Ejercicio 3.

(a) Muestre un escenario de ejecución que termine en `a={0,1,0,1,0,1 ...}` para la Figura 1, suponga siempre **atomicidad es línea-a-línea**.

(b) Agregue sincronización con semáforos para que el valor de salida sea siempre el del punto anterior. Puede colocar condicionales (`if`) sobre el valor de las variables `i` y `j` para hacer wait/post de los semáforos. No puede tocar los incrementos de las variables y las asignaciones al arreglo.

Pre: $0 < N \wedge i,j = 0,0 \wedge (\forall k : 0 \leq k < N : a[k] = 2)$	
1 P0: while (i<N) {	a P1: while (j<N) {
2 a[i] = 0;	b a[j] = 1;
3 ++i;	c ++j;
}	}

(a)

Planificación:

abc 123 123 abc abc 123 123 abc para N = 4

Para cualquier N, la planificación es “abc 2(123) 2(abc) … 2(123) 2(abc) abc”

Es decir, una ejecución de P0 al comienzo, luego intercalar 2 ejecuciones P1 y P0 seguidas,

finalmente P0 termina con una ejecución.

(b)

Pre: 0 <N ^ i,j = 0,0 ^ (∀ k : 0≤ k < N : a[k] = 2) s0 = 0 ^ s1 = 0	

P0
while (i<N) {
if(i mod 2 == 0)
wait(s1)
 	
a[i] = 0;
++i;
 	
if(i mod 2 == 0)
post(s0)
}
	P1
	while (j<N) {
	a[j] = 1;
	++j;
	
	if(j mod 2 != 0)
	post(s1)
	wait(s0)
	}

Ejercicio 4. Considere los procesos P0 y P1 a continuación, donde las sentencias **no son** atómicas.

Pre: n= 0	

P0 : while (n<100) {
n = n*2;
}
	P1 : while (n<100) {
	n = n+1;
	}

(a) ¿Qué valores posibles puede tomar n al terminar el multiprograma?. Explique.

(b) Sincronizar con semáforos para que se comporte como la versión atómica dada en el práctico que da valores de n entre 100 y 200.

(a)
una vez p1, luego p0
n = 1
n = 2, 4, 8, 16, 32, 64, **128**
solo p1
n = **100**

n puede llegar hasta 50 en p0. O sea por un lado los valores **finales** posibles podrían ser, 2 * k en donde k va desde 50 a 100 (100 pq existe un caso en donde se evalúa el while en n = 99 y luego el load es de n = 100)

(b)

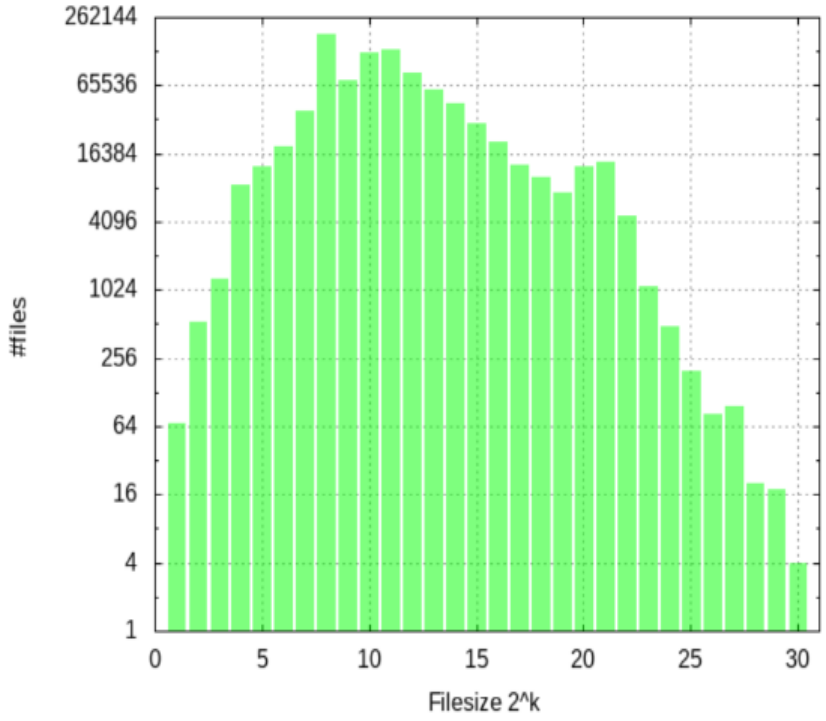
```
while(n<100){
        wait(s1)
        n = n*2
        post(s0)
}


while(n<100){
        wait(s0)
        n = n+1
        post(s1)
}
```

Ejercicio 5. Usted forma parte de un equipo al que le fue encargado el diseño de un sistema de archivos *UNIX-like* y uno de sus compañeros propuso la idea de de aprovechar los 75 bytes que sobran en los i-nodos (Cada i-nodo ocupa 181 bytes y se decidió "alinearlo" a 256 bytes para que entraran 2 por bloque de disco) almacenando allí los archivos que no superen esta capacidad[1].

(a) **Discuta** porque si o porque no esta **decisión de diseño** es acertada, en cuanto a eficiencia, velocidad y simpleza. Soporte sus argumentos con las mediciones[2] de la Figura 2.

(b) ¿Tiene sentido que estos 75 bytes **siempre se utilicen** sin importar la longitud del archivo? Discuta.



- Son más prominentes en cantidad los archivos chicos, entonces nos conviene mantener a esos en los propios inodos y no gastamos tiempo en buscar los bloques de datos, el disco es muy lento.
- No pues, un archivo muy grande es mejor tenerlo en bloques secuenciales antes que leer el inodo, luego saltar al bloque de datos, además cada vez que quieras entrar al inodo a ver metadata, te podrías topar con una sobrecarga de trabajo.

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 y 1 bloque doble indirecto. Cada bloque es de 4 KiB.

(a) Calcule la capacidad máxima del *block pool* para números de bloque de 16, 24 y 32 bits.

(b) Calcule la capacidad máxima de un archivo para números de bloque de 16, 24 y 32 bits.

(c) Realice un análisis de que longitud de número de bloque conviene.

(a) 2^16 = 64 KiB

2^24 = 16 MiB
2^32 = 4 GiB

(b)

16 bits / 8 = 2 bytes
2^12 = 4096 / 2 = 2048 archivos
4 KiB (12 + 2048 + 2048^2 + 2048^3)

…

24 bits / 8 = 3 bytes
2^12 = 4096 / 3 =1365
4 KiB (12 + 1365 + 1365^2 + 1365^3)

32 bits / 8 = 4 bytes
2^12 = 4096 / 4 = 1024
4 KiB (12 + 1024 + 1024^2 + 1024^3)

(c) Conviene el de 32 bits, el de 16 bits es re poco espacio no entra ni un archivo, lo mismo pasa con el de 24 bits.