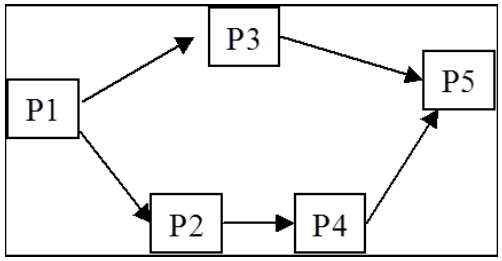
Sesión 7: Sincronización mediante semáforos

Un grafo de precedencias de tareas es un grafo dirigido acíclico que establece un orden de ejecución de tareas de un programa concurrente, de manera que una tarea no puede empezar a ejecutarse hasta que las que sean anteriores en el grafo hayan terminado. Por ejemplo, si consideramos el grafo de la figura 1, la tarea P4 no puede empezar a ejecutarse hasta que P2 y P1 terminen, mientras que P5 no puede empezar hasta que P1, P2, P3 y P4 hayan terminado. Asumamos que cada tarea ejecuta el siguiente código:

espera a que las tareas anteriores terminen

ejecuta el cuerpo de la tarea

avisa de su terminación a quien corresponda



Escribir el programa concurrente de la figura utilizando semáforos para resolver las cuestiones de sincronización. Además, esquematizar un método general de sincronización para un grafo de precedencias general.

Sesión 7: Sincronización mediante semáforos

Considerar el siguiente programa concurrente. Calcular para él el conjunto de los posibles valores finales para la variable x.

integer x := 0		
Semaphore s1 := 1		
Semaphore s2 := 0		
Process P1	*Process* P2	*Process* P3
wait(s2)	wait(s1)	wait(s1)
wait(s1)	x := x*x	x := x+3
x := 2*x	signal(s1)	signal(s2)
signal(s1)		signal(s1)

Dado el siguiente programa concurrente, se pide resolver mediante semáforos que:

- cada vez que el proceso *generador* calcula un nuevo dato, éste es leído una única vez por todos los procesos *lectores*

```
integer N = 10
float dato
...
Process generador::
   ...
   while true
      // calcular un valor para dato
      ...
      dato := ... //el valor calculado
      ...
   end while
end

Process lector(i: 1..N)::
   float miDato
   ...
   while true
      ...
      miDato := dato
      ...
   end while
end
```

Sesión 8: Sincronización mediante semáforos

- El listado que aparece a continuación corresponde a un programa concurrente en el que **nP** procesos permutan valores de un vector de enteros
- Se pide completar el código, utilizando semáforos, de manera que no haya interferencias entre los procesos en el acceso a las componentes del vector.
- No es aceptable una secuencialización total del programa.

```
constant integer n := ...  --n>0
                 nP := ...  --nP>0
integer array[1..n] datos := ... //lo que sea

Process P(i: 1..nP)::
   integer i1,i2,temp

   while true
      read(i1,i2) // 1<=i1<=n AND 1<=i2<=n
      temp := datos[i1]
      datos[i1] := datos[i2]
      datos[i2] := temp
   end
end
```

Sesión 8: Sincronización mediante semáforos

Considérese el siguiente programa concurrente:

```
-------------------------------------------------------------
              semaphore r1 := 1,r2 := 1

    process UNO::                    process DOS::
       loop forever                     loop forever
p0:       wait(r1)                         wait(r2)       :q0
p1:       wait(r2)                         wait(r1)       :q1
p2:       signal(r1)                       signal(r2)     :q2
p3:       signal(r2)                       signal(r1)     :q3
       end loop                         end loop
    end process                      end process
-------------------------------------------------------------
```

Sesión 8: Sincronización mediante semáforos

Se pide:

- Dar un modelo Red de Petri correspondiente al programa
- Construir el grafo de estados del modelo anterior
- Responder a las siguientes preguntas en base al grafo obtenido:
  - ¿Puede el programa llegar a bloquearse? En caso afirmación, decir cuántas historias pueden acabar en bloqueo y dar un ejemplo de ellas
  - ¿Existe alguna historia no equitativa (entendiendo que es una historia infinita en la que a partir de un momento determinado un proceso no interviene)?

Sesión 9: Paso de testigo (semáforos)

N procesos compiten por el uso de un recurso, del que se disponen de **k** unidades. Las peticiones de uso del recurso son del tipo:

- *reservar(r)* : necesito que me concedan "de golpe" r unidades
- *liberar(l)* : libero "de golpe" l unidades que previamente me habían concedido

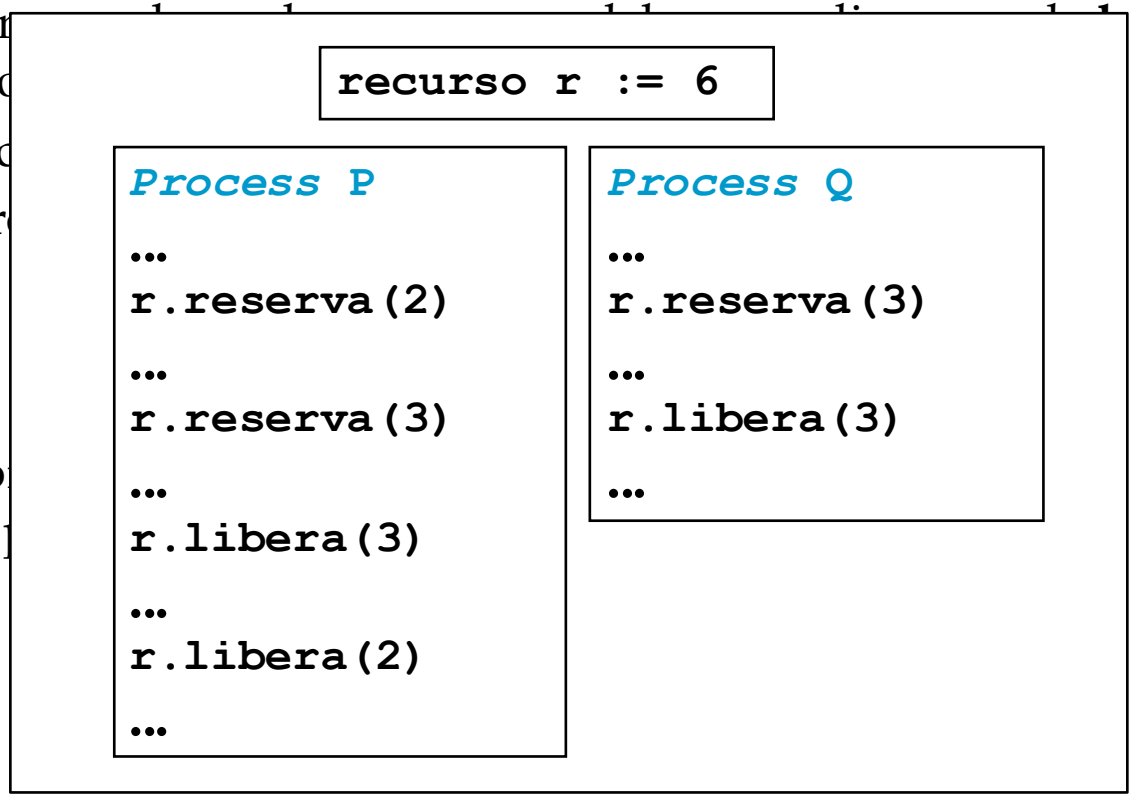
Se pide programar el siguiente programa concurrente resolviendo las cuestiones de sincronización por medio de semáforos. Para ello se debe aplicar la técnica de paso de testigo.

Sesión 9: Paso de testigo (semáforos)

N procesos compiter
unidades. Las peticic

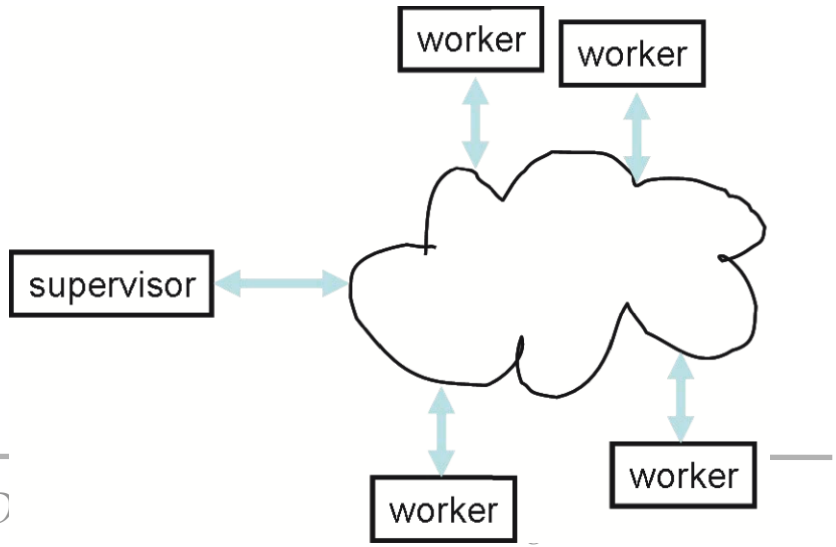
- *reservar(r)* : nec
- *liberar(l)* : liber
  concedido

Se pide programar
cuestiones de sincro
aplicar la técnica de



Sesión 9: Paso de testigo (semáforos)

La figura representa de manera esquemática el patrón de diseño *supervisor-worker*. Este esquema propone una organización sencilla en la que un proceso supervisor suministra tareas a los trabajadores a través de un medio común para el intercambio de información. Los procesos trabajadores van tomando las solicitudes, ejecutan la tarea correspondiente y envían los resultados al supervisor (si los hubiera). Esto se repite hasta que el supervisor le indique que puede terminar. El supervisor toma los resultados y actúa en función a cuál sea su objetivo.



Sesión 9: Paso de testigo (semáforos)

Se pide desarrollar un programa concurrente que ordene 100 ficheros de datos, de manera que un proceso supervisor suministra los nombres de los ficheros a través de una estructura de datos compartida, y cuatro procesos trabajadores van tomando los nombres de los ficheros de la estructura y ordenan el fichero correspondiente. Cada trabajador va iterando ese proceso hasta que recibe del supervisor, de alguna manera, la indicación de que debe terminar. Dependiendo del entorno de ejecución, es posible que distintos procesos trabajador tengan distintas velocidades, por lo que no es aceptable repartir equitativamente los ficheros entre los procesos trabajador.

La sincronización entre los procesos involucrados se llevará a cabo mediante el uso de semáforos, aplicando la técnica de paso de testigo.

Sesión 9: Paso de testigo (semáforos)

Se pide desarrollar un programa concurrente que ordene 100 ficheros de datos, de man
ficheros a trav
trabajadores v
ordenan el fi
proceso hasta
de que debe t
que distintos
no es aceptab
trabajador.

La sincroniza
mediante el us

```
constant natural NUM_WORKERS := 4
constant natural NUM_FICH := 100

//añadir lo que se considere necesario

process supervisor
     string array[NUM_FICH] ficheros := ... //ya inicicializado

    //completar de acuerdo al enunciado del ejercicio
end

process worker(i:1..4)::
    operation ordena_fichero(string nom_fich)
       //Pre:  el fichero en <nom_fich> existe
       //Post: el fichero en <nom_fich> está ordenado
       //Com: Asumimos que está implementada

          ...
    end operation

    //completar de acuerdo al enunciado del ejercicio
end
```