Ejercicio 1

El siguiente multiprograma corre LEGv8 y usa la memoria **compartida** `0x20000`. Dar los valores posibles que puede contener esa dirección de la RAM al finalizar el multiprograma.

{ M[0x20000] = 0 }		
1: MOVZ X8,#2,LSL #16 2: LDUR X1,[X8,#0] 3: ORR X1,X1,#1 4: STUR X1,[X8,#0]	A: MOVZ X8,#2,LSL #16 B: LDUR X1,[X8,#0] C: ORR X1,X1,#2 D: STUR X1,[X8,#0]	a: MOVZ X8,#2,LSL #16 b: LDUR X1,[X8,#0] c: ORR X1,X1,#4 d: STUR X1,[X8,#0]
{ M[0x20000] = ?}		

Esto era

0x1
…
0x7

Ejercicio 2

Se intenta una solución al "Simple Flag" del OSTEP. Se re-testea la guarda N veces para no caer en el viejo truco de preguntar, que te chafen el mutex y entrar como un campeón. La implementación de la región crítica se llama "Simple Flag^N" y fue diseñada por Nelson "Big Head" Bighetti.

{ mutex=0, N=4 }	

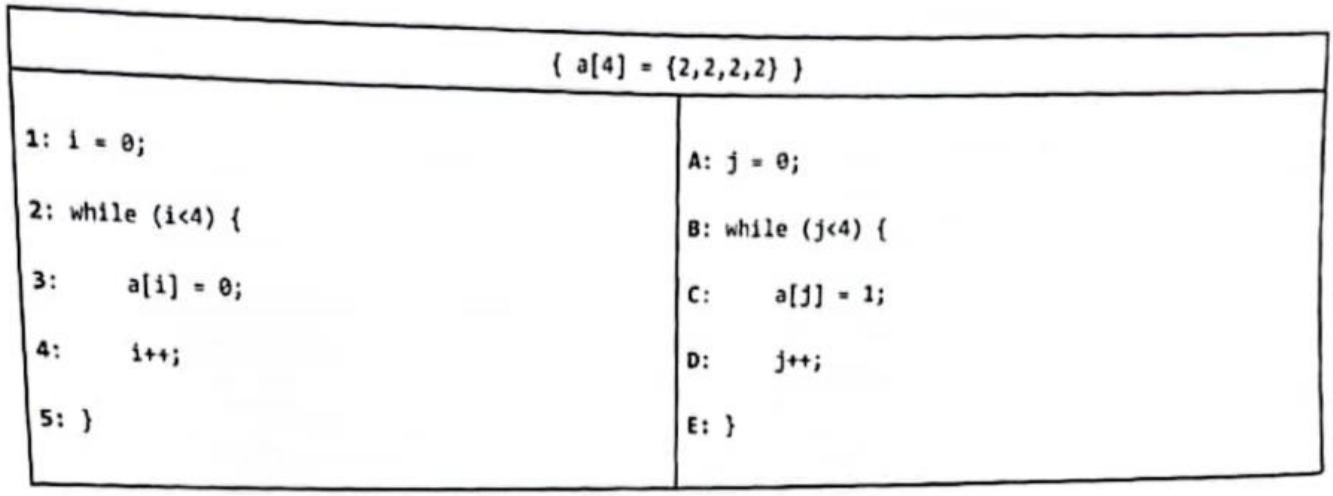
1: for(i=0; i<N; i++) // N veces el
2: while(mutex==1); // spinlock
3: mutex = 1; // tomar mutex
4: CS0;
5: mutex = 0; // soltar mutex
	A: for(i=0; i<N; i++) // N veces el
	B: while(mutex==1); // spinlock
	C: mutex = 1; // tomar mutex
	D: CS1;
	E: mutex = 0; // soltar mutex

Si es correcta, dar argumentos rigurosos. Si no lo es, un contraejemplo.

No funciona, haces N veces ambos forks de forma tal que ambos queden en la última evaluación, uno evalúa falso, sale del for, se rompe, cambia de contexto, evalúa lo mismo, toma el mutex, volvemos y se toma de igual forma.

Ejercicio 3

Al siguiente multiprograma se lo denomina *Concurrent Vector Writing*. **Suponga atomicidad linea a linea**. Agregar **semáforos entre lineas** para que el resultado final sea {1,1,1,1}. Hay varias soluciones posibles. Cuanto más concurrente, es decir más escenarios de ejecución tenga, mejor.



{ a[4] = {2,2,2,2} }	
1: i = 0; 2: while (i<4) { 3: a[i] = 0; 4: i++; 5: }	A: j = 0; B: while (j<4) { C: a[j] = 1; D: j++; E: }

s0 = 0

```
i = 0;
while(i<4) {
        a[i] = 0;
        i++;
        sem_post(s0)
}

j = 0;
while(j<4) {
        sem_wait(s0)
        a[j] = 1;
        j++;
}
```

Ejercicio 4

El disco duro rotacional' **Seagate BarraCuda de 1 TB** e interfaz SATA3 (ST1000DM010) tiene las características de la fila central de la tabla y **cuesta 70 USD**. Se calcula y presenta la $R_{i/o}$ velocidad de lectura al azar para bloques de 4 KiB.
Se está evaluando **duplicar la velocidad de rotación** "Ver.UpRPM" que incrementa el precio final de venta en 50 USD, o bien **reducir a la mitad el tiempo promedio de búsqueda** "Ver.DownSeek" y esto incrementa el costo en 45 USD.
El cociente $R_{i/o}$/Precio mide el **costo del disco para lecturas al azar** y se mide en KiB/s por USD. Calcularlo y ordenar los discos del más conveniente al menos conveniente.

Nombre	RPM	Seektime [ms]	Transf. [MiB/s]	$R_{i/o}$ [KiB/s]	$R_{i/o}$/Precio
Ver.UpRPM	15000	8.5	210	380.28	
Barracuda	7200	8.5	210	315.33	
Ver.DownSeek	7200	4.25	210	474.20	

Orden de conveniencia. Poner el nombre:

#1 _____ #2 _____ #3 _____

Ver.UpRPM Incrementa el precio final en 50 USD, es decir, aumenta un %71 su precio.
R_i/o = 3.169 KiB/s per USD

Ver.DownSeek Incrementa el precio final en 45 USD, es decir, aumenta un %64
R_i/o = 4.12 KiB/s per USD

Barracuda
R_i/o = 4.5 KiB/s per USD

#1 Barracuda #2 Ver.DownSeek #3 Ver.UpRPM

Ejercicio 5

Se tiene un Sistema de Archivos *UNIX-like* con un esquema diferente, pues tiene **varios indirectos** y **varios dobles indirectos**. Tamaño de bloque, 1 KiB. **Tamaño de índice de bloque, 24 bits**. Bloques directos, indirectos, doble indirectos: 8, 8, 8. Dar la **secuencia estrictamente creciente** de tamaños de archivos que van marcando el paso de Directos a Indirectos y a Doble Indirectos, hasta el máximo tamaño de archivo posible.

0 KiB → _____ KiB → _____ KiB → _____ KiB

Directos | Indirectos | Dobles Indirectos

Bloque pesa 24 bits = 3 bytes
8 directos
8 indirectos
8 doble indirectos

0 KiB → 8 KiB → 2736 KiB → 932992 KiB