

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:

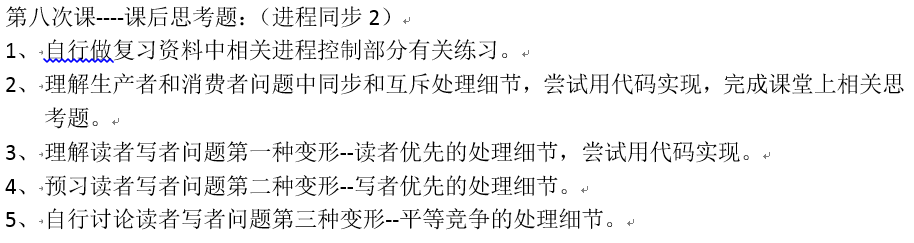
do{

wait(close\_door)

//drive

signal(stop\_bus)

}while(true)



3、4、5、