




Practica 1

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.
 - b) En algún caso el valor de x al terminar el programa es 22.
 - c) En algún caso el valor de x al terminar el programa es 23.

P1:: If (x = 0) then y:= 4*2; x:= y + 2;	P2:: If (x > 0) then x:= x + 1;	P3:: x:= (x*3) + (x*2) + 1;
--	--	---------------------------------------

1. Opcion A: p1.1, p1.2, p1.3, p2.1, p2.2, p3.1 → x = 56 
- Opcion B: p1.1, p1.2, p1.3, p3.1, p2.1, p2.2 → x = 52
- Opcion C: p1.1, p3.1, p2.1, p2.2, p1.2, p1.3 → x = 10
- Opcion D: p1.1, p3.1, p2.1, p1.2, p2.2, p1.3 → x = 10
- Opcion E: p1.1, p3.1, p2.1, p1.2, p1.3, p2.2 → x = 11
- Opcion F: p3.1, p2.1, p2.2 → x = 2
- Opcion G: p1.1, p1.2, p3.1.1, p1.3, p3.1.2, p3.1.3, p2.1, p2.2 = 22 
- Opcion H: p1.1, p1.2, p3.1.1, p1.3, p2.1, p2.2, p3.1.2, p3.1.3 = 23 

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.

Precondiciones: el tamaño del arreglo y el N son multiplo de 2

```
int N;  
int M;  
int p;  
int total = 0 //Compartida  
int array [] 1..M  
int porcion = M/p
```

```

process buscar [id = 0 to p - 1]{
    int cant = 0;
    for (i = id* porcion; (id * porcion) + porcion; i++){
        if (array[i] == N)
            cant++
    }
    <total += cant;>
}

```

3.

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

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

- a. Es necesario que se tome en productor como operacion atomica <await (cant < N); cant++; buffer[pri_vacia] = elemento> y en el proceso del consumidor <await (cant > 0); cant - -; buffer[pri_ocupada] = (pri_ocupada + 1)>

int cant = 0; int pri_ocupada = 0	int pri_vacia = 0; int buffer[N];
Process Productor [int p = 0 to P-1] { while (true) { // produce elemento <await (cant < N); cant++ buffer[pri_vacia] = elemento> pri_vacia = (pri_vacia + 1) mod N; } }	Process Consumidor[int c= 0 to C-1] { while (true) { <await (cant > 0); cant-- elemento = buffer[pri_ocupada];> pri_ocupada = (pri_ocupada + 1) mod N; } }

<pre> } } </pre>	<pre> // consume elemento } } </pre>
------------------	--------------------------------------

4.

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.

```

Recurso cola [] 1..5; int cant = 0;
process accesos [id = 0 to N]{
    Recurso recurso;
    while (true){
        <AWAIT cant < 5; cant++; recurso = cola.pop()>
        // Utilizo recurso
        <cant--; cola.push(recurso)>
    }
}

```

5.

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 ítem. Existen N personas que deben imprimir un trabajo cada una.
- 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*.
 - Modifique la solución de (a) para el caso en que se deba respetar el orden de llegada.
 - 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).
 - 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.

a.

```

process imprimir [p = 0 to n]{

```

```
<imprimir(documento)>
}
```

b.

Precondicion: tenemos dos metodos para cola, uno que agrega al final de la cola y otro que obtiene desde el principio de la cola.

```
int siguiente = -1; Cola c;
Impresora impresora;
process imprimir [id = 0 to N-1]{
  <if (siguiente == -1) siguiente = id;
  else c.agregar(c, id)>;

  <AWAIT siguiente == id;>
  impresora.imprimir(documento)
  <if (c.isEmpty()) siguiente = -1;
  else siguiente = c.sacar();>
}
```

c.

Precondicion: existe un metodo para list que nos devuelve el minimo de la lista.

```
int siguiente;
List listaEspera;
Impresora impresora;
process persona[int id = 0 to N-1] {
  while (true){
    //anotarse a lista de espera
    <listaEspera.add(id)>
    <AWAIT id == listaEspera.min(id);
    impresora.imprimir(documento);
    listaEspera.remove(id);>
  }
}
```

d.

Precondicion: tenemos dos metodos para cola, uno que agrega al final de la cola y otro que obtiene desde el principio de la cola.

```
int siguiente; Cola c; listo = false;
Impresora impresora
process coordinador {
    while true{
        //tomar el siguiente
        <AWAIT !c.isEmpty(); siguiente = cola.sacar()>
        // esperar el uso de la impresora y recibir mensaje de finalizacion
        <AWAIT listo; listo = false>
    }
}

process persona [int id = 0 to N-1]{
    while (true){
        //encolarme
        <c.agregar(id)>
        //esperar mi turno
        <AWAIT siguiente == id; impresora.imprimir(documento)
        listo = true;
        >
    }
}
```

6.

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:

<pre>int turno = 1; Process SC1:: { while (true) { while (turno == 2) skip; SC; turno = 2; SNC; } }</pre>	<pre>Process SC2:: { while (true) { while (turno == 1) skip; SC; turno = 1; SNC; } }</pre>
--	---

NO, no cumple con las 4 condiciones requeridas ya que uno si o si estara ejecutando la SC, no se producen deadlocks ya que una vez que uno sale de la SC le da el paso al otro. Eventualmente un proceso va a entrar a la SC tarde o temprano por lo que no hay inanición, pero existe demora innecesaria ya que se fuerza necesariamente a que cuando uno termina su sesion critica, el otro se ejecute pero si el porcesador le vuelve a asignar el procesador, no tiene el turno.

7.

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 3.

```
int acceso [0..N -1]; int llegue [0..N -1];
process proceso [int id = 0 to N-1]{
  while true{
    //aviso al cordinador que quiero entrar
    llegue [id] = 1;
    // esperar que me otorgue permiso
    while (acceso [id] == 0) skip
    //entrar a SC
    SC
    // avisar al cordinador que finalice
    llegue [id] = 0;
```

SNC

```
}  
}  
  
process coordinador{  
  while true {  
    for (int i = 0 to N-1){  
      //recibe solicitud  
      if (llegue[i] == 1){  
        //otorga permiso  
        acceso [i] = 1;  
        while (llegue [i] == 1) skip  
        //recibe señal de finalizacion  
        acceso [i] = 0;  
      }  
    }  
  }  
}
```