

Solution 1

var turn=0/1(turn is a atomic instruction)

P1

Repeat

While turn!=0 do no_op

CS

Turn=1

Remainder section

P2

repeat

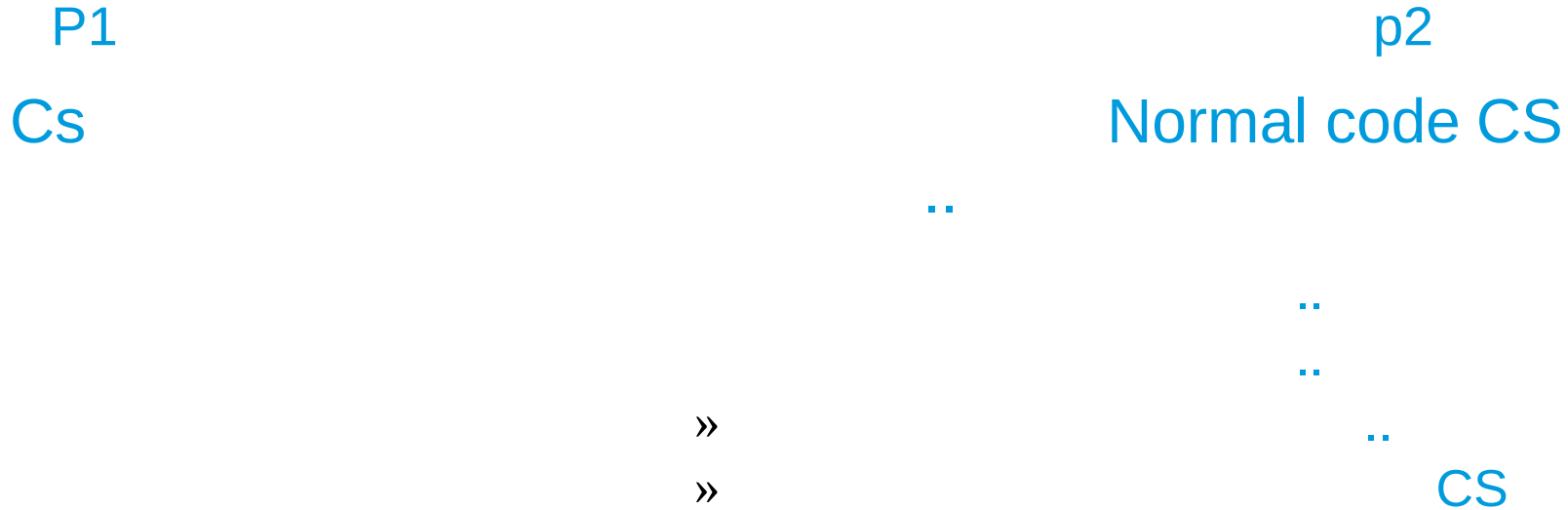
while turn!=1 do no_op

CS

turn=0

remainder section

Disadvantage?



It requires strict alteration , E.g: if turn=0 and process 2 is ready to enter the CS, process 2 can not do so even though process 1 may be in its remainder section.

Solution

replace turn with an array

Var array flag[0..1] of boolean

P0

flag[0]=1

While flag[1]==1 do
no_op

CS

Flag[0]=0

Remainder section

P1

flag[1]=1

while flag[0]==1 do no_op

CS

flag[1]=0

remainder section

Flag[0]=1 means that process 0 is ready to enter the CS. But flag[0]=0 means that it is no longer needed to be in its CS

Disadvantage

p0 sets flag[0]=1

P1 sets flag[1]=1

P0 and p1 will be looping forever in their while statement.

Algo-3

By combining the key idea of algo1 and 2 we got
a correct solution

P0

Flag[0]=1

Turn=1

While (flag[1]==1 and turn==1){

Do no_op }

CS

Flag[0]=0

Turn =1

p1

flag[1]=1

turn=0

while(flag[0]==1 and turn==0){

do no_op}

CS

flag[1]=0

turn=0

N processes

var

number:array[0...n-1] of integer

$(a,b) < (c,d)$ if $a < c$ or if $a = c$ and $b < d$

$\max(a_0, \dots, a_{n-1})$ is a number, k such that $k \geq a_i$ for $i = 0 \dots n-1$