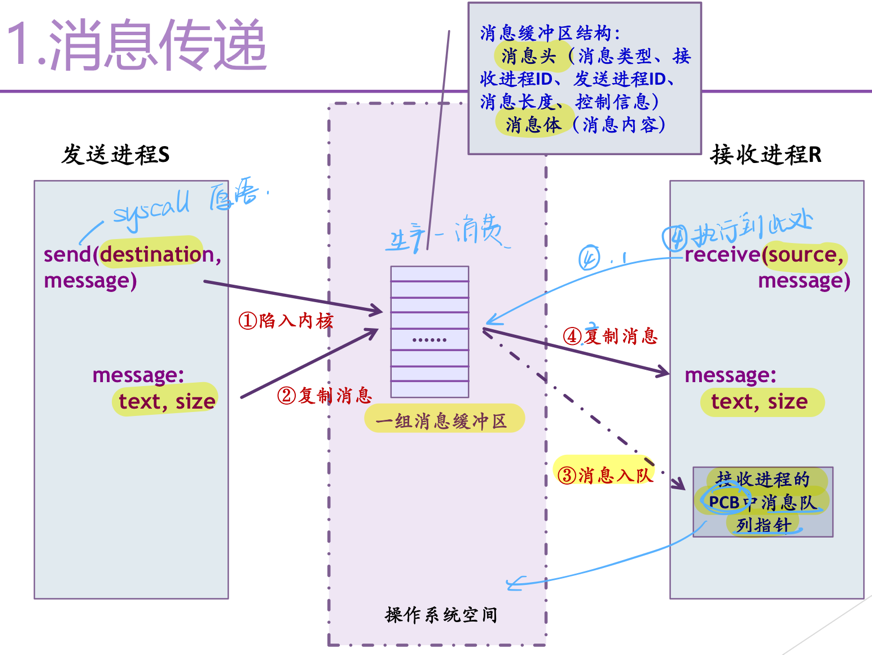
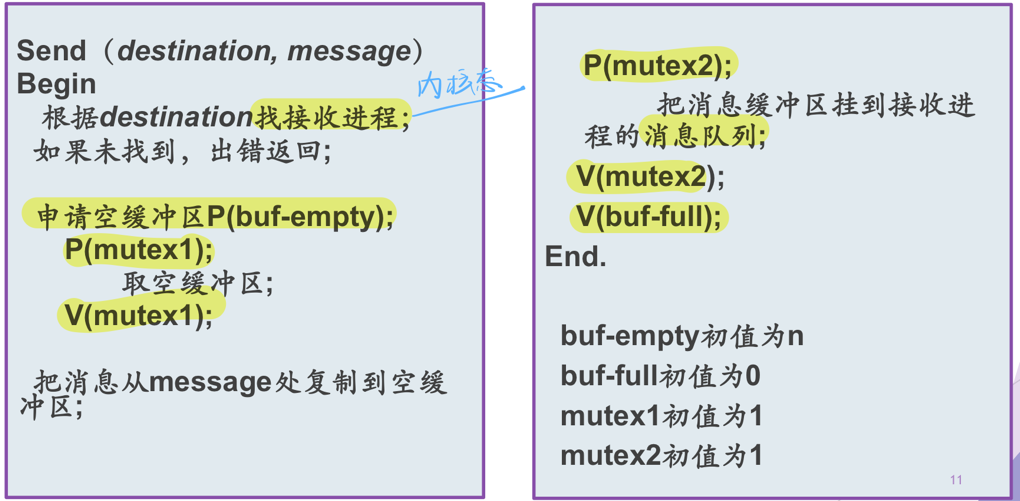
**Lecture11 进程间通信**

**Linux的IPC机制列表**

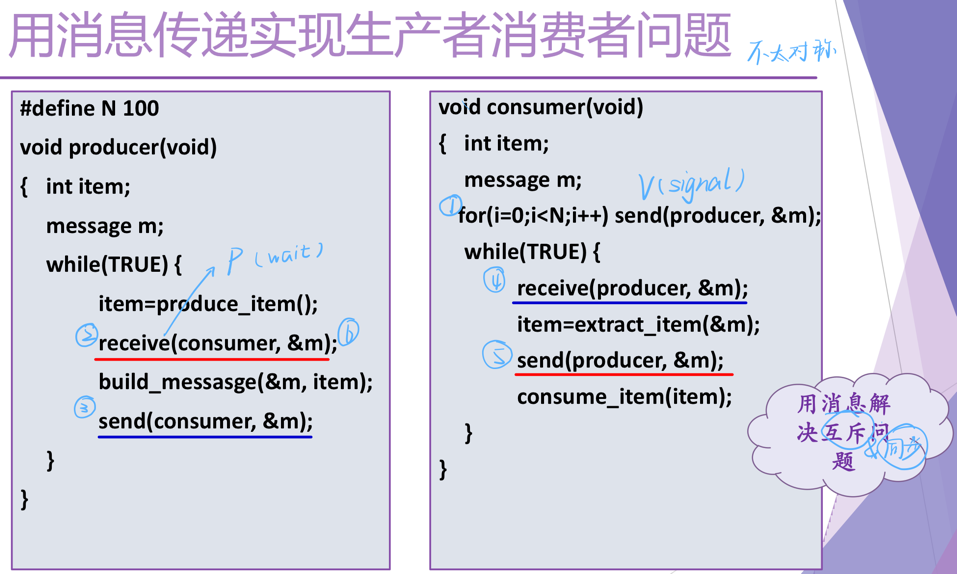
1. 原子操作
2. 管道
3. 信号
4. 套接字
5. 信号量
6. 消息队列
7. 共享内存
8. 自旋锁
9. 读写锁
10. 屏障
11. 完成变量
12. 顺序锁
13. RCU机制
14. Futex
15. 递归锁

**进程间同步/通信实例**

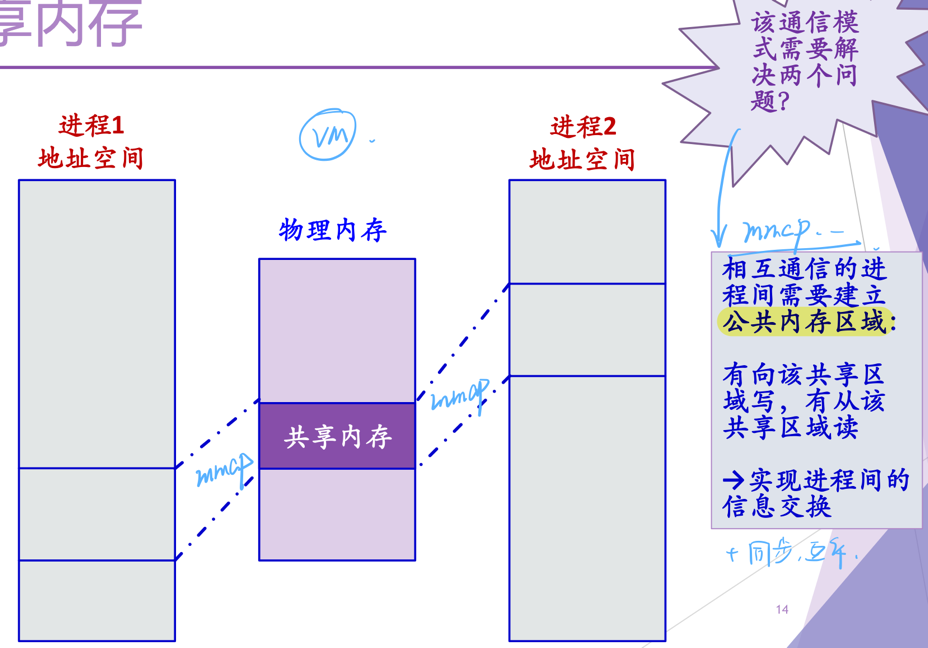
**进程间通信（IPC，Inter-Process Communication）**

1. 基本思想
2. 为什么需要通信机制？
3. 信号量及管程的不足
4. 多处理器情况下原语失效
5. 进程通信机制
6. 消息传递send & receive原语
7. 适用于分布式系统、基于共享内存的多处理机系统、单处理机系统
8. 可以解决进程间的同步问题、通信问题（支持大量消息的传递）
9. 基本通信方式
10. 消息传递
11. 基本机制
12. 用P、V操作实现Send原语

思考题：receive原语的实现

1. 用消息传递解决生产者消费者问题
2. 实现消息传递机制需要考虑的其他问题

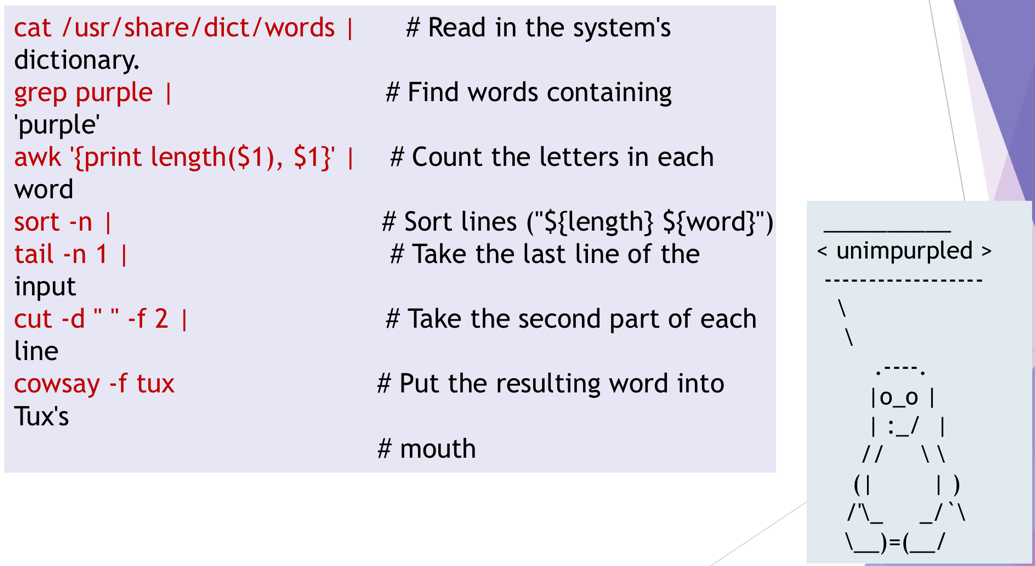
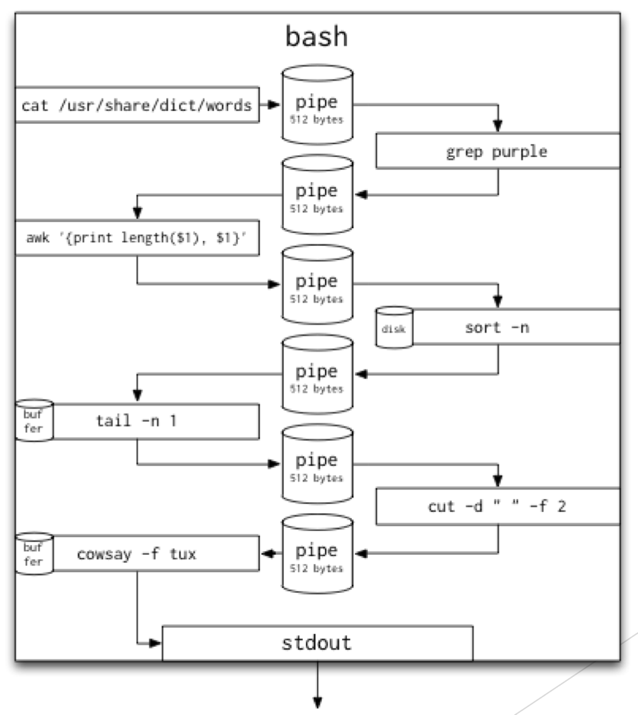
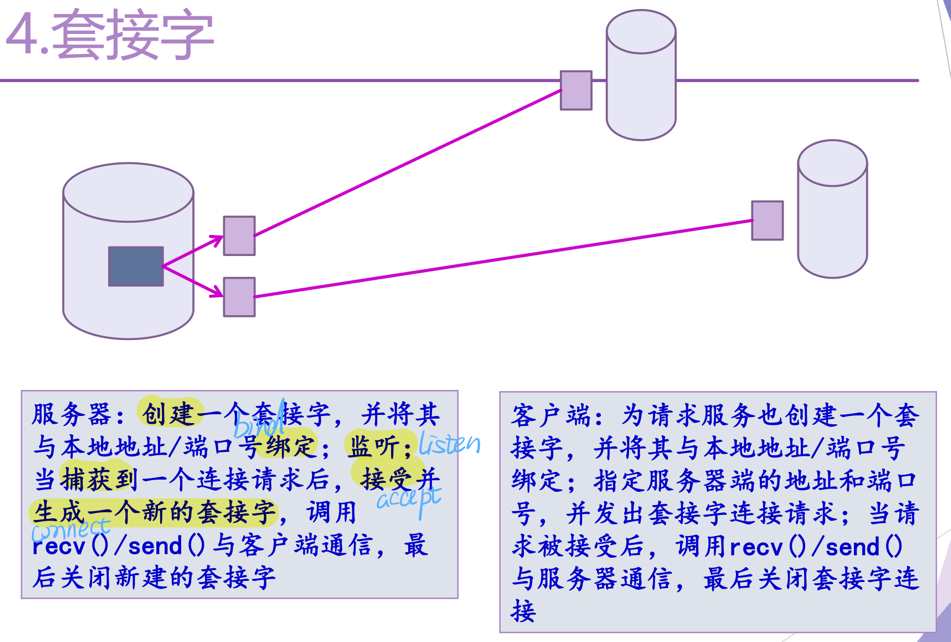
* 缓冲形式：信箱、无缓冲、无限缓冲
* 消息丢失（如在网络和分布式系统中）、身份识别、效率

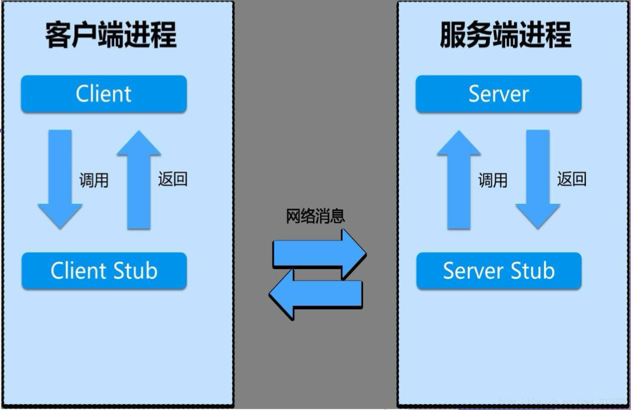
1. 共享内存
2. 管道（pipe）
3. 简介

* 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、同步以及能够判断对方进程是否存在）

1. 设计目标

高内聚、低耦合；它以一种“链式模型”来串接不同的程序或者不同的组件，让它们组成一条工作流；给定一个完整的输入，经过各个组件的先后协同处理，得到唯一的最终输出

1. 应用
2. 无名管道和命名管道
3. 套接字
4. 远程过程调用(RPC)
5. 简介

* 定义
* RPC（Remote Procedure Call）是一种技术思想，屏蔽网络编程细节，像调用本地方法一样调用远程方法
* 应用
* 应用访问量增加和业务增加时，单机无法承受，可以根据不同的业务拆分成互不关联的应用，分别部署在不同的机器上，应用与应用相互调用，此时需要用到 RPC。解耦服务、扩展性强、部署灵活，主要解决分布式系统中，服务与服务之间的调用问题

1. 一般过程

* 客户端Client通过本地调用的方式，调用远程接口服务
* 客户端存根Client Stub接收到调用后，将调用信息对象进行序列化，组装成网络传输的二进制消息体
* 客户端Client通过Socket将消息发送到远程服务端
* 服务端存根Server Stub收到消息后，对网络信息对象进行反序列化解码
* 服务端存根Server Stub根据解码结果，调用服务端本地的接口服务
* 本地接口服务执行，并将处理结果返回给服务端存根Server Stub
* 服务端存根Server Stub将返回结果对象进行序列化，组装成消息体
* 服务端Server再通过Socket将消息发送到客户端
* 客户端存根Client Stub收到结果消息后，对网络信息对象进行序列化解码
* 客户端Client拿到最终接口处理结果

1. 具体应用

* 电商系统

将电商系统拆分成用户服务、商品服务、优惠券服务、支付服务、订单服务、物流服务、售后服务等，这些服务之间都相互调用，内部调用使用RPC，同时每个服务都可以独立部署，独立上线

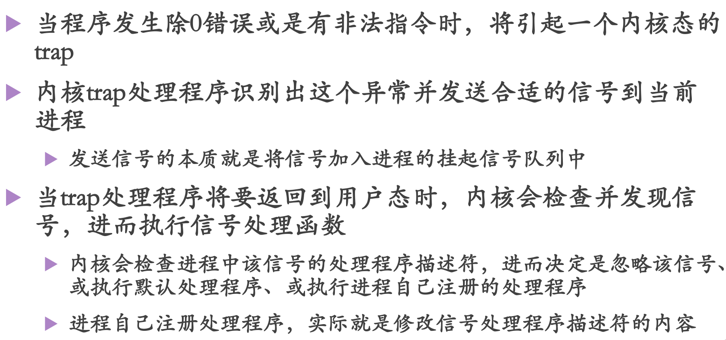
* 车载以太网

SOME/IP(Scalable service-Oriented MiddlewarE over IP)是基于IP的可扩展的面向服务的中间件，2011年由BMW集团的Dr. Lars Völker设计，是一种面向服务的车载以太网通信协议，采用Client/Server通信架构，其中Server是服务提供者，Client是服务消费者。根据服务接口类型，使用远程过程调用（Remote Procedure Call）机制，实现控制器之间的服务调用，通过数据序列化和反序化（Serialization/Deserialization）使数据在网络中传输，通过服务发现SD（Service Discovery）机制来实现服务的动态配置

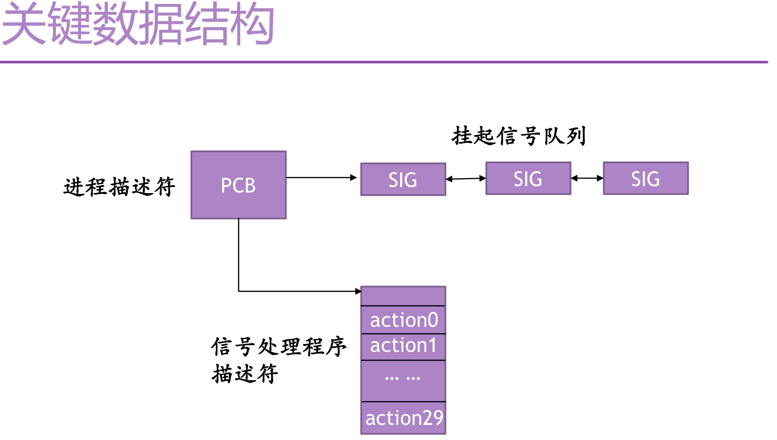
1. 信号（signal）
2. 概念

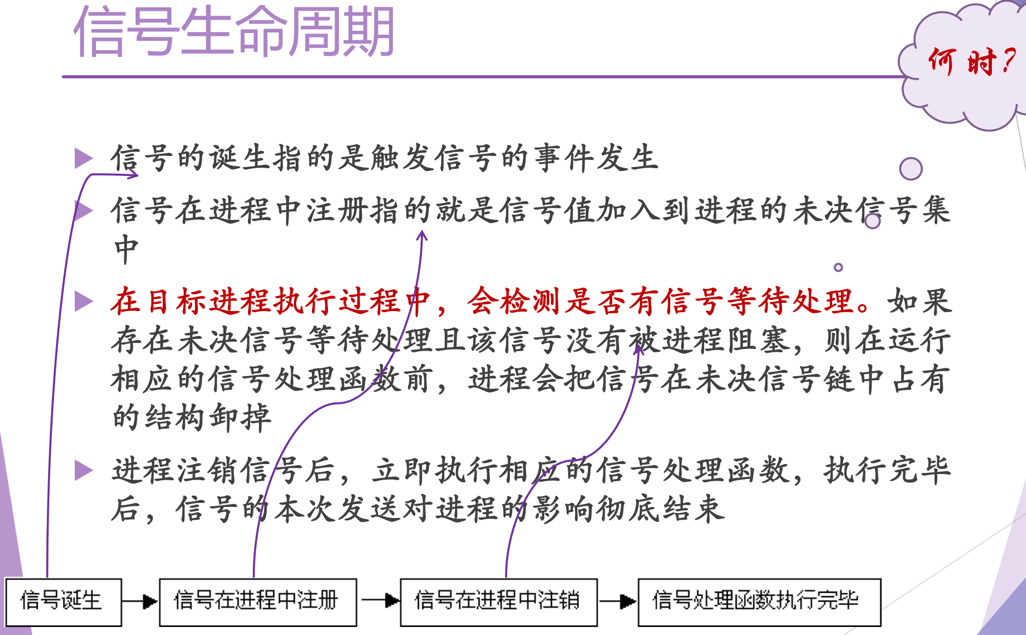
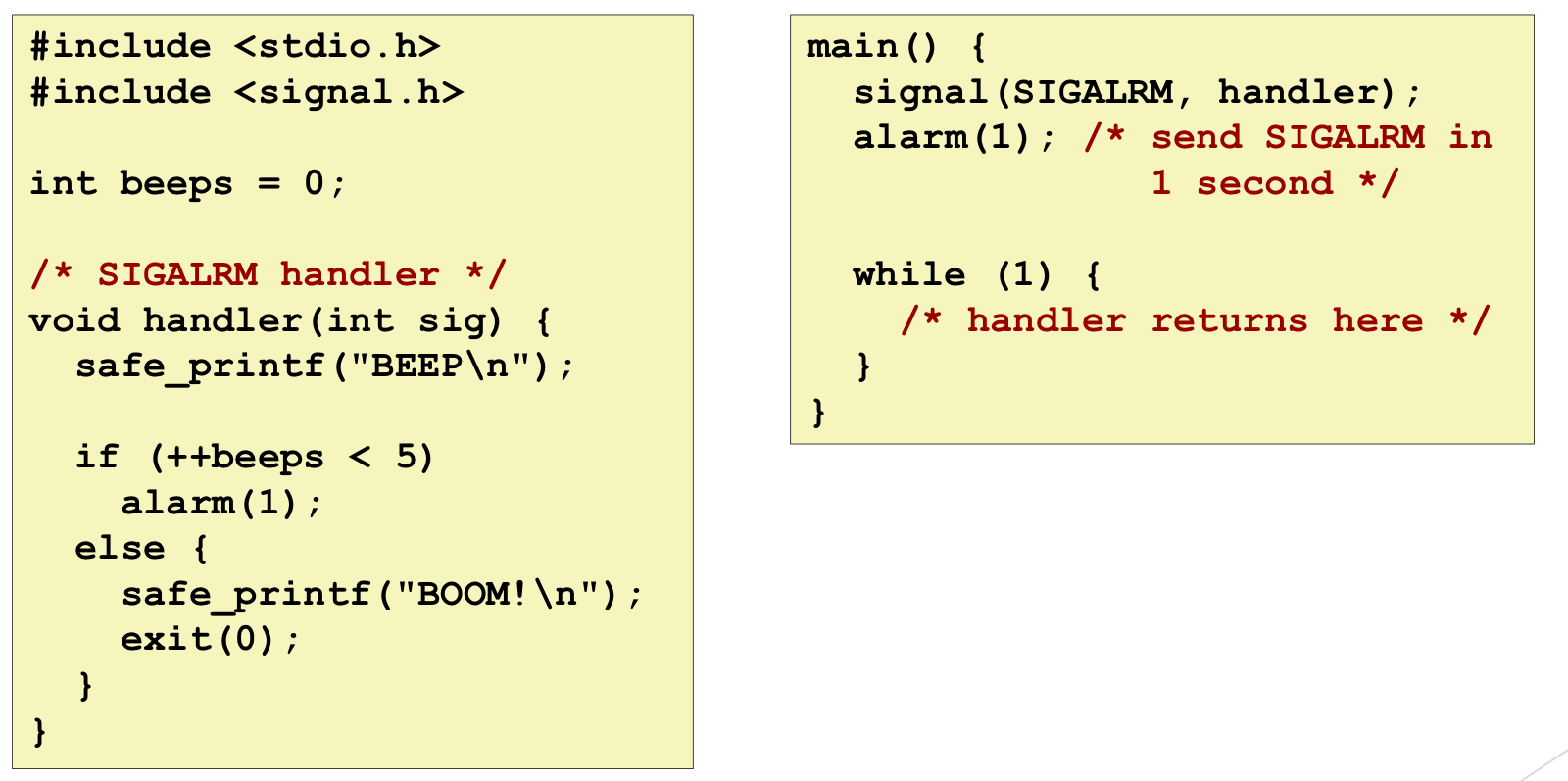
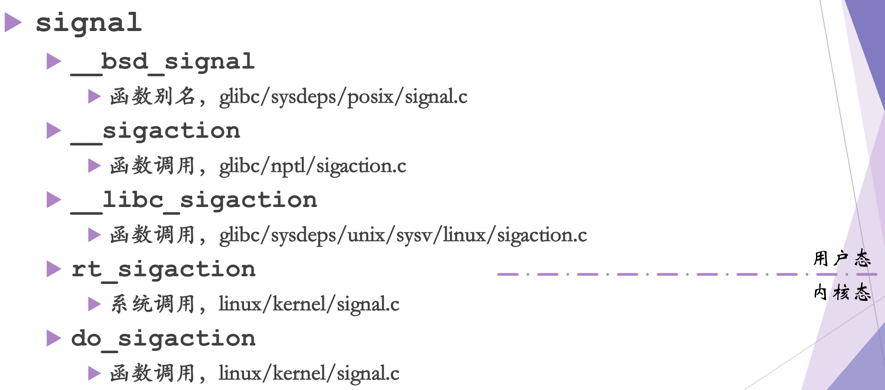
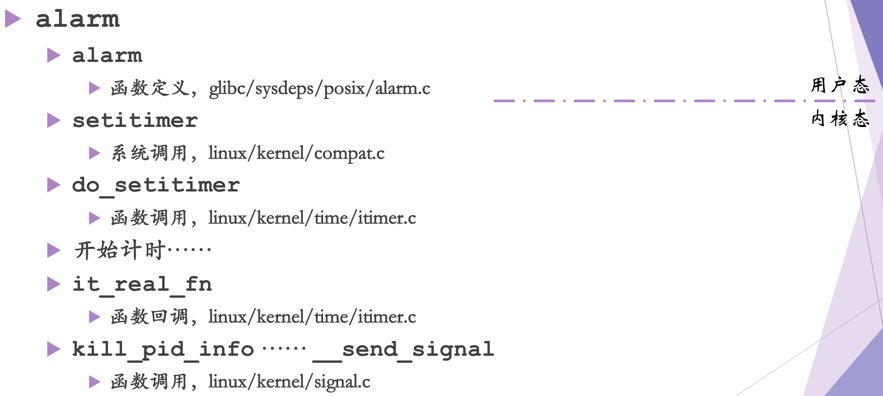
* 软中断信号简称为信号，用来通知进程发生了异步事件
* 信号本质上是在软件层次上对中断机制的一种模拟
* 进程之间可以互相通过系统调用发送软中断信号
* 内核也可以因为内部事件而给进程发送信号，通知进程发生了某个事件

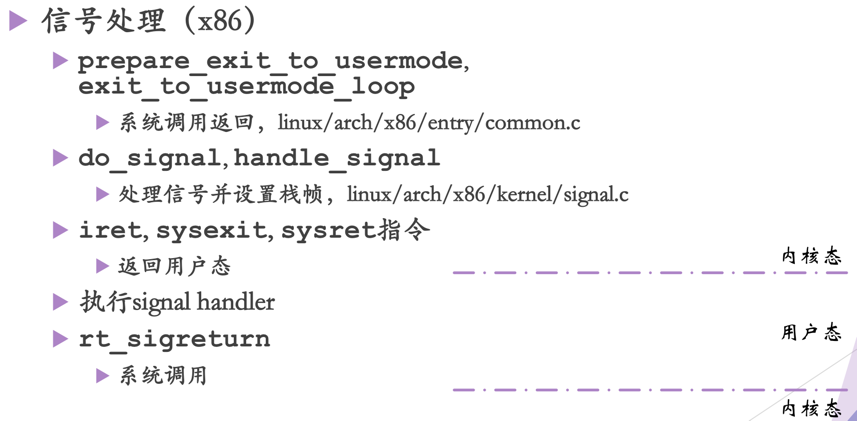
1. 信号的产生

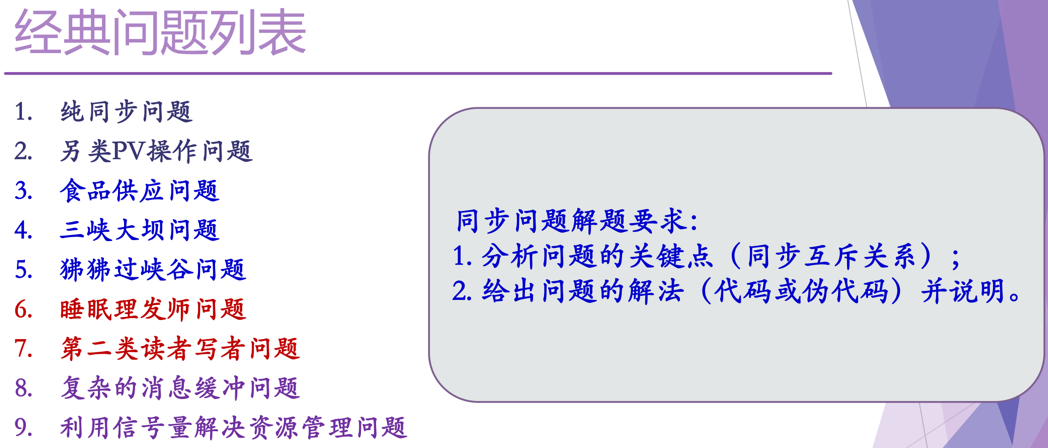
* 异常
* 当一个进程出现异常（比如试图执行一个非法指令，除0，浮点溢出等），内核通过向进程发送一个信号来通知进程异常的发生
* 例子
* 其他进程
* 一个进程可以通过kill或是sigsend系统调用向另一个进程或一个进程组发送信号。一个进程也可以向自身发送信号
* 作业控制
* 发送信号给那些想要读或写终端的后台进程。比如shell使用信号来管理前台和后台进程
* 通知
* 一个进程也许要求能被通知某些事件的发生。比如设备已经就绪等待I/O操作
* 闹钟
* 定时器产生的信号，由内核发送给进程
* ......

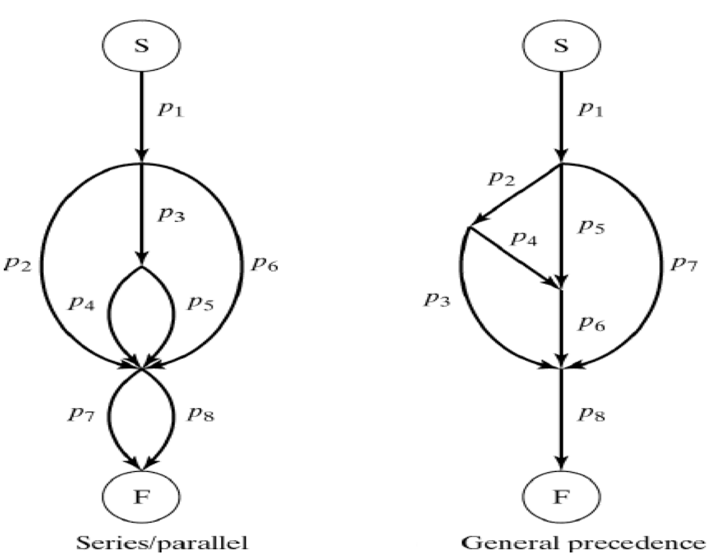
1. 信号的处理

* 收到信号的进程对各种信号有三类处理方法
* 对于需要处理的信号，进程可以指定处理函数，由该函数来处理
* 忽略某个信号，对该信号不做任何处理，就像未发生过一样
* 对该信号的处理保留系统的默认处理方式
* 检测并响应信号的时机
* 进程由于系统调用、中断或异常进入内核，当该进程从内核返回用户空间前
* 例子：进程被调度程序选中，由于检测到信号从而提前返回到用户空间
* 关键数据结构

1. 信号生命周期
2. Linux信号的实现



**经典IPC问题**

1. 纯同步问题
2. 另类PV操作问题

P(s):

s.count --;

if(s<0) then

将本进程插入相应队列末尾等待

V(s):

s.count ++;

if(s<=0) then

从相应等待队列末尾唤醒一个进程，将其插入就绪队列

1. 这样定义PV操作是否有问题
2. 用这样的PV操作实现N个进程竞争使用某一共享变量的互斥机制
3. 对b的解法，有无效率更高的方法
4. 食品供应问题

某商店有两种食品A和B， 最大数量各为m个。该商店将A、B两种食品搭配出售，每次各取一个。为避免食品变质，遵循先到食品先出售的原则。有两个食品公司分别不断地供应A、B两种食品(每次一个)。为保证正常销售，当某种食品的数量比另一种的数量超过k(k<m)个时，暂停对数量大的食品进货。试用P、V操作解决上述问题中的同步和互斥关系。

1. 三峡大坝问题

由于水面高度不同，有160~175米的落差，所以三峡大坝有五级船闸，T1~T5。由上游驶来的船需经由各级船闸到下游；由下游驶来的船需经由各级船闸到上游。假设只能允许单方向通行（此假设与实际情况不符，实际为双向各五级船闸）。试用P、V操作正确解决三峡大坝船闸调度问题。

1. 狒狒过峡谷问题
2. 一个主修人类学、辅修计算机科学的学生参加了一个研究课题，调查是否可以教会非洲狒狒理解死锁。他找到一处很深的峡谷，在上边固定了一根横跨峡谷的绳索，这样狒狒就可以攀住绳索越过峡谷。同一时刻，只要朝着相同的方向就可以有几只狒狒通过。但如果向东和向西的狒狒同时攀在绳索上那么会产生死锁（狒狒会被卡在中间），由于它们无法在绳索上从另一只的背上翻过去。如果一只狒狒想越过峡谷，它必须看当前是否有别的狒狒正在逆向通行。利用信号量编写一个避免死锁的程序来解决该问题。不考虑连续东行的狒狒会使得西行的狒狒无限制地等待的情况。
3. 重复上一个习题，但此次要避免饥饿。当一只想向东去的狒狒来到绳索跟前，但发现有别的狒狒正在向西越过峡谷时，它会一直等到绳索可用为止。但在至少有一只狒狒向东越过峡谷之前，不允许再有狒狒开始从东向西过峡谷。
4. 睡眠理发师问题
5. 理发店里有一位理发师，一把理发椅和N把供等候理发的顾客坐的椅子。如果没有顾客，则理发师便在理发椅上睡觉；当一个顾客到来时，他必须先唤醒理发师。如果顾客到来时理发师正在理发，则如果有空椅子，可坐下来等，否则离开。试用P、V操作解决睡眠理发师问题。
6. 思考：N个理发师的解决方案
7. 第二类读者写者问题（写者优先）

试用信号量及P、V操作解决写者优先问题，要求：

a. 多个读者可以同时进行读

b. 写者必须互斥(只允许一个写者写，也不能读者写者同时进行)

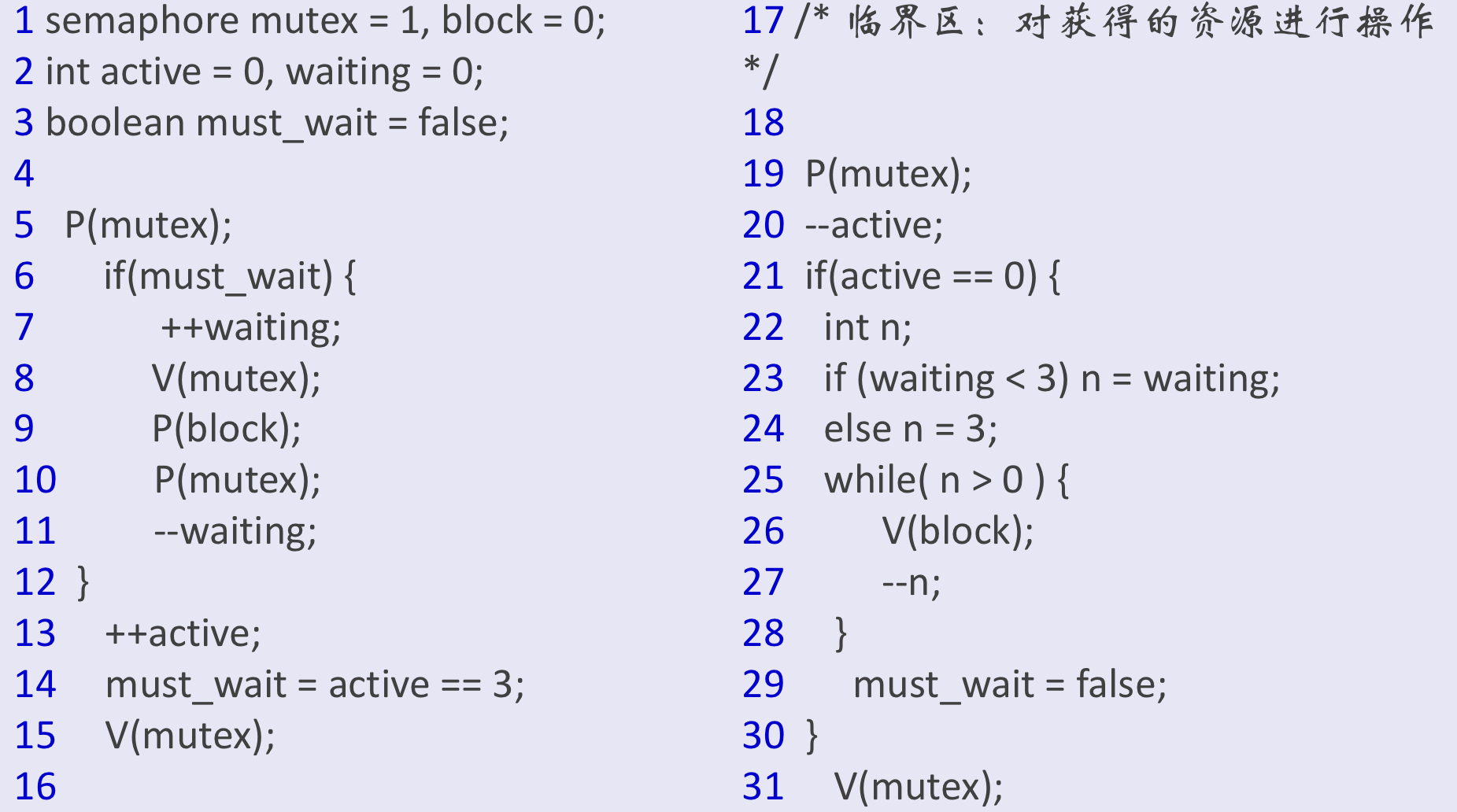
c. 写者优先于读者（一旦有写者，则后续读者必须等待，唤醒时优先考虑写 者）。

1. 复杂的消息缓冲问题

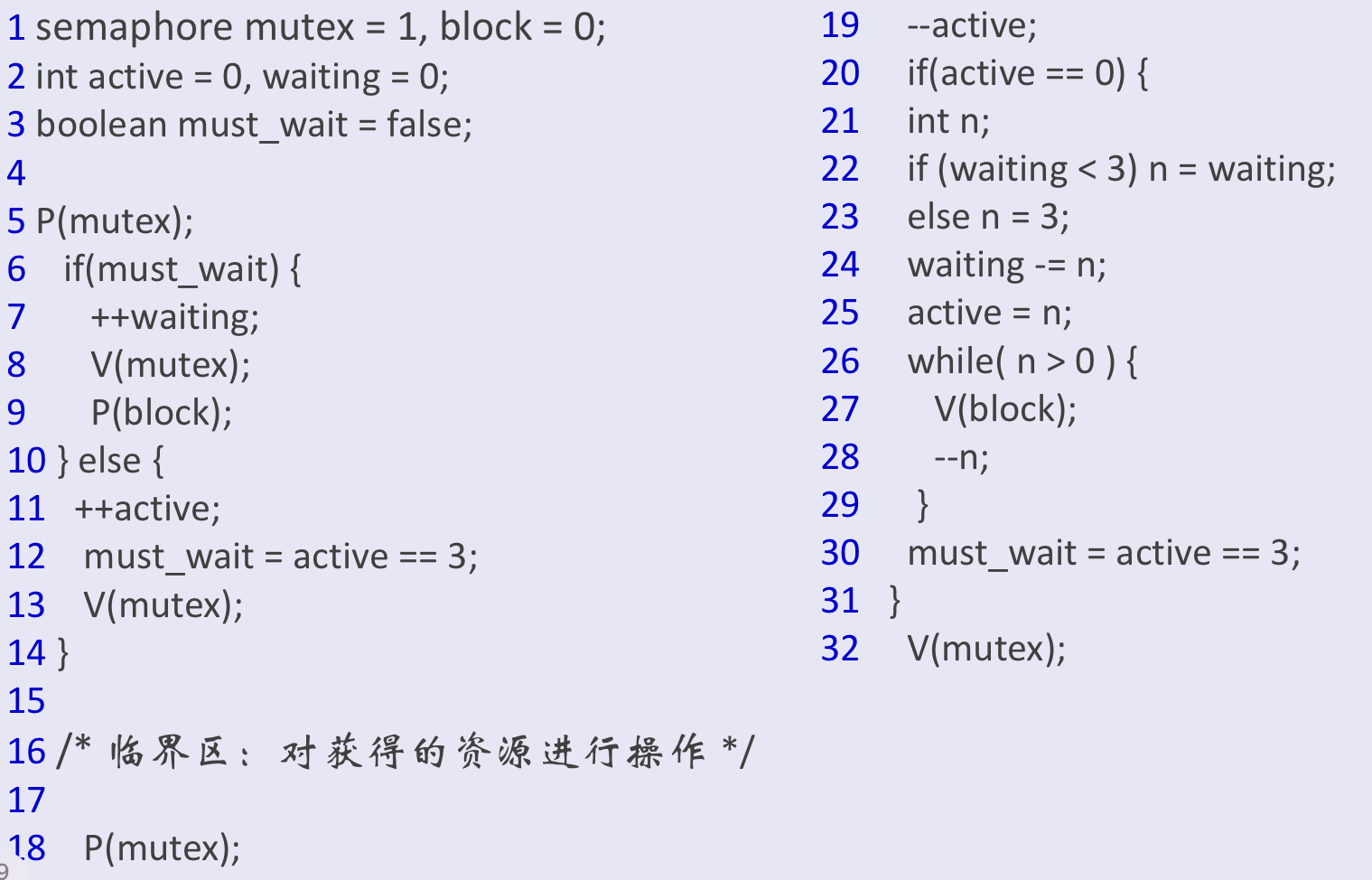
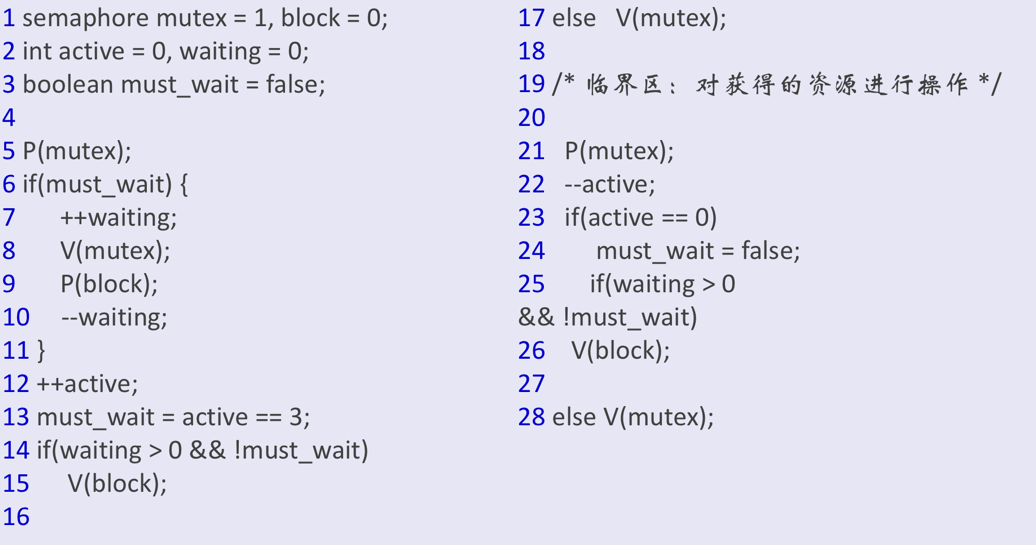
消息缓冲区为k个，有m个发送进程，n个接收进程，每个接收进程对发送来的消息都必须取一次。试用P、V操作解决发送进程和接受进程之间的正确通信问题。

1. 利用信号量解决资源管理问题

考虑具有如下特征的共享资源：当使用该资源的进程小于3个时，新申请资源的进程可以立刻获得资源；当三个资源都被占用后，只有当前使用资源的三个进程都释放资源后，其他申请资源的进程才能够获得资源。

1. 由于需要使用计数器来记录有多少进程正在使用资源和等待资源，而这些计数器自身也需要互斥执行修改动作的共享资源，所以可以采用如下的程序：

这个程序看起来没有问题：所有对共享数据的访问均被临界区所保护，进程在临界区中执行时不会自己阻塞，新进程在有三个资源使用者存在时不能使用共享资源，最后一个离开的使用者会唤醒最多3个等待着的进程

1. 这个程序仍不正确，解释其出错的位置；
2. 假如将第六行的if语句更换为while语句，是否解决了上面的问题？有什么难点仍然存在？
3. 现在考虑第一问的正确解法。解释下面这个程序的工作方式，为什么这种工作方式是正确的？
4. 这个程序不能完全避免新到达的进程插到已有等待进程前得到资源，但是至少使这种问题的发生减少了。给出一个例子
5. 这个程序是一个使用信号量实现并发问题的通用解法样例，这种解法被称作“I'll Do it for You”（由释放者为申请者修改计数器）模式。解释这种模式。
6. 现在考虑上一问的另一个正确解法。解释这个程序的工作方式，为什么这种工作方式是正确的？
7. 这个方法在可以同时唤醒进程个数上是否和上一题的解法有所不同？为什么？
8. 这个程序是一个使用信号量实现并发问题的通用解法样例，这种解法被称作“Pass the Baton”（接力棒传递）模式。解释这种模式。