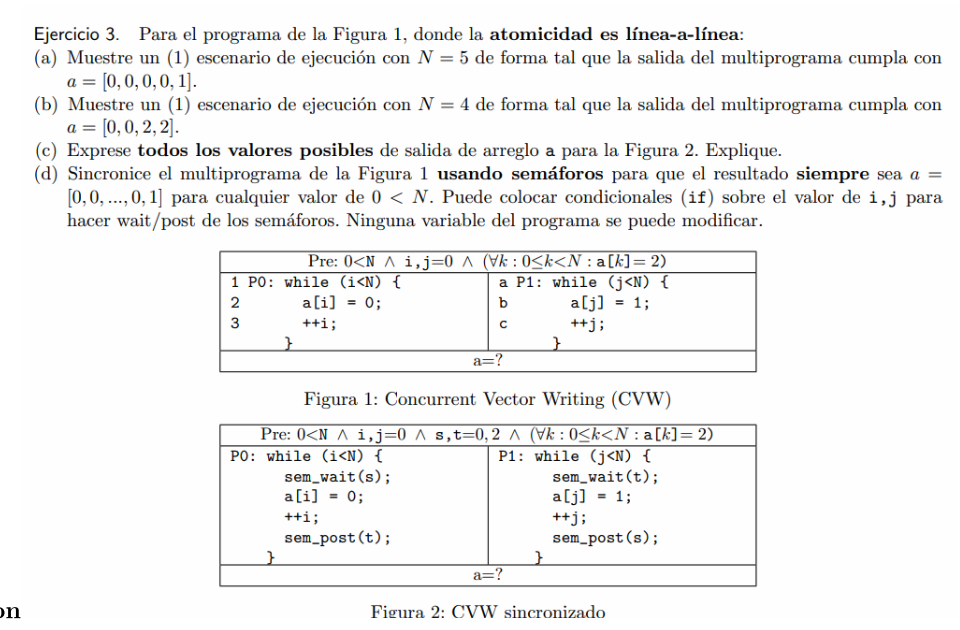
hola soy mane y tambien me ha tocado sufrir. espero que esto te ayude <3



c) [0,0,0,0,0]

[0,0,0,0,0]

segun scheduling policies: 2 caminos: se sigue ejecutando el semaforo t o se cambia al semaforo s

asumo que la policy es seguir con el proceso que ya estaba en ejecucion (puede producir starvation si esta mal programado)

| s =0; t=1; j,i = 0  p0 while (i<N){  semwait(s)  if (i != N-2) {  a[i]=0  i++  }  sempost(t)  } | p1 while(j<n){  semwait(t)  a[j]=1;  j++  sempost(s)  } |
| --- | --- |

[0,...0,1] N=5. [0,0,x,x,x] [0,4]

| 0 | 0 | 0 | 0 | 1 |
| --- | --- | --- | --- | --- |

| P0:   1. se ejecuta 2do, pisa el resultado anterior. ahora a[0]=0, i++ 2. se ejecuta 2do, pisa el resultado anterior. ahora a[1]=0, i++ 3. se ejecuta 2do, pisa el resultado anterior. ahora a[2]=0, i++ 4. se ejecuta 2do, pisa el resultado anterior. ahora a[3]=0, i++ 5. se ejecuta primero, a[4]=0, i++ | P1:   1. Se ejecuta primero. a[0]=1, j++ 2. Se ejecuta primero. a[1]=1, j++ 3. Se ejecuta primero. a[2]=1, j++ 4. Se ejecuta primero. a[3]=1, j++ 5. se ejecuta segundo, pisa el resultado anterior. ahora a[4]=1 |
| --- | --- |

1. wtf
   1. walter
   2. trae
   3. faso
2. Depende de la politica del scheduler.

como se inicializa en s=0,t=2:

| P0:  1er iteracion:  se hace sem\_wait(s)  como s=0 se bloquea el proceso  ahora s =1  AHORA AMBOS PROCESOS ESTAN  CUAL SE EJECUTA PRIMERO?  \*\*\* | P1:  1er iteracion:  se hace sem\_wait(t)= t-1 =2 -1 = 1  se ejecuta el codigo a[0]=1;j++;  ahora j=1  sem\_post(s)=s+1;  ahora t =1  DESBLOQUEADOS |
| --- | --- |

\*\*\* aca hay dos opciones:

>se vuelve a ejecutar p1 realizando su segunda iteracion para luego dejar paso a p0

>el scheduler frena a p1 y p0 es ejecutado

entonces la politica del scheduler dictaminara el resultado final.

1. {resuelvo suponiendo que el scheduler prioriza el ultimo proceso en ejecutarse}

j=0;i=0;s=0;t=2;

P0: while (i<N){

if(i==N-2){

s = 0;

}

sem\_Wait(s):

a[i]=0;

i++;

sem\_Post(t):

}

P1: while(j<N){

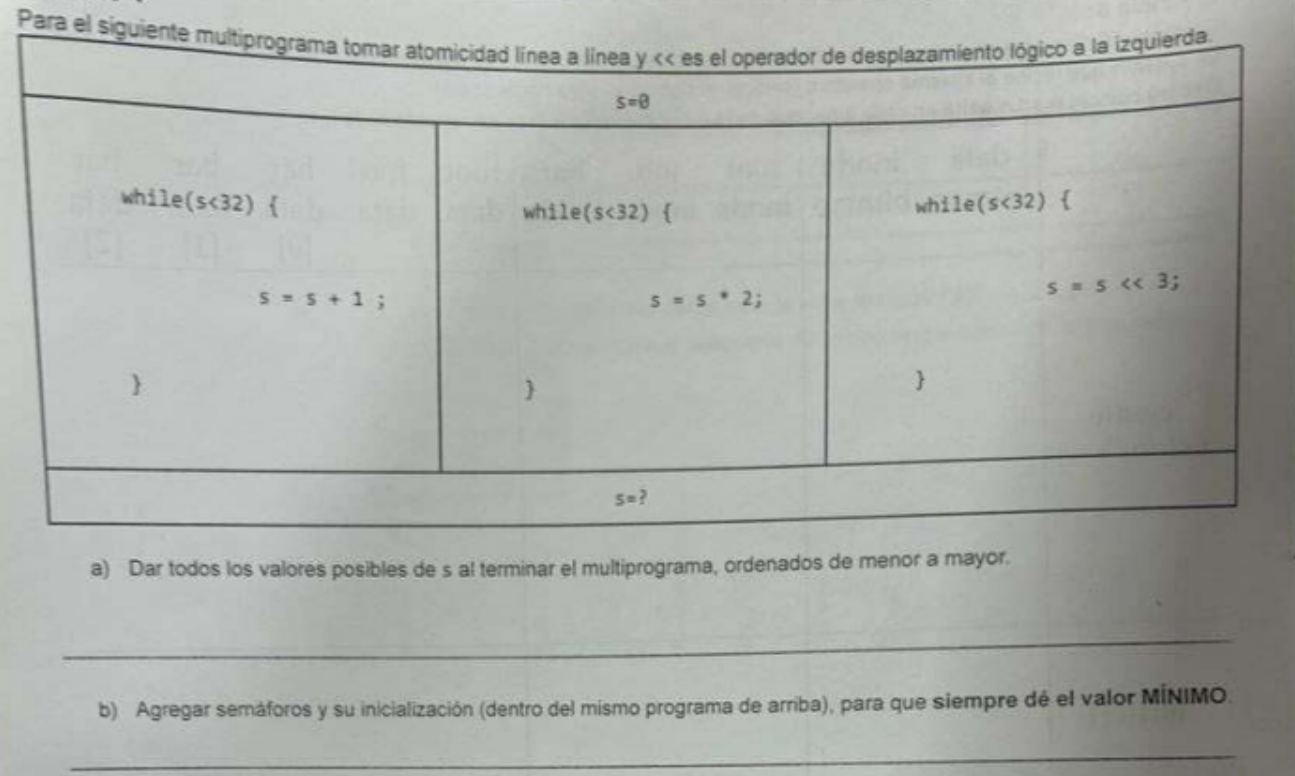
sem\_Wait(t):

a[j]=1;

j++;

sem\_Post(s):

}



s=[32,34, 36, 38, 40,42,44,46,48,50,52,54,56,58,60,62,64,72,80,88,96,104,112,120,128,136,144,152,160,168,176,184,192,200,208,216,224,232,240,248

32=100000

31 = 11111

30 =11110

29 =11101

28=11100

27 11011

26 11010

25 11001

24 11000

23 10111

22 10110

21 10101

20 10100

19 10011

18 10010

17 10001

16 10000

15 1111 xxxx

14 1110

13 1101

12 1100

11 1011

10 1010

9 1001

8 1000

7 0111

6 0110

5 0101

4 0100 xxxx

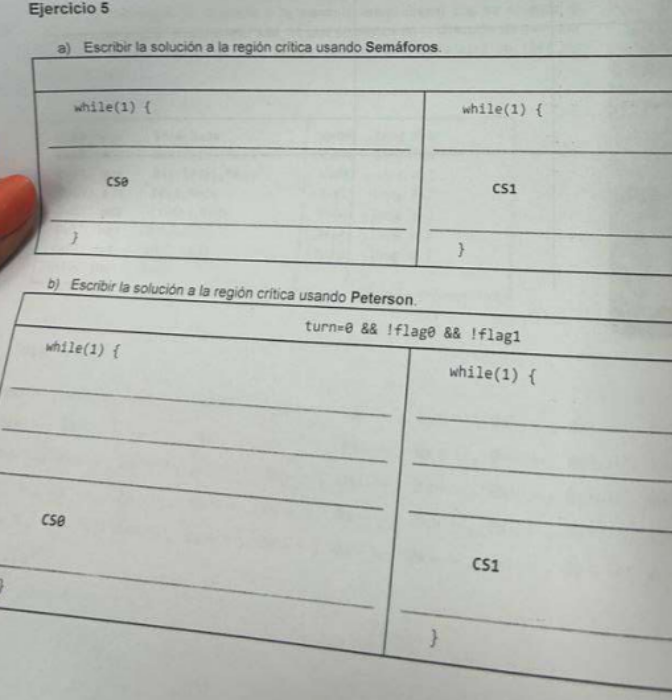
3 0011

2 0010

1 0001

tomo como valor minimo 32=

| s=0, sem1 = 1; sem2,sem3 = 0; | | |
| --- | --- | --- |
| while (s<32){  sem\_wait(sem1)  s =s+1;  sem\_post(sem2);  } | while (s<32){  sem\_wait(sem2)  s = s\*2;  if(s==4){  sem\_post(sem3)  }else {  sem\_post(sem2)  }  } | while (s<32){  sem\_wait(sem3)  s = s<<3;  if (s>=32){  sem3 = 0  } |
| s = ? | | |



while(1){

turn = 1;

flag0 = 1; //estoy queriendo ejecutarme

if (turn = 1 && flag1){

wait

}else{

cs0

}

flag0 = 0

}

while(1){

turn = 0;

flag1 = 1; //estoy queriendo ejecutarme

if (turn = 0 && flag0){

wait

}else{

cs0

}

flag1 = 0

}

| sem0 = 1 ; sem1=0; | |
| --- | --- |
| while(1){  sem\_wait(sem0)  cs0  sem\_post(sem1)  } | while(1){  sem\_wait(sem1)  cs1  sem\_post(sem0)  } |

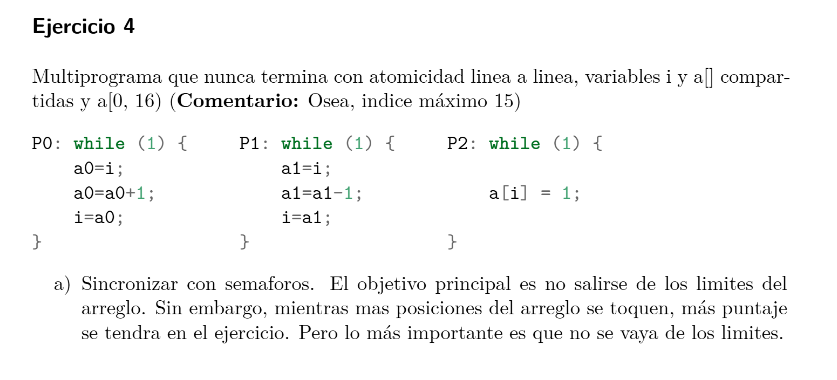
algoritmo de peterson: alto petero

soluciona los problemas en zonas criticas con variables compartidas (flag y turn) sin semaforos.

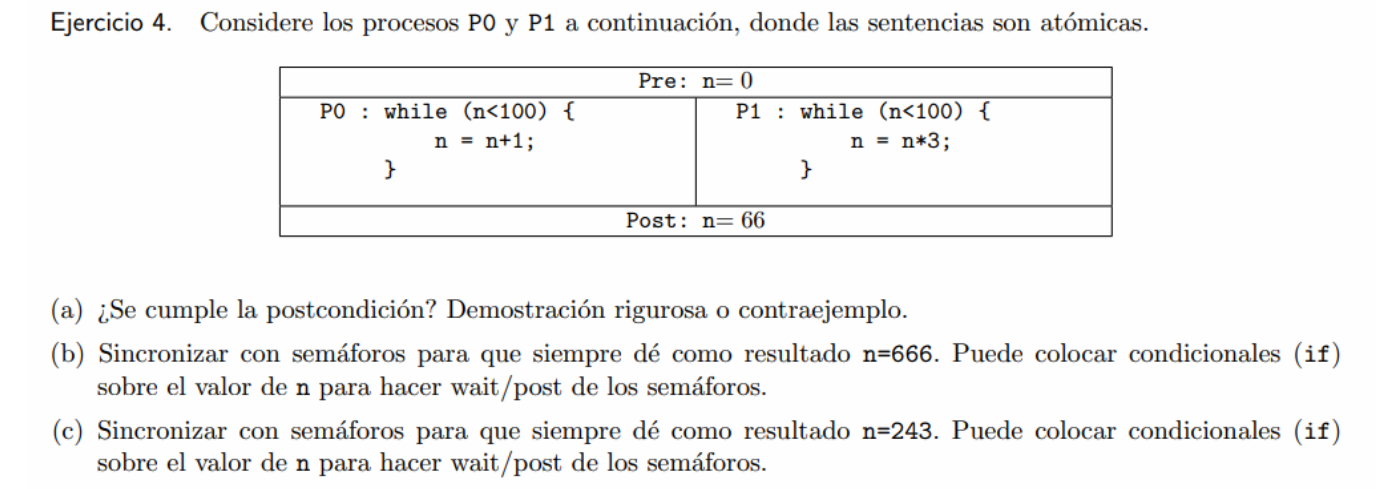
flag[2] = {false,false}// indican si el p0 o p1 quieren entrar a la seccion critica

turn = a quien le toca si ambos quieren entrar al mismo tiempo

| turn = 0; !flag0 && !flag1 | |
| --- | --- |
| while(1){  flag0 = true;  turn =1  while (flag1 && turn = 1){waits}  cs0  flag0 = false;  } | while(1){  flag1 = true;  turn = 0  while (flag0 && turn = 0){waits}  cs1  flag1=false;  } |



| i=0; sem0,sem1= 0; sem2 =1; | | |
| --- | --- | --- |
| while(1){  sem\_wait(sem0)  a0=i;  a0 = a0 + 1;  i = a0;  sem\_post(sem2)  } | while(1){  sem\_wait(sem1)  a1 =i ;  a1 = a1-1;  i = a1;  sem\_post(sem2)  } | while(1){  sem\_wait(sem2)  a[i]=1;  if(i==15){  sem\_post(sem1)  }else{  sem\_post(sem0)  }  } |



1. supongo el caso en el que solo se ejecuta p0 => n=100

“” solo p1 => n=0

1 caso de p0 y luego p1 hasta finalizar = 1\*3 = 3 \*3 =9 \*3 = 27 \*3=81 \*3= 243 resultado final

no hay forma de que post: n=66 sea verdadera porque la unica instancia en que los procesos se dejen de ejecutar seria con n>=100

b)

| pre: n=0; s0 =1; s1 = 0; | |
| --- | --- |
| P0: while (n<100){  sem\_wait(s0)  if (n=74){  sem\_post(s1)  } else {  n = n + 1;  sem\_post(s0)  }  } | P1: while (n<100){  sem\_wait(s1)  if(n!=666){  n = n \* 3;  sem\_post(s1)  }  } |
| n = 666 | |

c)

| pre: n=0; s0 =1; s1 = 0; | |
| --- | --- |
| P0: while (n<100){  sem\_wait(s0);  n = n + 1;  sem\_post(s0);  } | P1: while (n<100){  sem\_wait(s1)  if (n!=243){  n = n \* 3;  sem\_post(s1);  }  } |
| n = 243 | |

⡴⠒⣄⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⣼⠉⠳⡆⠀

⣇⠰⠉⢙⡄⠀⠀⣴⠖⢦⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠘⣆⠁⠙⡆

⠘⡇⢠⠞⠉⠙⣾⠃⢀⡼⠀⠀⠀⠀⠀⠀⠀⢀⣼⡀⠄⢷⣄⣀⠀⠀⠀⠀⠀⠀⠀⠰⠒⠲⡄⠀⣏⣆⣀⡍

⠀⢠⡏⠀⡤⠒⠃⠀⡜⠀⠀⠀⠀⠀⢀⣴⠾⠛⡁⠀⠀⢀⣈⡉⠙⠳⣤⡀⠀⠀⠀⠘⣆⠀⣇⡼⢋⠀⠀⢱

⠀⠘⣇⠀⠀⠀⠀⠀⡇⠀⠀⠀⠀⡴⢋⡣⠊⡩⠋⠀⠀⠀⠣⡉⠲⣄⠀⠙⢆⠀⠀⠀⣸⠀⢉⠀⢀⠿⠀⢸

⠀⠀⠸⡄⠀⠈⢳⣄⡇⠀⠀⢀⡞⠀⠈⠀⢀⣴⣾⣿⣿⣿⣿⣦⡀⠀⠀⠀⠈⢧⠀⠀⢳⣰⠁⠀⠀⠀⣠⠃

⠀⠀⠀⠘⢄⣀⣸⠃⠀⠀⠀⡸⠀⠀⠀⢠⣿⣿⣿⣿⣿⣿⣿⣿⣿⣆⠀⠀⠀⠈⣇⠀⠀⠙⢄⣀⠤⠚⠁⠀

⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⡇⠀⠀⢠⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡄⠀⠀⠀⢹⠀⠀⠀⠀⠀⠀⠀⠀⠀

⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⡀⠀⠀⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⡀⠀⠀⢘⠀⠀⠀⠀⠀⠀⠀⠀⠀

⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⡇⠀⢰⣿⣿⣿⡿⠛⠁⠀⠉⠛⢿⣿⣿⣿⣧⠀⠀⣼⠀⠀⠀⠀⠀⠀⠀⠀⠀

⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⢠⡀⣸⣿⣿⠟⠀⠀⠀⠀⠀⠀⠀⢻⣿⣿⣿⡀⢀⠇⠀⠀⠀⠀⠀⠀⠀⠀⠀

⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠘⡇⠹⠿⠋⠀⠀⠀⠀⠀⠀⠀⠀⠀⠙⢿⡿⠁⡏⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀

⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠻⣤⣞⠁⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⢢⣀⣠⠇⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀

⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠙⠲⢤⣀⣀⠀⢀⣀⣀⠤⠒⠉⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀⠀