2020 年 12 月 16 日

重庆大学<<操作系统>>期末考试

押题卷

一. 简答

1. API 和系统调用的关系

系统调用与 API 的关系

API（应用编程接口）是一些预先定义的函数，目的是提供应用程序与开发人员基于某软件或硬件得以访问一组例程的能力，而又**无需访问源码**。

API 函数通常为应用程序员调用实际的系统调用。(API 是对系统调用的再封装)

API 使得程序的可移植性好；实际系统调用比 API 更为注重细节且更加难用。

系统调用与 API 的区别与联系(简答题)

区别：**API 是函数的定义，规定了这个函数的功能，跟内核无直接关系。而系统调用是通过中断向内核发请求，实现内核提供的某些服务。**

联系：**一个 API 可能会需要一个或多个系统调用来完成特定功能。通俗点说就是如果这个 API 需要跟内核打交道就需要系统调用，否则不需要。**

程序员调用的是 API（API 函数），然后通过与系统调用共同完成函数的功能。

因此，**API 是一个提供给应用程序的接口，一组函数，是与程序员进行直接交互的**。

系统调用则不与程序员进行交互的，它根据 API 函数，通过一个软中断机制向内核提交请求，以获取内核服务的接口。

并不是所有的 API 函数都一一对应一个系统调用，有时，一个 API 函数会需要几个系统调用来共同完成函数的功能，甚至还有一些 API 函数不需要调用相应的系统调用（因此它所完成的不是内核提供的服务）

2. 进程和程序的区别和联系:

1.进程是一个动态的实体，有生命期；程序只是一个静态的实体，只是一组指令的集合
2.进程具有并发性；程序不能并发执行
3.进程具有独立性；没建立进程的程序不能运行

3.进程和线程的区别和联系:

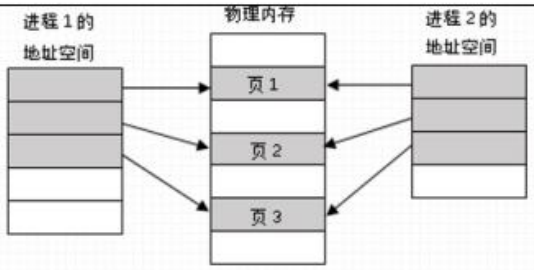
1.线程是调度和分派的基本单位；进程是资源分配的基本单位；同一进程的线程切换不会引起进程切换
2.进程之间可以并发执行，一个进程的多个线程之间也可以并发执行，因此可以更有效地利用系统资源，并发性更好
3.进程是拥有资源的独立单位，线程只拥有少量的资源，但可以访问其进程的资源；因此，线程创建，终止，切换 的系统开销比进程小
4.同一进程的线程共享内存和文件，在同一地址空间里，进程之间的通信无需调用内核，提高了通信效率

4.描述 写时复制

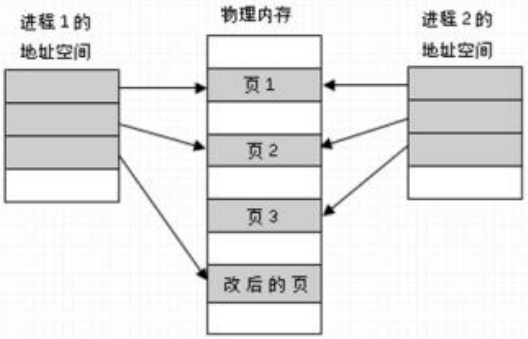
这都是针对进程而言的
父/子　页面共享
父/子 any 写入时，创建共享页面的副本；未被修改的页面仍然共享
只有可以修改的页面才标记为 写时复制，不能修改的页面可以由父子共享

写时复制/写时拷贝

父进程创建子进程时，最初父子进程共享内存空间。等到子进程修改数据时才真正分配内存空间，这是对程序性能的优化，可以延迟甚至是避免内存拷贝，当然目的就是避免不必要的内存拷贝。



a) 进程 1 发生页面修改之前



b) 进程1发生页面修改之后

5. 死锁条件

死锁的条件

死锁的发生必须具体以下四个条件：

(1) 互斥条件。
　　指进程的共享资源必须保持使用的互斥性，即任何一个时刻只能分配给一个进程使用，互斥条件是形成死锁最根本的原因，因为如果资源不要求排它性地使用，那么一定不会造成请求资源而无法满足的局面。

(2) 占有且等待条件。
　　一个进程占有了某些资源之后又要申请新的资源而得不到满足时，处于等待资源的状态，且不释放已经占用的资源。

(3) 不可剥夺条件。
　　任何进程不能抢夺另一个进程所占用的资源，即已经被占用的资源只能由占用进程自己来释放。

(4) 环路条件。
　　存在一组进程$P_1,P_2,\cdots,P_n$，其中每个进程分别等待另一个进程所占用的资源，形成环路等待条件。

6. 分页 / 分段 对比

	分页	分段
管理思想	页是信息的物理单位，是为了管理主存的方便而划分的，页的大小固定不变，实现了程序的非连续存放	段是信息的逻辑单位，它是根据用户的需要划分的，段的大小是不固定的，它由其完成的功能决定，因此段对用户是可见的；
虚地址	页式向用户提供的是一维地址空间，其页号和页内偏移是机器硬件的功能。	段式向用户提供的是二维地址空间
共享与存储访问控制	可以实现页面共享，但使用受到诸多限制，访问控制困难	便于共享逻辑完整的信息，易于实现存取访问权限控制
动态链接	不支持动态链接	支持动态链接

分页有内部碎片，不便于内存共享，难以访问控制

分段没有内部碎片，有外部碎片．便于内存共享，便于访问控制

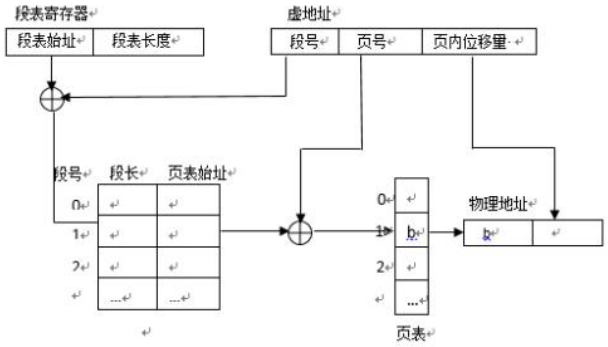
7. 段页式管理的原理/基本思想

》7.6段页式存储管理

段页式管理的基本思想：

一个进程有一个自己的二维地址空间。一个进程中所包含的独立逻辑功能的程序或数据仍被划分为段，并有各自的段号S，对于S中的程序或数据，则按照一定的大小将其划分为不同的页。

段页式管理系统的进程虚拟地址由三部分组成：
段号s、页号p和页内位移量d。
程序页可见的仍是段号s和段内位移量w。
p和d是由地址变换机构把w的高几位解释成页号p，以及把剩下的低位解释为页内地址d而得到的。



每个进程 1 个段表，每个段 1 个页表

也就是，1 个进程，1 个段表，多个段，

每个段，1 个页表，多个页

三次内存访问：

访问段表 得到页表 addr

访问页表 得到物理 addr

访问内存 得到 data

有二者的优点，但是：

复杂，开销增加；

执行速度下降

8. 叙述 请求分页存储机制

请求分页储存管理

缺页中断机构

地址变换

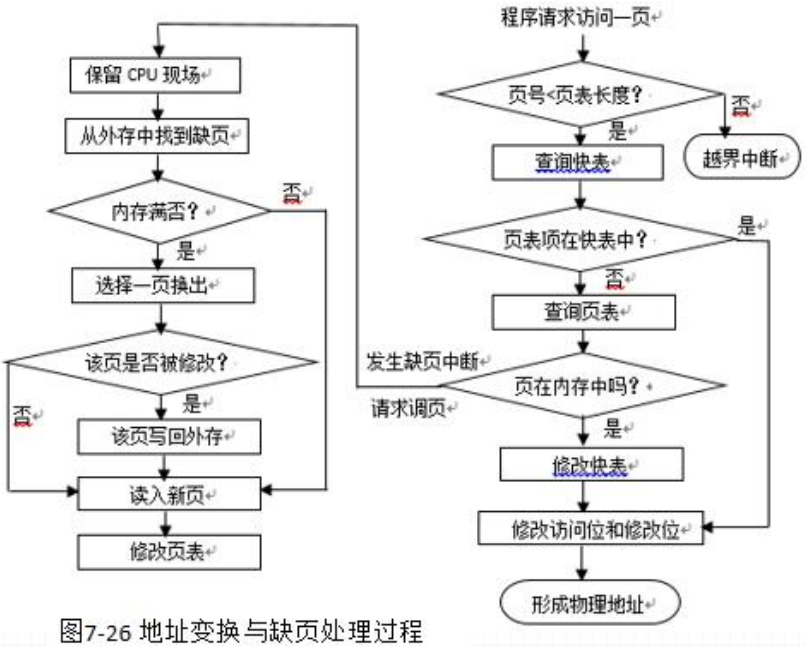
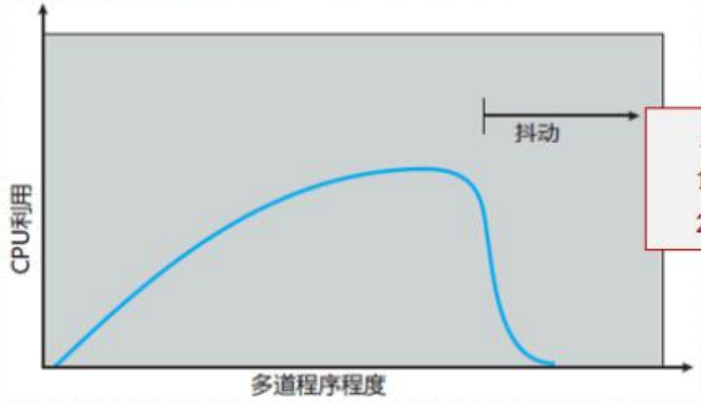


图7-26 地址变换与缺页处理过程

9. 系统抖动及解决办法

系统抖动

随着进程的增加，CPU的利用率也会增加，但是如果同一时间进程过多，每个进程占用的帧就相应变少了，就可能出现进程执行时需要经常性地发生缺页中断、CPU利用率又降低了的现象，而这时，操作系统还以为是进程数量太少导致的，还继续加入进程，导致每个进程占用的帧更少、CPU利用率更低的恶性循环，这种现象称为或系统抖动



系统抖动的原因：
1）分配的页帧数量太小
2）置换算法选择不当

解决系统抖动的方法

1）工作集策略

工作集合策略是通过计算每一个进程的工作集近似得到进程需要的页帧数，如果这个总数大于内存的物理帧数，则说明系统颠簸了，需要减少进程。

2）缺页率策略

缺页频率是另一种更为直接的防止抖动的方法。因此系统如果随时能够检测到系统中的缺页错误的情况，就可以动态地调整为进程分配的页帧数目。

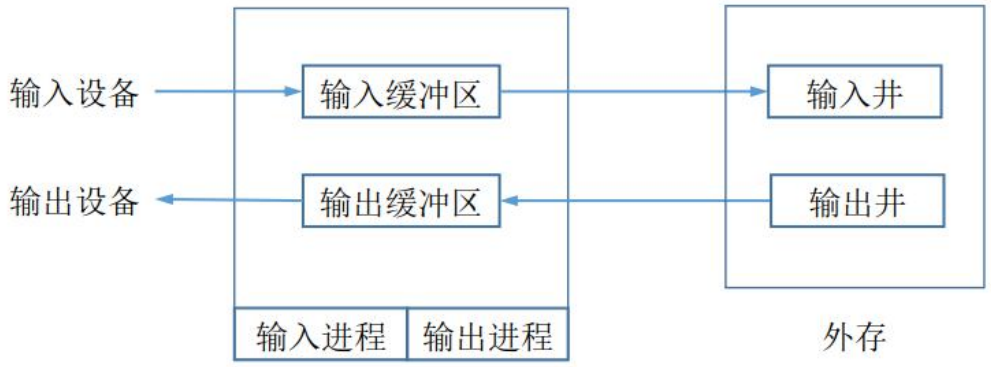
我们可以设置所需缺页率的上下限。如果实际缺页率超过上限，则可为进程再分配更多的页帧；如果实际缺页率低于下限，则可从进程中移走页帧。因此，可以靠直接监测缺页率来防止抖动。

实际上，抖动及其导致的交换对性能的负面影响很大。目前处理这一问题的最佳实践是，在可能的情况下提供足够物理内存以避免抖动和交换。

10. SPOOLing

- SPOOLING系统
  - Spooling是外部设备同时联机操作，又称为假脱机输入/输出操作，是操作系统中采用的一项将独占设备改造成共享设备的技术
  - Spooling系统是对脱机输入/输出工作的模拟，它必须有高速大容量且可随机存取的外存(如磁盘，磁鼓等)支持
  - 假脱机系统的组成
    - 输入井和输出井
    - 输入缓冲区和输出缓冲区
    - 输入进程和输出进程

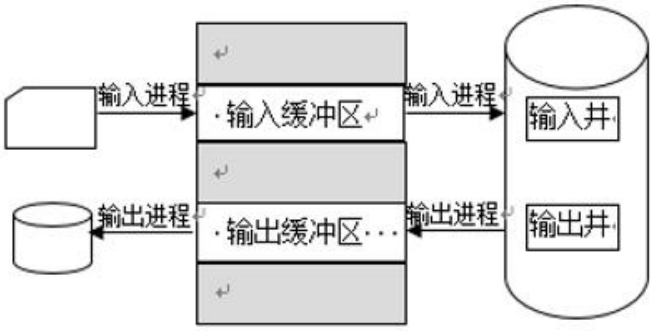
- Spooling系统的例子



虚拟设备的实现

通过虚拟技术将一台独占设备虚拟成多台逻辑设备，供多个用户进程同时使用，通常把这种经过虚拟的设备称为虚拟设备

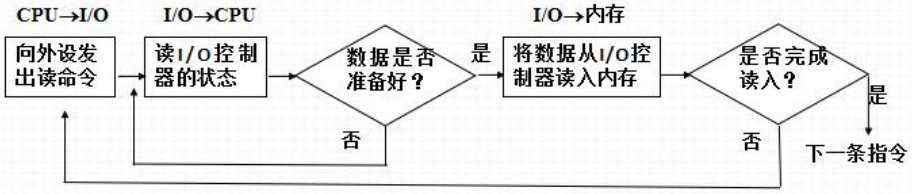
SPOOLing系统是Simultaneous Peripheral Operation On-Line 的缩写，通常称为“假脱机技术”。SPOOLing技术是对脱机输入、输出系统的模拟。



11. 轮询/中断/DMA

轮询方式

轮询方式也称为程序直接控制方式，该方式采用用户程序直接控制主机与外部设备之间进行输入/输出操作。CPU必须不停地循环测试I/O设备的状态端口，当发现设备处于准备好(Ready)状态时，CPU就可以与I/O设备进行数据存取操作。



中断控制方式

中断控制方式的基本思想是：引入中断处理机构，将轮询方式中的CPU最大限度地解放出来，使得CPU向I/O控制器发出I/O命令后能够调度其他进程执行，无需空转轮询，提升CPU与外设之间的并发执行能力。

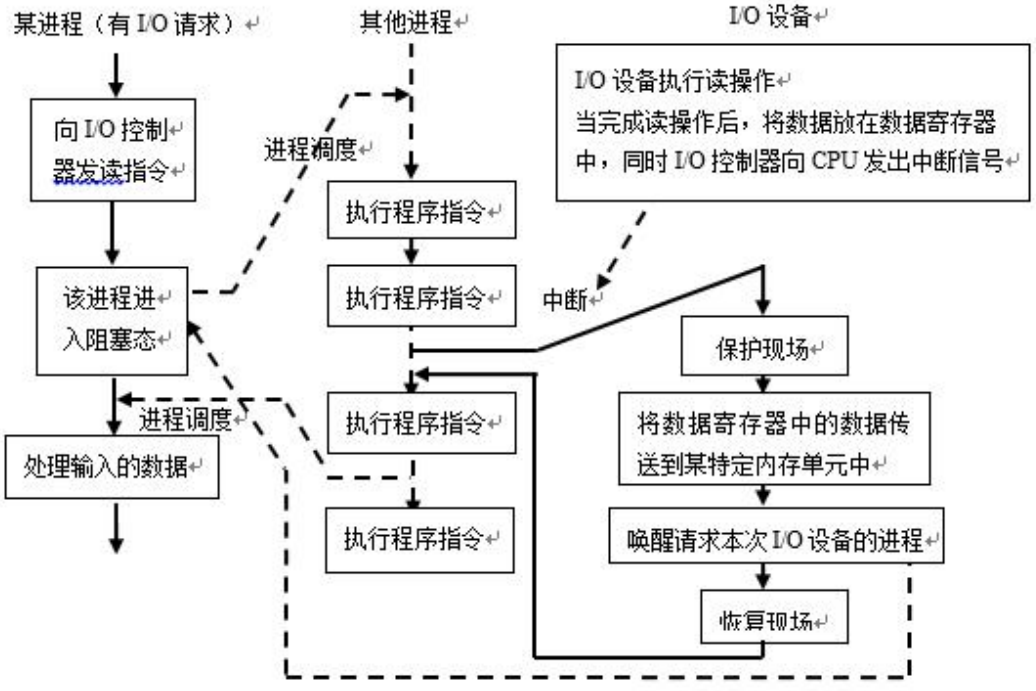


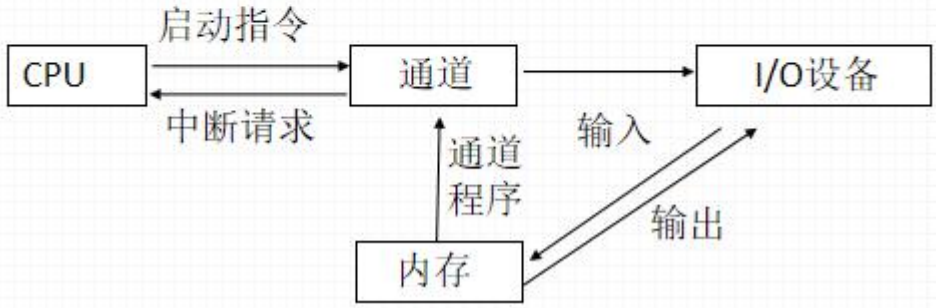
图 9-4 中断控制方式

DMA方式

DMA的英文拼写是“Direct Memory Access”，汉语的意思就是直接内存访问，是一种不经过CPU而直接从内存存取数据的数据传输模式。中断模式下硬盘和内存之间的数据传输是由CPU来控制的；而在DMA模式下，CPU只须向DMA控制器下达指令，让DMA控制器来处理数据的传送，数据传送完毕再把信息反馈给CPU，这样就很大程度上减轻了CPU资源占有率

通道方式

通道本质上是一个简单的处理器，专门负责输入、输出控制，具有执行I/O指令的能力，并通过执行通道I/O程序来控制I/O操作。



12. 缓冲区的作用

缓冲区的引入

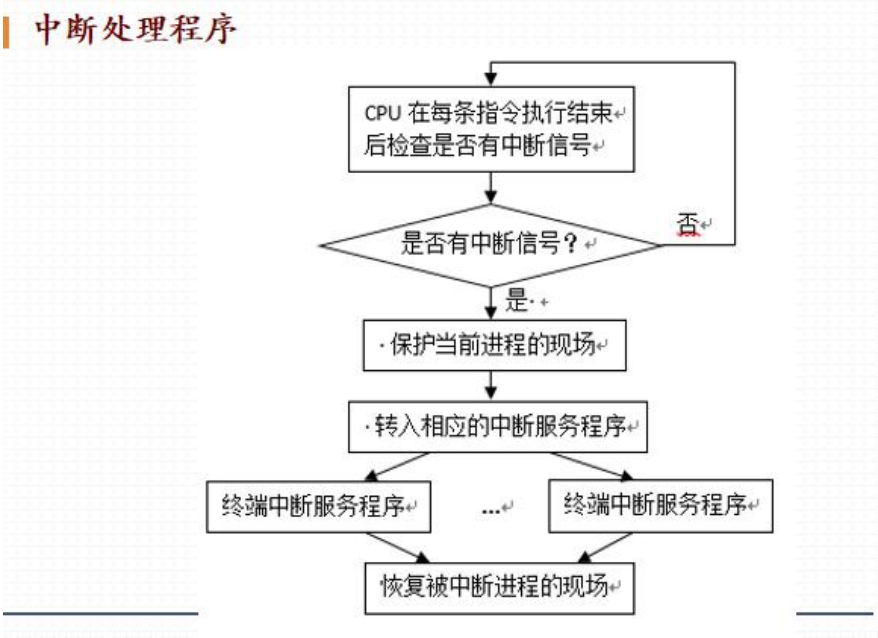
(1) 缓解设备之间速度差异的矛盾。

(2) 缓冲区可以缓解设备之间传输数据大小不一致的矛盾

(3) 支持应用程序I/O的复制语义

13. 中断处理程序

中断处理程序



14. 设备独立性

设备独立性的软件

什么是设备独立性?

“独立”两字体现在于：应用程序独立于具体使用的物理设备

为了实现设备独立性而引入了逻辑设备和物理设备这两个概念

逻辑设备名	设备类型	物理设备名	是否分配	驱动程序入口地址
/dev/tty	终端	3	是	1024
/dev/printer	打印机	5	否	2046
…	…	…	…	…

好处: 用户和物理的外围设备无关, 系统的外围设备变化时, 程序不用修改

易于对付输入输出设备的故障

15.中断控制方式 和 DMA 方式进行内存与外设之间数据传输的基本原理？对比优缺点

1、DMA方式和中断控制方式的主要区别

①、中断控制方式在**每个数据传送完成之后中断CPU**，而DMA控制方式则在所要求的**传送的一批数据传送结束时中断CPU**。

②、中断控制方式中的**数据传送在中断处理时由CPU控制**完成，而DMA控制方式则在**DMA控制器下**完成。不过，在DMA的控制方式中,**数据的传送方向，存放数据的内存始址及传送数据的长度等信息仍然由CPU控制。**

③、中断控制方式**以CPU为核心**，而DMA方式**以存储器为核心**，因此DMA方式可与CPU并行工作。

④、中断控制方式**传输的数据以字节为单位**，而DMA方式**传送批量数据，其基本单位为数据块**(通常一个数据块包含若干字节)。

2、DMA方式与通道方式的主要区别

①与DMA控制方式相加,**通道控制方式所需的CPU干预更少，并且一个通道可以控制多台设备**，进一步减轻了CPU的负担。

②**对通道来说,可以使用一些指令来灵活改变通道程序**，这一点DMA控制方式却无法做到。

③DMA方式需要CPU来控制传输数据块的大小、传输的内存位置，而**通道方式中这些信息是由通道控制**的。

DMA 提高了 CPU 的利用率，降低了其资源占有率; IO 设备和存储器直接进行成批数据的快速传输，效果好；中断会占用 CPU 大量时间.
DMA 主要用于内存中连续存放的数据的 IO 传输，不适合于不连续的数据块的传输，因为这需要多次 DMA 过程

什么是虚拟设备? SPOOLing 系统的功能和原理?
通过虚拟技术将一台独占设备虚拟成多台逻辑设备，供多个用户进程同时使用， 通常把这种经过虚拟的设备称为虚拟设备

16、什么是 spooling 系统？说明 spooling 系统的构成。SPOOLing 系统的主要功能是什么？

在多道程序环境下，利用一道程序来模拟脱机输入时的外围控制机的功能，把低速 I/O 设备上的数据传送到高速磁盘上；再利用另一道程序来模拟脱机输出时外围控制机的功能，把数据从磁盘传送到低速输出设备上。这种在联机情况下实现的同时外围操作称为 Spooling(Simultaneous Periheral Operations On-Line)

Spooling 系统的组成：

1、输入井和输出井。

2、输入缓冲区和输出缓冲区

3、输入进程 spi 和输出进程 spo

4、请求 I/O 队列。

SPOOLing 系统的主要功能是：将独占设备改造为共享设备，实现了虚拟设备功能。

(1)【◆题库问题◆】：[问答题] SPOOLing的含义是什么？试述SPOOLing系统的特点、功能以及控制过程。

【◆参考答案◆】：

SPOOLing它是关于慢速字符设备如何与计算机主机交换信息的一种技术，通常称为"假脱机技术"。

SPOOLing技术是在通道技术和多道程序设计基础上产生的，它由主机和相应的通道共同承担作业的输入输出工作，利用磁盘作为后援存储器，实现外围设备同时联机操作。

SPOOLing系统由专门负责I/O的常驻内存的进程以及输入井、输出井组成；它将独占设备改造为共享设备，实现了虚拟设备功能。

二. 大题

1. 信号量应用到 同步/互斥问题

解题步骤:

while 循环里面:

1. 声明信号量 semaphore s; //注释
2. 初始化 s.value = 1; //资源总数
3. 进入临界区时 ，调用 wait(s)
4. 出 临界区时， 调用 signal(s)

应用

- 会使用信号量机制来解决给定问题
- 生产者/消费者问题

进程互斥: 只涉及 1 个 semaphore

进程同步: 涉及 2 个及以上 semaphore

empty / full

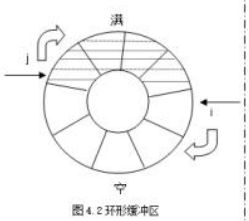
```
Semaphore Bufempty,Buffull;
Bufempty.value=n;Buffull.value=0;

void A( )                          void B( )
{                                  {
    wait(Bufempty);                    wait(Buffull);
     /*按照 FIFO 方式选择一个          /*按照 FIFO 选择一个装满
     空闲缓冲区*/                      数据的缓冲区*/
    save(data);                        retrieve(data);
    signal(Buffull);                   signal(Bufempty);
}                                  }

void main()
{
   parbegin(A,B);
}
```

1. 生产者-消费者问题[限定生产者消费者不能同时访问，也就隐含了互斥]

P. S 互斥一定写在内层




```
semaphore mutex,   mutext  .value =1;
semaphore empty,   empty  .value =n;
semaphore full;          full.value=0;
int i,j;
ITEM Buffer[n];
ITEM data_p,data_c;
void producer()
{
while(true)
    {
      Producer an item in data_p;
      wait(empty);
      wait(mutex);
      Buffer[i]=data_p;
      i=(i+1)%n;
      signal(mutex);
      signal(full);
    }
}
```

```
void consumer()
{
   while(true)
    {
      wait(full);
      wait(mutex);
      data_c=buffer[j];
      j=(j+1)%n;
      signal(mutex);
      signal(empty);
      consume theitem in data_c);
    }
}
```

2. 读者-写者问题

①多个读者同时读

②only 一个写者进程写入

③读写不能同时

```
int readcount;      readcount=0;
semaphore mutex_count, mutex_write;
mutex_count.value=1;    mutex_write.value=1;
```

```
void reader( )
{
   while(true)
   {
      wait(mutex_count);
      readcount++;
      if(readcount==1)
          wait(mutex_write);
      signal(mutex_count);
      Read file;
      wait(mutex_count);
      readcount--;
      if(readcount==0)
          signal(mutex_write);
      signal(mutex_count);
   }
}
```

```
void writer()
{
    while(true)
    {
        wait(mutex_write);
        Write file;
        signal(mutex_write);
    }
}
```

2. 银行家算法

死锁避免

资源分配拒绝策略, aka 银行家算法

Max

Allocation

Resource

Available

Need

安全序列

- 银行家算法
  - 安全序列
  - 安全状态
  - 算法
    - 当前是否为安全状态？
    - 满足资源请求后是否为安全状态？
  - 应用

3. 哲学家进餐问题解决

3 way to solve:

- 方法一：至多只允许四位哲学家同时去拿左筷子，最终能保证至少有一位哲学家能进餐，并在用完后释放两只筷子供他人使用。
- 方法二：仅当哲学家的左右手筷子都拿起时才允许进餐。
- 方法三：规定奇数号哲学家先拿左筷子再拿右筷子，而偶数号哲学家相反。

```
semaphore chopstick[5]={1 , 1 , 1 , 1 , 1};
semaphore r=4;
void philosopher(int i)
{
   while(true)
   {
      think();
      wait(r);                          //请求进餐
      wait(chopstick[i]);               //请求左手边的筷子
      wait(chopstick[(i+1) mod 5]);     //请求右手边的筷子
      eat();
      signal(chopstick[(i+1 ) mod 5]);  //释放右手边的筷子
      signal(chopstick[i]);             //释放左手边的筷子
      signal(r);                        //释放信号量r
      think();
   }
}
```

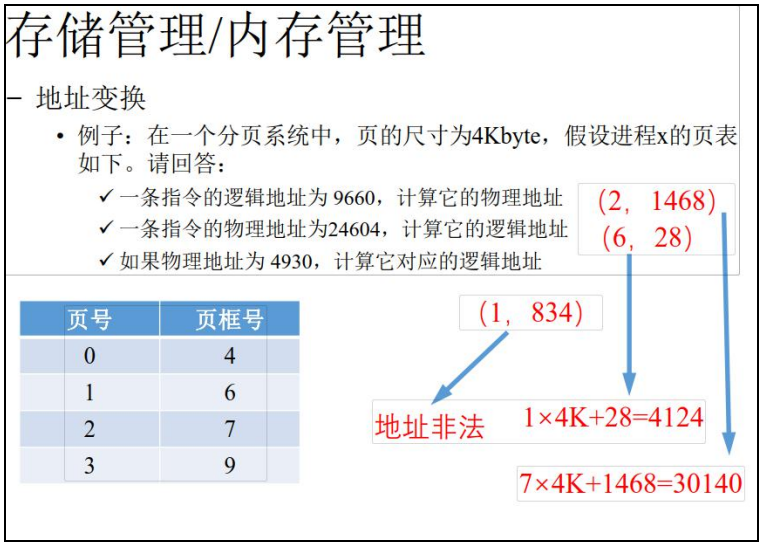


```
semaphore mutex = 1;
semaphore chopstick[5]={1 , 1 , 1 , 1 , 1};
void philosopher(int i)
{
   while(true)
      {
         think();
         wait(mutex);
         wait(chopstick[i]);
         wait(chopstick [(i+1) mod 5]);
         signal(mutex);
         eat();
         signal(chopstick [(i+1) mod 5]);
         signal(chopstick[i]);
         think();
      }
}
```



```
semaphore chopstick[5]={1 , 1 , 1 , 1 , 1};
void philosopher(int i)
{
   while(true)
      {
         if(i mod 2 == 0)                    //偶数哲学家，先右后左。
            {
               wait (chopstick [(i + 1) mod 5]);
               wait (chopstick [i]) ;
               eat();
               signal (chopstick [i]);
               signal (chopstick[(i+1) mod 5]);
            }
         Else                                //奇数哲学家，先左后右。
            {
               wait (chopstick [i]) ;
               wait (chopstick [(i+1) mod 5]);
               eat();
               signal (chopstick [(i+1) mod 5]);
               signal (chopstick [i]);
            }
      }
}
```


4. 地址变换



页号	页框号
0	4
1	6
2	7
3	9

5. 页面置换算法

6	5	1	2	5	3	5	4	2	3	5	3	2	1	2
6	6	6	2	2	2	2	4	4	4	5	5	5	5	5
	5	5	5	5	3	3	3	2	2	2	2	2	1	1
		1	1	1	1	5	5	5	3	3	3	3	3	2

FIFO 有belady

6	5	1	2	5	3	5	4	2	3	5	3	2	1	2
6	6	6	2	2	2	2	4	4	4	5	5	5	1	1
	5	5	5	5	5	5	5	5	3	3	3	3	3	3
		1	1	1	3	3	3	2	2	2	2	2	2	2

LRU 没有belady

6	5	1	2	5	3	5	4	2	3	5	3	2	1	2
6	6	6	2	2	2	2	2	2	2	2	2	2	2	2
	5	5	5	5	5	5	4	4	4	5	5	5	1	1
		1	1	1	3	3	3	3	3	3	3	**3**	**3**	3

OPT 没有belady

6. 磁盘结构+位示图

3．假设一个磁盘组共有100个柱面，每面有8个磁道，每个盘面被分成4个扇区。若逻辑记录的大小与扇区大小一致，柱面、磁道、扇区的编号均从“0”开始，现用字长为16位的200个字(第0字～第199字)组成位示图来指示磁盘空间的使用情况。请问：

(1)文件系统发现位示图中第15字第7位为0而准备分配给某一记录时，该记录会存放到磁盘的哪一块上?此块的物理位置(柱面号，磁头号和扇区号)如何?

(2)删除文件时要归还存储空间，第56柱面第6磁道第3扇区的块就变成了空闲块，此时，位示图中第几字第几位应由1改为0?

(1)块号＝15×字长+7＝15×16+7＝247;
柱面号＝［块号／每柱面扇区数］＝[247／(8×4)]＝7;
磁头号＝[(块号mod每柱面扇区数)／每盘面扇区数]＝［(247 mod 32)／4]＝5;
扇区号＝(块号mod每柱面扇区数)mod每盘面扇区数＝(247 mod 32) mod 4＝3;
所以该记录会存放在第247块上，即在第7个柱面，第5磁头,第3个扇区上。

(2)块号＝柱面号×每柱面扇区数＋磁头号×每盘面扇区数+扇区号＝56×(8×4)＋6×4+3＝1819 ;
字号＝[块号／字长]＝[1819／16]＝113;
位号＝块号mod字长＝1819 mod 16=11;
所以位示图中第113字第11位应由1变成 0 。

7. 磁盘调度算法 寻道计算