1、

（1）互斥性：只有当flag[j] == false或者turn == i时，进程Pi才进入临界区。如果两个进程同时在临界区执行，那么flag[0] == flag[1] == true， 但由于turn不可能同时等于0和1，所以必定有一个进程P0无需进入if（turn == 1），而另一个进程P1需要进入并执行flag[1] == false。这将会使P0跳出while（flag[1]）并进入临界区。而P1则因为turn == 0而进入等待状态，无法进入临界区。因此可得，P0、P1互斥

（2）有空让进：若P0跳出临界区，则必定会执行flag[0] == false，从而使进程P1跳出循环而进入临界区

（3）有限等待：由（2）可知，P1最多在P0进入临界区一次后就可以进入，因此P1的等待次数有限

2、如果用户级程序具有禁用中断的能力，那么它可以禁用计时器中断并防止发生上下文切换，从而不让其他进程执行，达到独占处理器的目的。

3、如果多个进程运行在不同的CPU上，每个进程都试图进入一个临界区，即使禁止中断，其他进程依旧能够在其他处理器上进入临界区。

4、 设置一个信号量connection == N，当connection <= N时说明已经有N个连接建立，此时根据进程同步原则无法再有更多的连接（进程）建立，实现了用信号量控制服务器的连接数。

5、wait()操作以原子的方式递减与信号量相关的值。如果一个信号量的值为1时，在该信号量上执行了两个等待操作，如果这两个操作不是原子执行的，那么这两个操作可能会继续减小信号量的值，从而违反互斥。

6、

Semaphore barberReady = 0

Semaphore accessSeat = 1

Semaphore num\_wait = 0

int seat\_free = n

customer()

{

while(1)

{

wait(accessSeat)；

if(seat\_free > 0)

{

seat\_free--;

signal(num\_wait);

signal(accessSeat);

wait(baberReady);

/\*理发\*/

}

else {

signal(accessSeat);

/\*离开\*/

}

}

}

barber()

{

while(1)

{

wait(num\_wait);

wait(accessSeat);

seat\_free++;

signal(baberReady);

signal(accessSeat);

/\*理发\*/

}

}