Variables compartidas

1. Para el siguiente programa concurrente suponga que todas las variables están inicializadas en 0 antes de empezar. Indique cual/es de las siguientes opciones son verdaderas:
a) En algún caso el valor de x al terminar el programa es 56. Verdadero
b) En algún caso el valor de x al terminar el programa es 22. Quizas(?
P1→ x=10. P3→Si en p3 se sobreescribe x*3 con el valor de x*2, queda 21 +p2= 22
c) En algún caso el valor de x al terminar el programa es 23. No creo

P1::	P2::	P3::

If (x = 0) then
y:= 4*2;
x:= y + 2;
	If (x > 0) then
	x:= x + 1;
		x:= (x*3) + (x*2) + 1;

- P1, p2, p3 ➔ y=8, x=10 //x=11//x=56

- Si hace p1 completo queda y=8, x=10

P3=x*3 ➔Load pos mem x, Mult * 3 (Ponele que a esto le llamamos a)
x*2➔ Load pos mem x, Mult * 2 (Ponele que a esto le llamamos b)
x*3+x*2+1➔Acum a + acum b + 1
Si la parte de x*3 se puede sobreescribir por x*2, x=20, +1=21
Se hace p2 y se lo incrementa en 1 quedando en 22.

- Si hace p1 completo queda y=8, x=10

P2= x=11
P3= x*3 ➔Load pos mem x, Mult * 3 (Ponele que a esto le llamamos a)
x*2➔ Load pos mem x, Mult * 2 (Ponele que a esto le llamamos b)
x*3+x*2+1➔Acum a + acum b + 1
Si la parte de x*3 se puede sobreescribir por x*2, x=22, +1=23

P1, p3, p2 ➔ y=8, x=10//x=51//x=52
P2, p1, p3 ➔ // y=8, x=10 // x=51
P2, p3, p1 ➔ //x=1//y=8, x=10
P3, p1, p2 ➔ x=1// - //x=2
P3, p2, p1 ➔ x=1 // x=2 //y=8, x=10

2. Realice una solución concurrente de grano grueso (utilizando <> y/o <await B; S>) para el siguiente problema. Dado un número N verifique cuántas veces aparece ese número en un arreglo de longitud M. Escriba las pre-condiciones que considere necesarias.
Precondición: Se necesita saber todo el tiempo cuánto vale en ese momento la variable total y hay tantos procesos como posiciones tiene el arreglo.

```
Int total=0;
a=array of int;
int n = read();
int M = a.length();

Process Calcular [id: 0..M-1]
        If (a[id]=n) then
                <total+=1>
```

======SI NO SABES LA CANTIDAD DE PROCESOS QUE TENES==========

```
Int total=0
A= array of int;
Int n= read ();                 //numero a buscar
Int m= a. length();             //long del array
Cantp=read ();
```

```
Posxproc=a.length()/cantp;      //porción de array para cada proceso

Process calcular [id 0..m-1]
        Int cant=0;             //contador parcial
        Inicio= id*posxpros;
        For i=inicio to (inicio+posxpros) do
                If (a[id]= n) then
                        Cant:=cant+1
        <total:=total+cant>
```

3. Dada la siguiente solución de grano grueso:
a) Indicar si el siguiente código funciona para resolver el problema de Productor/Consumidor con un buffer de tamaño N. En caso de no funcionar, debe hacer las modificaciones necesarias.

```
int cant = 0;      int pri_ocupada = 0;      int pri_vacia = 0;      int buffer[N];

Process Productor::                          Process Consumidor::
{ while (true)                               { while (true)
    { produce elemento                           { <await (cant > 0); cant-- >
      <await (cant < N); cant++>                   elemento = buffer[pri_ocupada];
      buffer[pri_vacia] = elemento;                pri_ocupada = (pri_ocupada + 1) mod N;
      pri_vacia = (pri_vacia + 1) mod N;           consume elemento
    }                                            }
}                                            }
```

El problema con esto es que cambia el valor de cantidad antes de dejar o tomar el elemento, por lo que el otro proceso pasa su await, y podría querer agarrar algo que no existe, o pisar algo que no haya sido consumido.

```
Int cant = 0;
Int pri_vacia = 0;
Int buffer [n];

Process Productor
        While (true)
                //Producir elemento//
                <await (cant<n);
                        Buffer [pri_vacia]=elemento;
                        Pri_vacia=(pri_vacia+1) mod n;
                        cant++>

Process Consumidor
        While (true)
                <await (cant>0);
                        Elemento= Buffer [pri_vacia-1];
                        Pri_vacia=(pri_vacia-1) mod n;
                        Cant-->
                        //Consumir elemento
======================OTRA OPCION DEJANDO PRI_VACIA Y PRI_OCUPADA=======================
Process productor
        While (true)
                //producir elemento
                <await (cant<n)>
                Buffer [pri_vacia]=elem
                Pri_vacia=(pri_vacia+1) mod n
                <cant ++>

Process consumidor
```

```
        While (true)
                <await (cant>0)>
                Elem=buffer [pri_ocupada]
                Pri_ocupada=(pri_ocupada+1) mod n
                <cant -- >
                //consumir elemento
```

b) Modificar el código para que funcione para C consumidores y P productores.

```
Int cant = 0;
Int pri_vacia = 0;
Int buffer [n];

Process Productor [id: 0..p]
        While (true)
                //Producir elemento//
                <await (cant<n);
                        Buffer [pri_vacia]=elemento;
                        Pri_vacia=(pri_vacia+1) mod n;
                        cant++>
        end;
end;
Process Consumidor [id: 0..c]
        While (true)
                <await (cant>0);
                        Elemento= Buffer [pri_vacia-1];
                        Pri_vacia=(pri_vacia-1) mod n;
                        Cant-->
                        //Consumir elemento
        End;
End;
```

4. Resolver con SENTENCIAS AWAIT (<> y <await B; S>). Un sistema operativo mantiene 5 instancias de un recurso almacenadas en una cola, cuando un proceso necesita usar una instancia del recurso la saca de la cola, la usa y cuando termina de usarla la vuelve a depositar.

```
colaEspecial C [5];

Process Coordinador
 { while (true)
    { <await (not empty (C)); //Espero a algun elemento encolado
   C.pop (recurso)>;    //Retira recurso de la cola
   //Usa_Recurso
   <C.push(recurso)>;     //Encola el recurso
    }
 }
```

5. En cada ítem debe realizar una solución concurrente de grano grueso (utilizando <> y/o <await B; S>) para el siguiente problema, teniendo en cuenta las condiciones indicadas en el item. Existen N personas que deben imprimir un trabajo cada una.

a) Implemente una solución suponiendo que existe una única impresora compartida por todas las personas, y las mismas la deben usar de a una persona a la vez, sin importar el orden. Existe una función Imprimir(documento) llamada por la persona que simula el uso de la impresora. Sólo se deben usar los procesos que representan a las Personas.

```
Boolean Desocupada:=True;

Process Persona [id: 0..n]
        <await (Desocupada ==True); Desocupada:= false>;
        //Imprimir (documento);
        Desocupada:=True;
End;
```

b) Modifique la solución de (a) para el caso en que se deba respetar el orden de llegada.

```
Int Siguiente=-1;
Cola C;

Process Persona [id: 0..n]
        <if (Siguiente==-1)  then          //si está desocupada y no hay cola, pasa
                Siguiente:= id;
        Else                               //si no, se pone en la cola
                Encolar (C, id)>;
        <await (Siguiente==id)>            //espera su turno
        //Imprimir (documento);
        <if (cola.estavacia()) then        //si no hay nadie en la cola
                Siguiente:=-1;
        Else
                Siguiente:=desencolar (c, id)>;  //si hay alguien en la cola
End;
```

c) Modifique la solución de (a) para el caso en que se deba respetar el orden dado por el identificador del proceso (cuando está libre la impresora, de los procesos que han solicitado su uso la debe usar el que tenga menor identificador).

```
Int Siguiente=-1;
Array a;

Process Persona [id: 0..n]
        <if (Siguiente==-1)  then          //si está desocupada y no hay nadie, pasa
                Siguiente:= id;
        Else                               //si no, se pone a esperar
                Agregar(A, id)>;
        <await (Siguiente==id)>            //espera su turno
        //Imprimir (documento);
        <if (array.estavacio()) then       //si no hay nadie esperando
                Siguiente:=-1;
        Else
                Siguiente:=min(a, id)>;  //si hay alguien esperando, busca al minimo de esos
End;
```

*Tambien puede hacerse con “Cola ordenada” y desencolar

d) Modifique la solución de (b) para el caso en que además hay un proceso Coordinador que le indica a cada persona que es su turno de usar la impresora.

```
Cola C;
Ocupada:= False;
Siguiente;

Process Persona [id: 0..n]
```

```
        <Encolar (C, id)>;                          //se pone en la cola
        <await (Siguiente==id); ocupada:=true>      //espera su turno.
        //Imprimir (documento);
        ocupada:=False;
End;

Process Coodinador;
        <await ((cola.noestavacia() and (ocupada=false) );
                Siguiente:=desencolar (c, id)>;             //llama a la persona
                Ocupada:=True;                      //No va protegida por que la usa solo una a la vez
End;
```

Robado de drive:

a)

```
bool ImpresoraLibre = True;

Process Persona[i:1..N]
 { <await (ImpresoraLibre);
   ImpresoraLibre:=false>;
   Imprimir(documento);
   ImpresoraLibre:=true;
 }
```

---

b)

```
ColaEspecial C;
int siguiente = -1;

Process Persona[i: 1..N]
 { <if (siguiente=-1) siguiente=i;
    else Agregar(C, i)>;
   <await (siguiente==i)>;
   Imprimir(documento);
   <if (empty(C)) siguiente:=-1
    else siguiente = Sacar(C)>;
 }
```

---

c)

```
ColaEspecialOrdenada C;
int siguiente=-1;

Process Persona[i:1..N]
 { <if (siguiente=-1) siguiente= i;
    else agregarOrdenado(C, i)>;
   <await (siguiente==i)>;
   Imprimir(documento);
   <if (empty(C)) siguiente:=-1
    else siguiente = sacar(C)>;
 }
```

---

d)

```
ColaEspecial C;
```

```
bool ImpresoraEnUso=True;
int siguiente=-1;

Process Persona[i:1..N]
 { agregar(C, i);
   <await (siguiente==i)>;
   Imprimir(documento);
   ImpresoraEnUso:=false;
 }

Process Coordinador[i:1]
 { while (true)
    { <await (not empty(C))>;
   ImpresoraEnUso=true;
      <siguiente = sacar(C)>;
   <await (ImpresoraEnUso==false)>
    }
 }

======OTRA VERSION DEL D
5)
b)
var siguiente = -1;
var cola = [];
Process Persona::[id:0..P]{
        <if(siguiente == -1); siguiente = id
        else cola.add(id)>;
        <await (siguiente == id)>
        imprimir();
        <siguiente = cola.length > 0 ? cola.pop() : -1>
}


c)
var siguiente = -1;
var cola = [];
Process Persona::[id:0..P]{
        <cola.add(id)>
        <if(siguiente == -1 && minimo(cola) == id); siguiente = cola.pop(id)>;
        <await (siguiente == id)>
        imprimir();
        <siguiente = cola.length > 0 ? cola.pop() : -1>
}

d)
var siguiente = -1;
var cola = [];
var salida = false;
Process Persona::[id:0..P]{
   <cola.add(id);>
   <await (siguiente == id);>
imprimir();
salida = true;
}
```

```
Process Coordinador:: {
   while true{
      <await cola.length != 0>
salida = false;
      <siguiente = cola.pop()>
<await salida == true>
}
}
```

6. Dada la siguiente solución para el Problema de la Sección Crítica entre dos procesos (suponiendo que tanto SC como SNC son segmentos de código finitos, es decir que terminan en algún momento), indicar si cumple con las 4 condiciones requeridas:

```
int turno = 1;
```

Process SC1::	**Process SC2::**

{ while (true)
{ while (turno == 2) skip;
SC;
turno = 2;
SNC;
}
}
	{ while (true)
	{ while (turno == 1) skip;
	SC;
	turno = 1;
	SNC;
	}
	}

Exclusión mutua: A lo sumo un proceso está en su SC

SC1 y SC2 se excluyen en el acceso a la SC, ya que comparten una única variable que lo controla. Para entrar a la SC, cada proceso se debe esperar que "turno" tome un valor específico, y esa variable no puede tener dos valores a la vez. El cambio de valor recién se da luego de pasar la sección crítica, por lo que no hay chance de que el otro proceso pueda iniciarse si el que está ejecutando no terminó.

Ausencia de Deadlock (Livelock): si 2 o más procesos tratan de entrar a sus SC, al menos uno tendrá éxito.

Para que haya deadlock ambos procesos deberían estar esperando entrar a la SC sin éxito, lo que no es posible ya que esto se controla mediante la variable "Turno" que inicia con el valor 1, y solo puede tomar este o 2, ambos válidos.

Ausencia de Demora Innecesaria: si un proceso trata de entrar a su SC y los otros están en sus SNC o terminaron, el primero no está impedido de entrar a su SC.

Si un proceso termina de ejecutar su SC, la instrucción siguiente es establecer el valor de turno, dando acceso a otro proceso a dicha sección antes de pasar a ejecutar su SNC. El error se encuentra en que ambos procesos hacen mal el cambio de valor de la variable turno, manteniendo siempre el valor con el que inició el programa.

Eventual Entrada: un proceso que intenta entrar a su SC tiene posibilidades de hacerlo (eventualmente lo hará).

En este caso no se cumple la propiedad. Al iniciar va a comenzar a ejecutarse SC1, ya que la variable "turno" inicia en 1. Al finalizar la ejecución de su SC, este proceso asigna un valor a "turno", pero en lugar de darle un 2, permitiendo que el otro proceso se ejecute, la mantiene en 1, por lo que, aunque no esté ejecutando su SC, el proceso SC2 nunca podrá ejecutarse.

7. Desarrolle una solución de grano fino usando sólo variables compartidas (no se puede usar las sentencias await ni funciones especiales como TS o FA). En base a lo visto en la clase 3 de teoría, resuelva el problema de acceso a sección crítica usando un proceso coordinador.
En este caso, cuando un proceso SC[i] quiere entrar a su sección crítica le avisa al coordinador, y espera a que éste le dé permiso. Al terminar de ejecutar su sección crítica, el proceso SC[i] le avisa al coordinador.
Nota: puede basarse en la solución para implementar barreras con "Flags y coordinador" vista en la teoría

```
Int arribo [1:n] = ([n] 0)
Int continuar [1:n] = ([n] 0)
```

```
Process SC1 [i=1 to n]
        While (true)
                Arribo [i] =1                   //marca en su pos que llego y esta esperando SC
                While (continuar[i] ==0) then:  //espera el acceso
                        skip;
                //sección critica
                continuar [i] =0;               //avisa que termino
        End;
End;

Process coordinador
        While (true)
                For (i=1 to n) do
                        If (arribo [i] ==1) then                    //si esta esperando
                                arribo [i] =0;                      //desmarca para la proxima
                                Continuar[i] =1;                    //lo deja pasar
                                While (Continuar[i] ==1) then       //espera que termine
                                        Skip;
                End;
        End;
End;

int Arreglo [0:N];
bool termino = False;

Process SC [id: 0..N]
 { Arreglo[id] = true;
   while (Arreglo[id]) do skip;
   //Seccion_Critica;
   termino = true;
   //Fin_Seccion_Critica;
 }

Process Coordinador
 { while (true) do
    { for (int i=0 to N)
    { if (lista[i] == true)
      lista[i] = false;
      while (termino == false) skip;
      termino = false;
      }
     }
 }
=======OTRA SOLUCION

int Arreglo [0:N];
bool termino = False;

Process SC [id: 0..N]
 { Arreglo[id] = true;
   while (Arreglo[id]) do skip;
   //Seccion_Critica;
   termino = true;
   //Fin_Seccion_Critica;
```

```
 }

Process Coordinador
 { while (true) do
    { for (int i=0 to N)
           { if (lista[i] == true)
                  lista[i] = false;
                  while (termino == false) skip;
                  termino = false;
            }
     }
 }

===============OTRA
var lista = [0..P];
var termino = false;

Process PSC [id: 0..P]{
          lista[id] = true;
          while (lista[id]) continue;
          SC();
          termino = true;
          NSC();
}

Process Coordinador{
          while (true){
                    for(int i=0 to p){
                              if(lista[i] == true) {
                                        lista[i] = false;
                                        while(termino == false) continue;
                                        termino = false;
                              }
                    }
          }
}
```