

1.

进程间同步：多个进程共同地完成一项任务，或者进程间存在一定的数据依赖关系。

进程互斥：进程之间会对同一份数据进行修改，为了保持数据的一致性，我们认为地使得这些进程不能同时对其进行修改，称这些进程为互斥进程。

可以发生在无关进程之间，不如说原本不相关的进程对同一份资源发出了调用请求，这是他们就是互斥的。

2.

没有

3.

软件：

优点：实现了互斥、空闲让进、有先等待的三个临界区实现的标准。

缺点：所有的进程等待均为忙等，并不释放CPU资源。多进程间的实现

较为复杂。基于load-store的机器指令可能并不能实现软件中的 原子级操作。

硬件：

优点：能够实现原子级的操作，简化了编程实现，提高了执行效率

缺点：忙等、饥饿、死锁，对多CPU处理机的实现复杂。

4.

Semaphore、Mutex（互斥体）、Monitor（管程）

5.

信号量机制：是和一组资源相关的一个实体，由一个整数和一个PCB表组成，可以表示剩余资源数量和等待资源的进程数量。

（如何做到元自己访？ ）

P原语：将资源量减一，如果当前资源量大于等于0，说明有资源分配给该进程，该进程进入临界区执行，否则，当前没有可用的资源，将当前进程阻塞并放入该资源的等待对列中。

V原语：将资源量加一，释放当前进程占用的资源，如果资源数小于等于0，说明有进程在等待该资源，从资源等待对列中挑选一个唤醒。

6.司机与售票员问题：

司机开车，售票员售票，售票员关门后司机才可以开车，司机停车售票员才可以打开车门。

共享（冲突）资源：开车（car）、开门(door)。

售票员的操作：关门🡪售票🡪开门

司机的操作：启动🡪开车🡪停车

通过分析，开关门，和启动停车均是互斥操作，我们应该设置close\_door的初始值为0，这样只有等售票员发出V原语后，驾驶员才会被唤醒，进入启动汽车；stop\_bus初始值应该设置为0，使得只有当驾驶员发出V原语后，售票员在会被唤醒，进行开门操作。

bus conductor:

do{

signal(close\_door)

//sell ticket

wait(stop\_bus)

//open door

}(true)

bus driver:

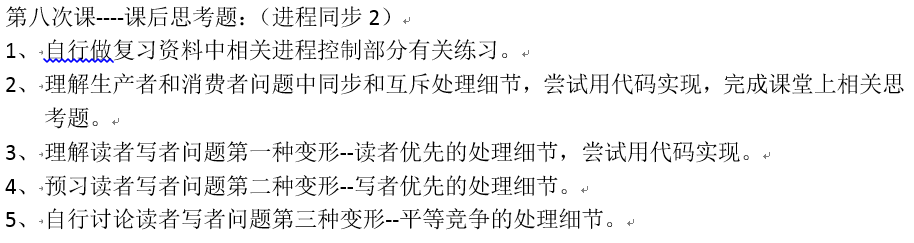
do{

wait(close\_door)

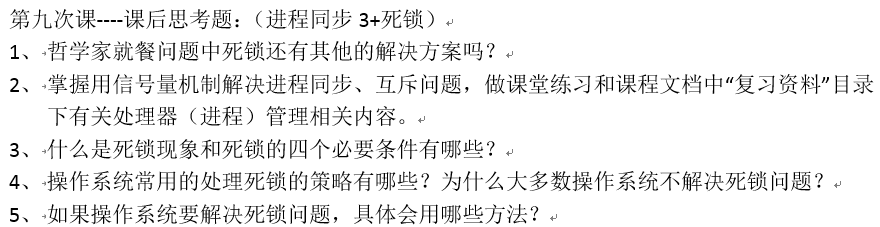
//drive

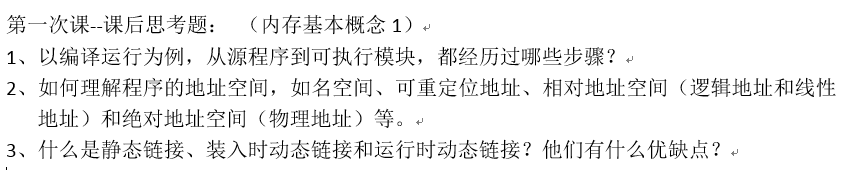
signal(stop\_bus)

}while(true)



3、4、5、





1. 源程序到可执行模块会经历以下步骤：
   1. 编译
   2. 链接
   3. 载入
2. 地址空间的理解：
   1. 名空间：是指在用户程序中的各个变量，对用户来说仅仅是一个表示变量的标示符，在编译后会转化为该变量的存储的逻辑地址。

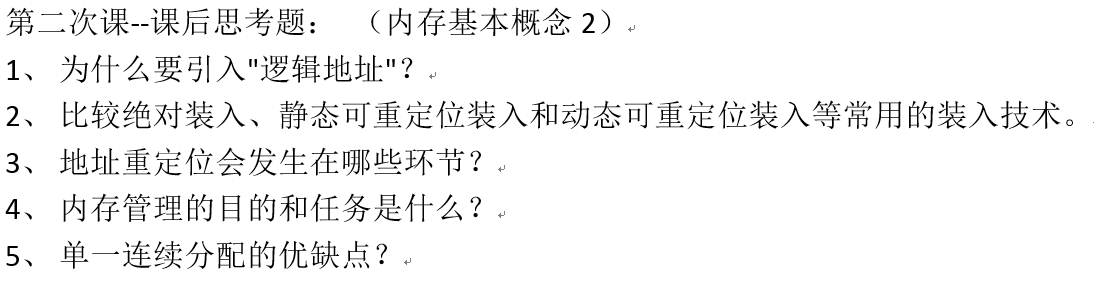
这是为了方便用户进行理解并使用。

* 1. 可重定位地址：指的是该地址在程序链接后可以重新更改其相对的逻辑地址。比如文件一和文件二均有逻辑地址为001的变量，在将他们连接后成为一个文件时，这两个变量的地址将会重新定位，一种可能的情况为文件一的变量的地址变为001，文件二的逻辑地址为002。

这是为了各个逻辑上独立的文件在进行连接的时候能够有正确的行为。

* 1. 相对地址空间：
     1. 逻辑地址：指的是用户在程序中的逻辑地址，均以0为基质进行编码。在装入的过程中会进行一定的地址映射。
     2. 线性地址：？？？？？
  2. 绝对地址空间：指的是某一变量物理上的存储地址，直接对应于真实的内存。

1. 什么是静态链接、装入时动态链接和运行时动态链接？他们的优缺点？
   1. 静态链接：程序执行时所需要的库文件在链接阶段就链入程序中，是完全装入，占用内存容量在三者之中最大。
   2. 装入时动态链接：程序装入内存时再将需要用到的库文件装入内存中，这样可以减少一些不必要的内存开销，一些实际上没有用到的部分不会装入内存。这样可以十分方便的对各个库文件进行修改，而不需要重复编译。
   3. 运行时动态链接：在程序执行的时候先检查环境中是否存在需要的文件的副本，若存在则直接使用该库文件，如果不存在，则再将该文件装入到内存中。这样可以保证内存中只有任何库文件的一份副本，极大的降低了内存开销。但是这样可能会导致在某些情况下程序执行的速度下降。



1. 为什么要引入逻辑地址？

逻辑地址是为了隔离用户和底层实现，对于用户来说仅仅需要给出各个变量之间的相对位置关系，并不需要考虑底层的具体实现，具体的地址映射将由操作系统来完成，这样能够保证用户不会非法访问内存。

1. 比较各个装入方式
   1. 绝对装入：程序再装入的时候只能装载到内存制定位置，不会发生变化。

这是用于单道程序执行环境。

* 1. 静态可重定位装入：程序可以装入到内存中的任何允许的位置，可以允许多个程序同时存在内存中。程序在运行的时候无法改变物理地址。
  2. 动态可重定位装入：和静态一样，但是将程序的物理地址和逻辑地址的映射延迟到程序真正执行的时候发生，故在程序运行的时候其物理地址可能会发生改变。需要硬件支持，实现较为复杂。不需要连续存放在内存中。

1. 地址重定位会发生在那些环节？

链接：各个模块进行连接时，会根据一定的关系将各个模块内相对独立的逻辑地址统一整理，形成一份新的逻辑地址。

执行：程序实际执行的时候，由于动态可重定位技术，也会发生地址重新的定位

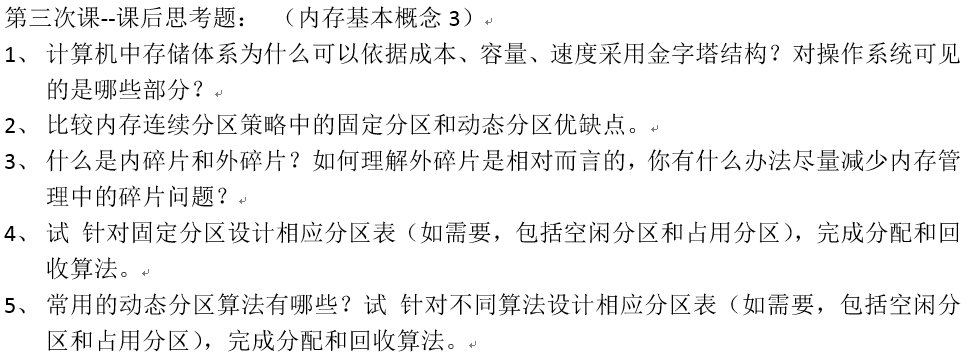
1. 内存管理的目的和任务？
   1. 目的：最大化利用内存空间，让一定大小的内存可以装入更多的程序。

安全、共享和通讯，提供最高性价比的解决方案。

* 1. 任务：为需要内存的程序分配内存，为终止或者挂起状态的程序回收内存空间。管理和维护可用的内存资源。

存储共享、保护、扩充。

1. 单一连续内存分配的优缺点？（一定是单任务操作系统吗？）
   1. 优点：逻辑简单，易于实现。
   2. 缺点：仅仅适用于单道批处理的环境，一次只会有一个程序在内存中，降低了内存的利用率。

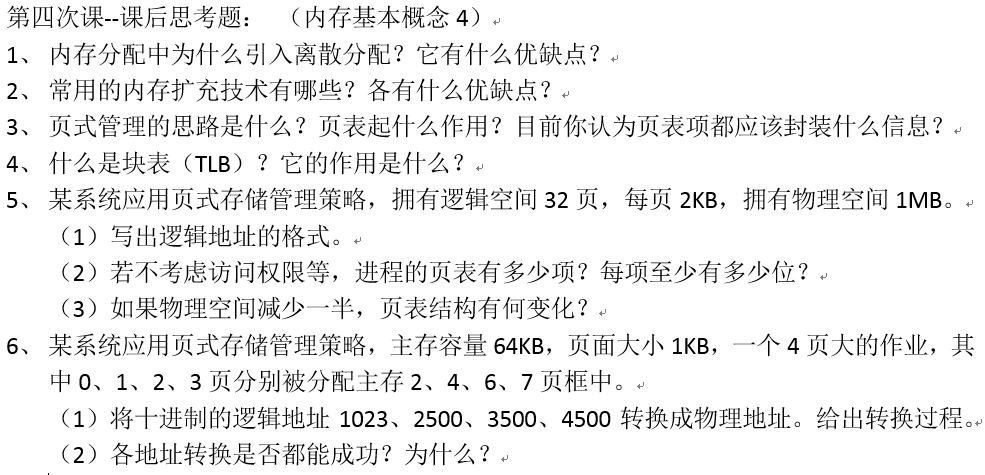


1. 金字塔结构？操作系统可见什么？

因为往往速度快的存储体容量低，成本高，为了实现最大的性价比，故产生了多集存储体系的金字塔结构

操作系统可见的部分为内存，辅存，剩下的为硬件维护。

1. 固定分区和动态分区的优缺点：
   1. 固定分区：每一个分区大小固定，不会产生外碎片，但是由于分区大小是实现确定的，每一份区并不能完全的被程序完全使用，会产生一定的内碎片。难以分区共享，即使有重复的部分也无法分享。分区数目有限，限制了总的进程数。
   2. 动态分区：不为内存实现定义分区，而是在程序装入的时候量体裁衣地为程序分配内存空间，分区的大小由各个程序的实际情况动态划分，这样可以提高内存资源的利用效率。但是在程序执行过程中，随着不断释放和分配，会产生一定的外碎片。
2. 内碎片，外碎片？怎么减少呢？
   1. 内碎片：在系统为程序分配的内存区域内部产生的没有被程序利用的空间。
   2. 外碎片：在系统为程序分配的内存区域的外部之间的空闲区，由于该区域不能完全的放下一个程序，虽然是空闲，但是实际上无法利用。
   3. 减少方法：
      1. 内碎片：使用好的分配算法，装入时为程序分配静可能合适大小的内存。
      2. 外碎片：使用好的分配算法，进行碎片整理。
3. 哈啊
4. 动态分区算法有哪些？
   1. Best fit：为程序分配最适合的空闲区，该空闲区刚好能放下该程序。
   2. Worst fit：为程序分配当前最大的一块空闲区。
   3. First fit：程序分配第一个查询到的能够放下该程序的空闲区。



1. 为什么离散分配？优缺点？

因为在连续分配技术中，随着程序的运行会产生大量的外碎片，降低了内存空间的利用率，浪费了内存资源。

优点：提高了内存利用率，没有外碎片。易于实现共享和进程间通信。

缺点：相比连续分配，系统需要维护更多的信息。

1. 内存扩充技术有哪些？优缺点？
   1. 覆盖技术cover：让不存在调用关系的模块不同时出现在内存中。一个程序的几个代码段或者数据段，按照时间先后顺序和大小进行分配，在逻辑上并行的区域不同时装入。

缺点：程序需要预先确定覆盖关系，在使用到了不在内存中的部分时需要临时掉入。

优点：提高内存的使用效率。

* 1. 交换技术swapping：将内存中处于等待状态的程序调出内存。

缺点：时间开销大

优点：用户不需要提供覆盖关系，增加了并发运行的程序数目？

1. 页式管理的思路？页表的作用？页表项要封装什么？
   1. 思路：用户程序和系统为内存空间进行划分，每一份成为一页，之后建立一定的对应关系，将用户的逻辑页对应到物理上的实际页面。
   2. 作用：将逻辑地址映射到真实的物理地址上。
   3. 物理地址的高位地址，也就是物理页号；该页的访问信息，一些保护相关的信息。
2. 什么是快表（TLB 相连存储器）？有啥作用呢？

TLB即相连存储器，输入key值后能够快速给出相应的value值。起到一个缓冲作用，提高地址映射的速度。

