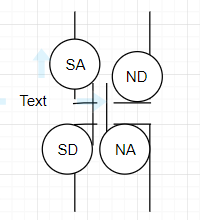
one way tunnel问题



int southCounter=0

int northCounter=0

Cond Busy

NA(){

If(SC>0)//南半隧道有车

BUSY.WAIT;

NC++;//不在if里，你激活就在路口

}

SA(){

if(NC>0)

BUSY.WAIT();

SC++;

}

SD(){

SC- -;

if(sc==0)

while(busy.queue>0)

busy.signal;

}

ND(){

nc--;

if(NC==0)//只有所有NORTH车走完，开始检查有没有堵着的S车

while(busy.queue>0)

busy.signal;

}

逻辑就是如果另一边有车，wait， 没车，COUNTER++,

离开的时候counter--,如果这边的车都跑完了，激活另外一边

逻辑问题：

假设有两个N车，三个S车

假设NA().SA()，这时N会进去，S会busy.wait()

两个N走完，busy.wait()开始走，

这时一个N车又进来，因为SC>0,所以他也busy.wait()，无法区分，fail

解决方案：busy1.wait ,busy2.wait

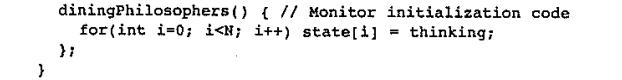
哲学家恰饭问题

int x;

enum states{eating,thinking,hungry}

status state[n];

Cond self[n];



pickUp（int c）{

state[c]=hungry;

test(c);//即使signal了，Method剩下的部分还是会继续

if (state[i]!=eating)

self[i].wait;

}

puddown(int c){

state[c]=thinking

test(c+1 mod N);

test(c-1) mod N;

}

test(int c){

if state[c-1]!=eating&&state[c+1]!==eating&&state[c]==hungry

{state[c]=eating;

self[c].signal;

}

}

改进：即使左筷没了，那你也应该拿右筷，不然你左右人疯狂交替，你就饿死

或者even可以拿1筷攒2筷，odd只能一次性拿2

monitor至今不存在，他只是一个idea

monitor可以决定哪里不能Interrupt，哪里能interrupt然而这很难做到。

主要原因是semaphore已经存在，而且有效

没必要花钱搞Monitor

其他了类型的高等synchronization

ENCS(int t)

{while(turn!=t)

yield;

}

LCS(INT T)

{turn =1-t

}

P0

ENCS(0)

CS

LCS(0)

P1

ENCS(1)

CS

LCS(1);

一开始给个turn值，决定先后顺序

ENCS与semaphore区别，ENCS是有turn的，是P0 P1 P0 P1 P0 P1

turn=1

P0

encs(0)

cs

lcs(0)

encs(0)

cs

lcs(0)

p1

ENCS(1)

CS

LCS(1)

ENCS(1)

CS

lCS(1)

那么顺序就是进行第一个CS1，进行CS0，进行第二个CS1，进行第二个CS0，完美

但是问题在于，如果有第三个CS0，那么实际上是没法进行的

所以即使没有Mutual exclusion，还是不好

判断一个synchronization好不好

1.deadlock()，一个进程必须等另外一个进程，然而另外一个进程也在等他，所以无法推进。ENCS没有deadlock，他在第二个CS0以后寻找第三个CS1，没有CS1，那么就直接exit了，第三个CS0没有wait，而是被忽略

2.mutual exclusion

3. Progress is guaranteed//process的实际运行是满足我们的预期需求的

这个encs的问题就是没有完成3

PS1

9.00am cs

8.00pm cs

PS2

9.01am cs

11.00am cs

2.00pm cs

如果按ENCS，会PS1，PS2这两步是合理的，接着8.00PM PS1，但实际上我们应该11.00am PS2，没必要过多wait

及解决方案

Boolean flag[2];

flag[0]=F Flag[1]=F;

ENCS(int t)

flag[t]=T;

while(Flag[other]==T)

yield;

LCS(int t)

flag(t)=F;

问题在于我们如果进行ENCS(1)这时FLAG1是T

我们又ENCS0，会while wait

再encs1， 也会while wait， deadlock

改进

ECS(0)

int other=1-t

flag[0]=true

turn=1;

while(turn=1&&flag[1]=T;

LCS(0)

Flag[0]=false

flag[1]=false

turn=0;

ECS(1)

int other

other=1-t

flag[1]=T;

turn=0;

while(turn==0)&&Flag[0]==T;

在pdf179页