Ejercicio Comunicación/Sincronización

```
#define TIPO1           1
#define TIPO2           2
#define MAX_PAQUETES    100
int nTipo1=0, nTipo2=0;
mutex mAlmacen
variable de condición vcAlmacen, vcTipo1, vcTipo2;
```

```
Recepcionista () {
  int tipo;
  while(1){
    lock(mAlmacen)
    while(nTipo1+nTipo2==MAX_PAQUETES)
      wait(vcAlmacen, mAlmacen);
    unlock(mAlmacen)
    tipo=atenderCliente();
    lock(mAlmacen);
    switch(tipo){
      case TIPO1:
        nTipo1++;
        //Avisar a repartidores tipo 1
        if(nTipo1==1)
          signal(vcTipo1);
        break;
      case TIPO2:
        nTipo2++;
        //Avisar a repartidores tipo 2
        if(nTipo2==1)
          signal(vcTipo2);
        break;
    }
  }
  unlock(mAlmacen)
}
```

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

    lock(mAlmacen)

    if(tipo==TIPO1){
      while(nTipo1==0)
        wait(vcTipo1, mAlmacen);
      nTipo1--;
    }
    else{
      while(nTipo2==0)
        wait(vcTipo2, mAlmacen);
      nTipo2--;
    }

    if(nTipo1+nTipo2==MAX_PAQUETES-1)
      signal(vcAlmacen);

    unlock(mAlmacen)

    repartir(tipo);
  }

}
```

Puede haber todos los repartidores que queramos, ya que acceden de manera excluyente al almacén y avisan al recepcionista en caso de que haya un hueco. En cuanto a los recepcionistas, sólo puede haber uno en la configuración actual, ya que si no es así se puede saturar el almacén. Podría haber múltiples recepcionistas haciendo atenderCliente y que nTipo1+nTipo2 pase a ser mayor que MAX_PAQUETES

Lo podríamos solucionar manteniendo una variable global nPaquetes que mantuviese la cuenta del número de total de elementos en el almacén, Habría también que incrementar/decrementar esta variable cuando se produzca/consuma.

```
while(nPaquetes=MAX_PAQUETES)
  wait(vcAlmacen);
nPaquetes++;
unlock(mAlmacen);
tipo=atenderCliente();

nTipoX--;nPaquetes--;
```

Ejercicio Entrada/Salida

1) Leer 51200 es lo mismo que leer 100 sectores ->
   1/(3000/60) = 20ms por vuelta. Con 20 sectores por pista -> 1ms por sector
   100 * (1+20/2+6*13) = 8.9sec (disperso)
   100 * (1+20/2+6*2) = 2.3sec (compacto)
2) Caso mejor: leemos 100 sectores y desplazamos 1 cilindro (hay 20*4 sectores por cilindro, el fichero no cabe en 1 cilindro sino en 2)
   100+6 = 0.106s
   Caso peor: para cada sector desplazamos la cabeza todo el disco (40-1 cilindros) y tenemos que esperar una vuelta entera para empezar a leer
   100*(1+20+6*39)=25.5sec

Ejercicio Memoria

1) TMP:
   entrada 0: P=1, marco = 2
   entrada 1: P=1, marco = 5
   entradas 2-5: P=0 → no válidas
   entrada 6: P=1, marco = 6
   entradas 7-43: P=0 → no válidas
   entrada 44: P=1, marco = 0
   entradas 45-127: P=0 → no válidas

2) TMP:
   entrada 0: P=1, marco = 2
   entrada 1: P=1, marco = 5
   entradas 2-20: P=0 → no válidas
   entrada 21: P=1, marco = 6
   entradas 22-43: P=0 → no válidas
   entrada 44: P=1, marco = 0
   entradas 45-93: P=0 → no válidas
   entrada 94: P=1, marco = 3
   entradas 95-127: P=0 → no válidas

3) dv: 0x0 pag,off (0,0) → df: marco,off (2,0), 2048+0 =2048= 0x800
   dv: 0x410 pag,off (1,16) → df: marco,off (5,16), 5120+16=5136=0x1410

4) dv: 0x812 pag,off (2,18) → ¡¡fallo de página!!

Ficheros y procesos

1) La utilización compartida de los descriptores de fichero de "Origen" y "Destino" por parte de los 4 procesos hace que no se obtenga la necesaria independencia de operaciones de "copia" entre ellos.
   Eventos más relevantes:
   1. El padre posiciona los punteros L/E de Origen y Destino al principio y Crea al hijo 0
   2. El padre posiciona los punteros en 1K y crea al hijo 1
   3. El padre posiciona los punteros en 2K y crea al hijo 2
   4. El padre posiciona los punteros en 3K y crea al hijo 3
   5. El padre hace wait para todos los hijos
   6. El hijo 0 se encuentra el puntero de escritura en la posición 3K y lee de Origen Escribe en destino 1K de números 3 (el puntero avanza)
   7. El hijo 1 intenta leer de Origen y encuentra EOF, no escribe nada
   8. El hijo 2 intenta leer de Origen y encuentra EOF, no escribe nada
   9. El hijo 3 intenta leer de Origen y encuentra EOF, no escribe nada

Contenido de Destino

Región	0 -> 1K-1	1K -> 2K-1	2K -> 3K-1	3K -> 4K-1
Datos				333…333

   Modificaciones de 'buf' por parte del padre:

   El padre espera a los hijos y luego lee con el puntero posicionado a final del fichero (debido al último hijo):
   read(fdd, buf, BLOCK);
   Puesto que lee de FIN-DE-FICHERO, buf queda inalterado: "xxx…x"
   // Posiciona en el tercer bloque
   lseek(fdd,3*BLOCK,SEEK_SET);
   read(fdd, buf, BLOCK);
   Lee en buf "333…333" que es la única escritura con éxito realizada por uno de los procesos hijo

2) Origen: inodo X: tipo: regular; permisos: rw-rw-rw-; tamaño en disco: 4.5KB; Bloques se datos:
   Contenido de los punteros directos: 100 y 101 (0's), 102 y 103 (1's)
   Puntero indirecto 104, puntero bloques: 105 y 106 (2's), 107 y 108 (3's)

3) Versión correcta:

```
#define BLOCK 1024
char buf[BLOCK] = "xxxxxxx...xxxxx";
void main() {
  pid_t pid;
  int fdd,size;
  fdd =
open("Destino",O_RDWR|O_CREAT|O_TRUNC,0666);
  for (int i=0; i < 4; i++) {
    pid = fork();
    if (pid==0){
      copia_bloque(i)
      exit(0);
    }
  }
}
```

```
void copia_bloque(int i){
  int fdo, fdd, size;
  fdo = open("Origen",O_RDONLY);
  fdd = open("Destino",O_WRONLY);
  lseek(fdo,i*BLOCK, SEEK_SET);
  lseek(fdd,i*BLOCK, SEEK_SET);
  size= read(fdo,buf,BLOCK);
  write(fdd,buf,size);
}
```