Ejercicio 3. Considere el siguiente multiprograma de dos componentes P0 y P1, donde la **atomicidad es línea a línea** y todas las variables (i, j, a) son compartidas.

Pre: i=0 $\wedge$ j=0	

P0 : while(i<16) {
a[i] = j;
++i;
}
	P1 : while(j<32) {
	++j;
	}

Post: ?	

(a) Exprese de manera concisa todos resultados posibles en el arreglo $a[0, 16)$. Explique.
(b) ¿Se puede cumplir con la postcondición $(\forall k : 1 \leq k < 16 : a[k-1] < a[k])$? Explique.
(c) Sincronice con semáforos para lograr que **siempre** se satisfaga la postcondición $(\forall k : 1 \leq k < 16 : a[k-1] < a[k])$ sin hacer peligrar la terminación y maximizando la concurrencia. Solo puede agregar ifs para hacer los wait y post, no se puede tocar el resto del programa.

a) Los resultados posibles para el arreglo son todas las permutaciones posibles en sus entradas de los valores que van en un rango de 0 a 31. Es decir, para cada entrada tenemos 32 opciones, como son 16 entradas, esto es 32^16 posibles resultados para el arreglo a.

b) El b) nos pide cumplir que para todos los valores de k ∈ [1,15] se cumpla que a[k-1] < a[k]. Lo cual si es posible, nos basta con para los índices desde el 1 hasta el 15 el multiprograma sea alternado, primero P0 asigna un valor, luego P1 incrementa en uno y repetimos.

¿En qué valores puede ir aumentando j?

j = 32, si aumentamos a j de a 3 en 3 tendríamos
[...] [0] [3] [6] [9] [12] [15] [18] [21] [24] [27] [30] [33] [36] [39] [42]
[...] [0] [3] [6] [9] [12] [15] [18] [21] [24] [27] [28] [29] [30] [31] [32]
si aumentamos a j de a 2 en dos, podemos obtener
[...] [0] [2] [4] [6] [8] [10] [12] [14] [16] [18] [20] [22] [24] [26] [28]

Lo cual si es alcanzable por el rango posible de j.

¿Cómo calculamos eso?

Si i = 9 y a[9] = 24.
Nos queda 6 posiciones más 15 - 9 = 6
Luego j = 24 podemos aumentar de a 3?

```
if (a[i-1] > j)
        sem_up();
```

if (31 - j) / (15 - i) ≥ 1
    j++
y hacemos sem_up? yo diria de que si eso es menor a 1, entonces hacemos un sem_down(limit)

SE PUEDE AUMENTAR, es decir, siempre y cuando se puedan distribuir los valores en las posiciones restantes teniendo en cuenta que j sólo puede llegar hasta 31.

```
increment = 0
limit = 0

while(i<16) {
    while (i != 0 && a[i-1] > j) {
        sem_down(increment);
    }
    a[i] = j;
    i++
    sem_up(limit)
}

while(j<32) {
    while ((31 - j) / (15 - i) < 1 ) {
        sem_down(limit);
    }
    j++
    sem_up(increment)
}
```

Ejercicio 4. Debajo se muestra solución **incorrecta** al problema de la sección crítica de dos procesos, presentada por Hyman en 1966, a fin de competir con el algoritmo de Dekker.

(a) Muestre un **escenario de ejecución** donde no se cumple la condición de sección crítica.

(b) ¿**Siempre** funcional mal? Explique.

$Pre : \neg flag_0 \wedge \neg flag_1 \wedge turn = 0$

P_0 :	**do**	$true \rightarrow$		P_1 :	**do**	$true \rightarrow$	
1		NCS_0		A		NCS_1	
2	;	$flag_0 := true$		B	;	$flag_1 := true$	
3	;	**do** $turn = 1 \rightarrow$		C	;	**do** $turn = 0 \rightarrow$	
4		**do** $flag_1 \rightarrow$	**skip**	D		**do** $flag_0 \rightarrow$	**skip**
5		**od**		E		**od**	
6		$;turn := 0$		F		$;turn := 1$	
7		**od**		G		**od**	
8	;	CS_0		H	;	CS_1	
9	;	$flag_0 := false$		I	;	$flag_1 := false$	
	od				**od**		

a) ABCDE (flag1 = true, turn = 0, flag0 = false) ningún problema
1238 (flag0 = true, turn = 0 y saltó a la zona crítica)
FGH (turn = 1, entró en la zona crítica)

b) No siempre funciona mal, solo cuando p1 pasa todas las guardas, y antes de poner turn en 1, p0 se mete en medio y la guarda del turn = 1 da falso, saltando directamente a la zona crítica.

Ejercicio 5. El disco rotacional *Seagate Mach.2 Exos 2X14* de 14 TiB e intefaz SAS 3.0, tiene una velocidad de rotacional de 7200 RPM, 4.16 ms de latencia de búsqueda y 524 MiB/s de tasa de transferencia máxima.
Este es **el disco rotacional más rápido del mundo** y eso es gracias a que tiene **dos juegos de cabezales independientes**.

(a) Indicar cuantos *ms* tarda en dar una vuelta completa.

(b) Indicar la tasa de transferencia de lectura **al azar** de bloques de 8 MiB [1].

(c) Si la tasa de transferencia máxima está dada por la velocidad rotacional que no requiere cambio de pista (no sufre del *seek time*), deducir cuantos MiB almacena cada cilindro.

7200RPM
4.16 ms seek
524 MiB/s de tasa de transferencia máxima

Que sean independientes hace que duplique la tasa de transferencia de y además reduce por la mitad la latencia de búsqueda. Lo interpreto como que el x2 en la tasa de transferencia es

para ambos, y el seek ya está dividido por la mitad, como fabricante me interesa más vender con esos números antes de tener que especificar que es por cada cabezal los valores.

(a)

7200 RPM → 120 RPS
1 / 120 RPS = 0.0083s
8.33ms Tarda en dar una vuelta

(b)

T_rotation = 8.33ms / 2 = 4.165ms
T_seek = 4.16
T_transfer = 8MiB / 524 MiB/s = 0.015s → 15.27ms

T_i/o = 4.165ms + 4.16ms + 15.27ms = 23.595ms

R_i/o = 8MiB / 23.595ms = 0.339MiB/ms = 339MiB/s

(c)

524 MiB/s de tasa de transferencia máxima.
Se tarda 8.33ms en dar una vuelta completa.

1s → 524 MiB/s
0.0083s → 4.35MiB Datos por vuelta.

Ejercicio 6. En un sistema de archivos de tipo UNIX, tenemos los bloques de disco dispuestos dentro del *i-nodo* con 12 bloques directos, 1 bloque indirecto, 1 bloque doble indirecto, 1 bloque triple indirecto. Cada bloque es de 4 KiB.
(a) Calcule la capacidad máxima **de un archivo** para números de bloque de 16, 24 y 32 bits.
(b) Calcule la capacidad máxima **de la *data region*** para números de bloque de 16, 24 y 32 bits.
(c) Realice un análisis de que longitud conviene para codificar el número de bloque.

(a)

16 bits:
BS = 2 bytes
BN = 4 KiB / 2 bytes = 2048 bloques

max_file_size = 4KiB * (12 + 2048 + 2048^2 + 2048^3) = Mucho más de 4 TiB

24 bits:

BS = 3 bytes
BN = 4 KiB / 3 bytes = 1365 bloques

max_file_size = 4KiB * (12 + 1365 + 1365^2 + 1365^3) = 4 TiB

32 bits:
BS = 4 bytes
BN = 4 KiB / 4 bytes = 1024 bloques

max_file_size = 4KiB * (12 + 1024 + 1024^2 + 1024^3)

(b)

16 bits: data_region = 2^16 * 4 KiB = 256 MiB
24 bits: data_region = 2^24 * 4 KiB = 64 GiB
32 bits: data_region = 2^32 * 4 KiB = 16 TiB

(c)

Conviene el de 32 bits, ya que si bajamos mucho los bits (16) lo que ocurre es que aumenta el tamaño de max_files pero se reduce mucho la capacidad del disco. Lo mismo para con (24) bits. De hecho el tamaño máximo de archivo para el de 24 bits supera lo que la data región puede direccionar. Entonces si o si el de 32 bits.