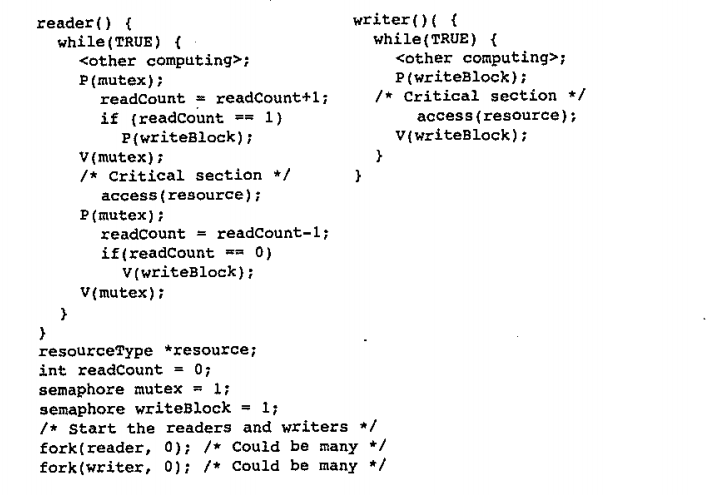
reader writer problem

规则1，如果write，没人能看包括其他writer，如果reader，那么其他reader也能看，直到reader=0之前，writer不能write

标答



while true是可有可无的，因为Pmutex本身就可以让你wait

reader(){

P(mutex);

readCounter++;

if （readcounter==1）

P(WB)

V(mutex)

<read>

P(mutex)

readcounter--

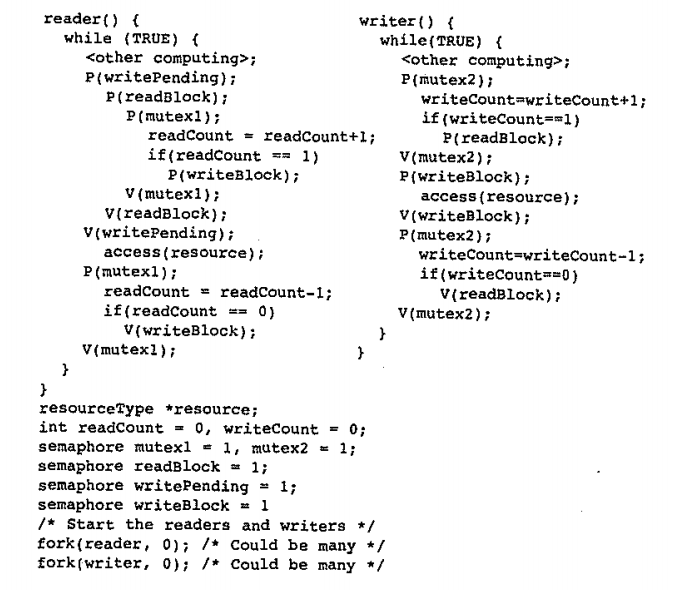
if (readCounter==0)

V(WB)

V(mutex)

重点：readCounter以及他的判定也是另一种的critical section，所以我们要用Mutex包上，大于0，不能write=0，才能write

规则2：如果有一个writer，不管在里还是在外，都应该不能再进入了reader了



writer解释：无论怎样，有wriiter进来就要count+1，

这时有几种情况，

1.没有writer没有reader，这时直接不让read（pRB）然后开始pWB写作，然后让pWC-1，注意这时别的writer也可以写了，如果没有WRITECOUNT，就可以开readBlock

2.有别的reader，这时P readBlock，告诉你不要再让别的reader进来了，接着同上

3.有reader有writer，就跳过这一步，readBlock永远在确认没writer的时候解开

writePending是为了锁住reader自己的，防止在判定过程中有reader加入判定。

active semaphore:V(S):[s=s+1;yield]

passive semaphore: V(S):[s=s+1]

V(S)这个操作可能需要90ms,然而需要100ms才能转换到ps2,yield（释放CPU）主动转换，避免了浪费10ms，产生了一个新问题：太多process相互转换

Process与active并没有具体好坏

Busy waiting

P(S):DI[while S==0;EI;DI;S=S-1]EI

会不停EI DI来check是否S已经不是0

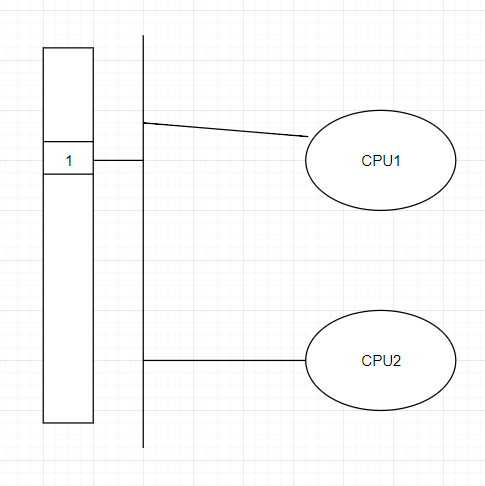
单核情况，PS1占用了CPU，PS2不停地1check，只有PS1 VS了，PS2成才可以运转

多和情况：不用管ps1,只要用空闲的核，PS2就可以直接进CPU

如果单核，busy waiting浪费时间

如果多核，busy waiting 有意义

多核问题：

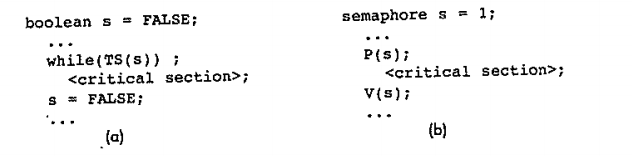


CPU1与CPU2读到的semaphore(存储与内存中)都是1，那么都work，从根本上来说，他们都同时对CS进行操作，根本上就有问题

面对多核问题，我们必须用hardware方法

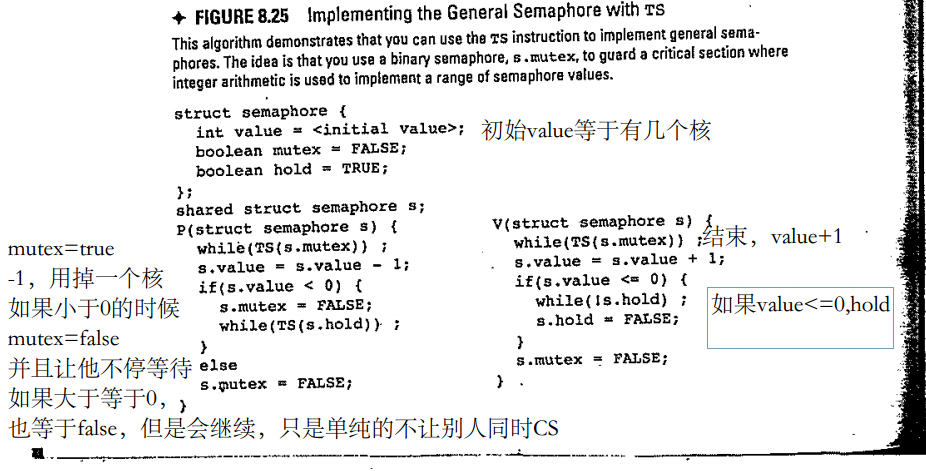
TS-TEST AND SET

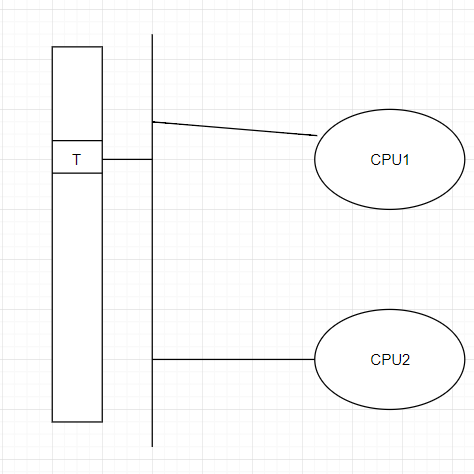
ts(s)，如果S在内存上是FALSE，就会return false并内存上改成true，不然直接return true



TS(S),这里的S是一个boolean值，也可以是0,1

用TS来构建一个semaphore





开始情况：CPU1在verify T/F，这一步占据了bus与内存，这样别人就不能用，

情况1，如果内存是T，这代表了不可以使用CS，cpu waiting,别人可以使用BUS,保持内存还是T

情况2，如果得到F，这代表可以使用CS,把memory设置成T，这样别人不能用BUS，完成后改成F

例如，一共四个核

4

P1 3

P2 2

P3 1

P4 0

P5 - WAIT

P6 - WAIT

P7 - WAIT

while(s.hold)意义，如果hold已经是false，wait，等到别人消耗他

P6，P7，P8都停留在P的while(TS(s.hold))里，这时一个V（s）成功运行，s.hold等于false，放跑一个