Ejercicio Comunicación/Sincronización

```
#define TIPO1           1
#define TIPO2           2
#define MAX_PAQUETES    100
int nTipo1=0, nTipo2=0;
semaphore semRep1(0), semRep2(0), mutex(1), almacen(MAX_PAQUETES);
```

```
Recepcionista () {
  int tipo;
  while(1){
    wait(almacen);
    tipo=atenderCliente();
    switch(tipo){
    case TIPO1:
       wait(mutex);
       nTipo1++;
       post(mutex);
       //Avisar a repartidores tipo 1
       post(repartidor1);
       break;
    case TIPO2:
       wait(mutex);
       nTipo2++;
       post(mutex);
       //Avisar a repartidores tipo 2
       post(repartidor2);
       break;
    }
  }
}
```

```
Repartidor(int tipo) {
  while(1){

    if(tipo== TIPO1){
       wait(repartidor1);
       wait(mutex);
       nTipo1--;
    }
    else{
       wait(repartidor2);
       wait(mutex);
       nTipo2--;
    }
    post(mutex);
    post(almacen);
    repartir(tipo);
  }

}
```

Si usamos procesos emparentados lo mejor sería crear una región de memoria compartida (no necesitamos que tenga fichero asociado) y 'meter' dentro semáforos sin nombre, y resto de variables. Los hijos heredarán dicha región.

En caso de procesos no emparentados necesitamos un mecanismo con nombre, podemos usar lo mismo que antes pero con una región de memoria compartida con nombre o usar directamente semáforos con nombre (pero tendríamos problemas con las variables compartidas).

Ejercicio Entrada/Salida

Capacidad=cilindros·cabezas·sectores·tam.sector=100*10*20*512=10.240.000B=9.77MB

FIFO: A-B-C-D-E-F
SSTF: E-B-A-C-D-F (*)
SCAN: A-C-D-F-E-B (*)
C-SCAN: A-C-D-F-B-E (*)

A(90,5,5)->B(10.9,6)
1/(3000/60)=20ms por vuelta (1ms por sector)
0.5ms por cilindro -> ½ sector por cilindro, de 90 a 10 -> 0.5*80=40ms
al mover 80 cilindros avanzo 40 sectores -> da 2 vueltas justas, luego al llegar estaré en (10,x,5), necesito 1ms más para ir al (10,x,6) -> 41ms

(*)Se dijo durante el examen que sólo se tuviese en cuenta el cilindro para elegir la petición y el sector para el desempate. Todas las variaciones razonables se han considerado válidas debido a la posible ambigüedad existente.

Ejercicio Memoria

0x1F -> página 0 -> marco 4 -> 0x9F
0x20 -> página 1 -> fallo de página
0x93 -> página 4 -> marco 2 -> 0x53
0xFF -> página 7 -> fallo de página

Por la instrucción 2 accesos:

- 1 acceso a TP 0xD0 -> página 6 -> marco 0 -> 0x10 (acierto),
- 1 a la instrucción en si

Por el dato un mínimo de 33 accesos:

- 1 acceso a TP por 0x20 -> fallo de página
- Al menos 32 bytes por traer la página necesaria de HDD
- Puede haber más accesos debido a una posible expulsión de página sucia y por ejecución de algoritmo de paginación.

Ficheros y procesos

Tabla FD Padre	Tabla FD Hijo
STDIN	STDIN
STDOUT	STDOUT
STDERR	STDERR
fd1 -> id-A	fd1 -> id-A
fd2 -> id-C(*)	fd2 -> id-B

Tabla SO		
Id	Punt.LE	Fichero
id-A	6K5	test1
id-B	4K	test2
id-C(*)	0	test1

(*)Puede estar, dependiendo de si consideramos que tiene prioridad el padre sobre el hijo. En caso de que el hijo haga **close(fd2)** antes de que el padre abra otra vez **tes1** esta entrada no estaría.

Contenido de **test1**

Región	0 -> 1K-1	1K -> 2K-1	2K -> 3K-1	3K -> 6K-1	6K -> 6K5-1
Contenido	111….111	111….111	333….333	vacío	222…222

Contenido de **test2**

Región	0 -> 4K-1
Contenido	333…333

El fichero tiene un tamaño lógico de 6K5Bytes, lo que supondría un total de 13 bloques para los datos (2 bloques por KB) si el espacio vacío ocupase, pero como no es así, el fichero necesita 3*2=6 bloques de datos para los primeros 1's y 3's y uno más para los últimos 2's. El espacio vacío no ocupa bloques de datos y se lee como 0's.

En la tabla de bloques indirectos caben 512/4=128 entradas. El último valor accesible es 6K5, lo que supone el bloque lógico 13, así que necesitamos los 2 directos más 11 indirectos. Esto supone que necesitamos un bloque de datos adicional (además de los 7 de datos) para almacenar los índices indirectos.

```
500 -> bloque de datos 0
501 -> bloque de datos 1
502 -> indirecto simple
503 -> bloque de datos 12 (entrada 10) (*)
504 -> bloque de datos 2  (entrada 0)
505 -> bloque de datos 3  (entrada 1)
506 -> bloque de datos 4  (entrada 2)
507 -> bloque de datos 5  (entrada 3)
```

¡¡Este bloque se pidió antes!! aunque no se ha tenido el cuenta el orden en la corrección.