1. Suponga que varios terminales comparten una impresora. Programe el acceso a la misma utilizando semáforos.

```
Procedure principal
begin
        semáforo em := 1
        cobegin
                terminal1(); terminal2(); … terminaln();
        coend

End


Procedure terminali()
begin
        wait(sem)
        usarImpresora()
        signal(sem)
end
```

2. Considere una sucesión donde cada término es la suma de los dos anteriores. Para calcular las sumas parciales de los términos mpares de la sucesión, existen dos procesos: `sucesión()` y `suma()`. El primero calcula los términos de la sucesión y el segundo las sumas parciales de los términos impares. Se pide implementar el algoritmo concurrente anterior utilizando operaciones sobre semáforos.

```
Procedure principal
begin
        em: semaforo := 1
        impares: semáforo := 0
        buffer: array(0…n-1)
        sumaParcial: entero:= 0
        cobegin
                sucesión(); suma()
        coend
end


Procedure sucesión
begin
        i: entero = 0
        loop
                wait(em)
                buffer[i] := obtenerTermino(i)
                if(i mod 2 = 1)
                        signal(impares)
                signal(em)
                i := i +1
        until(i >= n)
end


Procedure suma()
begin
        i: entero := 1
        loop
                wait(impares)
                wait(em)
                sumaParcial := sumaParcial + buffer[i]
                signal(em)
                i := i + 2
        until(i >= n)
end
```

3. Añadir un semáforo al siguiente programa de modo que siempre imprima 40.

```
Program Aumentar;
Const m=20;
Var n: Integer;

Procedure inc;
Var i :Integer;
Begin
  For y:=1 to m Do
    n:=n+1;
End;

Begin {Principal}
  n:=0;
  Cobegin
    inc;inc
  Coend;
  Writeln (n);
End.
```

```
Program Aumentar;
Const m=20;
Var n: integer;
Var em: semáforo := 1;

Procedure inc;
Var i: Integer;
Begin
        //Aquí también podría ir wait(em)
        for y:=1 to m do
                wait(em)
                n := n +1
                signal(em)
        //Aquí también podría ir signal(em)
end
```

principal igual ;)

4. Se dispone de un sistema con t erminales y n impresoras. En cada terminal se ejecuta un procesos que desea imprimir en alguna de las impresoras. Se pide programar, utilizando semáforos, el acceso de los terminales a las impresoras en exclusión mutua. Debe ser posible que terminales distintos impriman en impresoras distintas.

```
em: semaphore := 1
impresorasDisp: semaphore := n
impresorasLibres: BooleanArray(0… n-1)


    Procedure terminali
    begin
        id: entero
        boolean encontrada
        wait(impresorasDisp)
        wait(em)
        for i:=1 to n-1 and not(encontrada)
            if(impresorasLibres[i])
                encotrada := True
                impresorasLibres[i] = False
                id := i
        signal(em)
        usarImpresora(id)
        wait(em)
        impresorasLibres[i] True
        signal(em)
        signal(impresorasDispo)
    end
```

5. Construir semáforos generales a partir de semáforos binarios.

```
Semaphore em := 1
int S = 0
Semaphore lista := 0
int numProcesosEnLista := 0

Procedure wait()
wait(em)
if(S > 0)
        S = S – 1
        signal(em)
else
        numProcesosEnLista = numProcesosEnLista + 1
        signal(em)
        wait(lista) //bloqueamos el proceso


Procedure signal()
wait(em)
if(numProcesosEnLista = 0)
        S = S + 1
        signal(em)
else
        numProcesosEnLista = numProcesosEnLista - 1
        signal(lista) //desbloqueamos
        signal(em)
```

6. Una carretera cruza dos puentes de una sola vía. Se pide programar, utilizando semáforos, el comportamiento de los coches procedentes de uno y otro extremos del puente (norte y sur), de forma que la solución esté libre de interbloqueos.

```
em1:Semaphore := 1
em2:Semaphore := 1

Procedure CocheDerecha
begin
        wait(em1)
        cruzarPuente1()
        signal(em1)
        wait(em2)
        cruzarPuente2()
        signal(em2)
end


Procedure CocheIzquierda
begin
        wait(em2)
        cruzarPuente2()
        signal(em2)
        wait(em1)
        cruzarPuente1()
        signal(em1)
end
```

7. Una tribu de salvajes cenen en comunidad una gran olla que contiene M misioneros cocinados. Cuando un salvaje quiere comer, se sirve de la olla un misionero, a menos que esté vacía. En este caso, despier al cocinero y espera a que vuelva a llenar la olla. Desarrollar el código de los procesos salvaje y cocinero utilizando semáforos.

```
em: Semaphore := 1
raciones: entero := M
cocina: Semaphore := 0
respuesta: Semaphore:= 0

Procedure Salvajei
begin
        wait(em)
        if(raciones = 0) then
                signal(cocina)
                wait(respuesta)
                signal(em)
        else
                raciones = raciones – 1
                signal(em)

end

Procedure cocinero
begin
        wait(cocina)
        raciones = M
        signal(respuesta)
```

ES LOGICO COÑO >:(

8. Sea un sistema con dos unidades de disco D1 y D2. Existen tres clases de proceso que los utilizan: procesos que sólo utilizan D1, procesos que sólo utilizan D2 y procesos que usan D1 y D2. Desarrollar el código de los tres tipos de procesos utilizando semáforos para controlar el acceso y la liberación de los discos, aprovechando los recursos tanto como sea posible.

Consideraciones: como dijo Juan Carlos, el problema da a interpretaciones, por lo que yo supondré existen 3 procesos y cada uno usa infinitamente el disco que le corresponde, y específicamente el proceso 3 usa primero D1 y luego D2.

```
D1: Semaphore := 1
D2: Semaphore := 1

Procedure process1
Begin
        loop
                Wait(D1)
                UsarDisco1()
                Signal(D1)
        forever
end

Procedure process2
Begin
        loop
                Wait(D2)
                UsarDisco2()
                Signal(D2)
        forever
end

Procedure process3
Begin
        loop
                Wait(D1)
                UsarDisco1()
                Signal(D1)
                Wait(D2)
                UsarDisco2()
                Signal(D2)
        forevere
end
```

8. Sea un sistema con dos unidades de disco D1 y D2. Existen tres clases de proceso que los utilizan: procesos que sólo utilizan D1, procesos que sólo utilizan D2 y procesos que usan D1 y D2. Desarrollar el código de los tres tipos de procesos utilizando semáforos para controlar el acceso y la liberación de los discos, aprovechando los recursos tanto como sea posible.

Consideraciones: En este modelo, lo voy a complicar. Voy a hacer lo mismo que en el anterior, pero que escriba el proceso que pueda en el disco que pueda >:)
em1: Semaphore :=1
em2: Semaphore :=1

```
Procedure process1
Begin
        loop
                wait(em1)
                if(not(UsandoD1))
                        usandoD1 = True
                        signal(em1)
                        UsarDisco1()
                        wait(em1)
                        usandoD1 = False
                        signal(em1)
                else
                        signal(em1)
        forever
end

Procedure process2
Begin
        loop
                wait(em2)
                if(not(UsandoD2))
                        usandoD2 = True
                        signal(em2)
                        UsarDisco2()
                        wait(em2)
                        usandoD2 = False
                        signal(em2)
                else
                        signal(em2)
        forever
end
```

Seminario III - PCTR

```
Procedure process3
Begin
        loop
                wait(em1)
                if(not(UsandoD1))
                        usandoD1 = True
                        signal(em1)
                        UsarDisco1()
                        wait(em1)
                        usandoD1 = False
                        signal(em1)
                else
                        signal(em1)
                        wait(em2)
                        if(not(UsandoD2))
                                usandoD2 = True
                                signal(em2)
                                UsarDisco2()
                                wait(em2)
                                usandoD2 = False
                                signal(em2)
                        else
                                signal(em2)
        forever
end
```

Creo que así vale :3

Francisco Javier Molina Rojas

9. Resolver con semáforos el problema de los lectores/escritores eliminando las prioridades.

```
em: Semaphore := 1
escribiendo: Semaphore := 1
numLectores: integer := 0
seEstaEscribiendo: bool := False

Procedure Lector
begin
        loop
                wait(em)
                if(not(seEstaEscribiendo)) //Inecesario
                        numLectores = numLectores + 1
                        sigal(em)
                        leer()
                        wait(em)
                        numLectores = numLectores – 1
                        signal(em)
                else
                        signal(em)
        forever
end


Procedure Escritor
begin
        loop
                wait(em)
                if(numLectores = 0)
                        wait(escribiendo)
                        escribiendo := True
                        escribir()
                        signal(escribiendo)
                        signal(em)
                else
                        signal(em)
        forever
end
```

10. Se dispone de un proceso productor y k procesos consumidores que se comunican a través de un buffer limitado de n elementos. Productor y consumidores acceden al buffer para insertar información (si hay ranuras libres) o extraerla (si hay datos). Cada mensaje del productor debe ser recibido por los k consumidores, y en el orden en que fueron depositados. Resolver el problema con semáforos.

```
em: Semaphore := 1
elementos : Semaphore := 0
ProdPtr : integer := 0
ConsPtr : integer := 0
buffer : Array(0…n-1)

Procedure Productor
begin
        while(prodPtr < n)
                buffer[ProdPtr] := producirElemento()
                wait(em)
                ProdPtr := ProdPtr + 1
                signal(em)
                signal(elementos)
end


Procedure Consumidor
begin
        wait(elementos)
        wait(em)
        consumir(buffer[ConsPtr])
        ConsPtr = ConstPtr + 1
        signal(em)
end
```