Resoluciones P1 - Variables compartidas

2. Realice una solución concurrente de grano grueso (utilizando <> y/o) 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.

   Respuesta :

```
`int N ; int M ; arreglo[M] ; int Total = 0;`

    `Process recorrer[id 0 .. P-1]{`
        `int aux  = 0`
        `int comienzo = (M/P) * id`
        `int fin = comienzo + (M/P)`

        `for ( int i = comienzo , i < fin , i++ ){`

            `if (arreglo[i] == N ){`
                    `aux ++`
            `}`
        `}`

        `<total += aux>`

        `}`
```

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

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

   Respuesta :

📌 La idea general

Tenemos **dos tipos de procesos concurrentes**:

1. **Productores** → generan datos (elementos) y los guardan en un **buffer compartido** (una especie de cola).
2. **Consumidores** → toman los datos del buffer para procesarlos.
   🔗 Ambos se comunican **solo a través del buffer**.

📌 El buffer compartido

- Es una memoria intermedia de tamaño finito **N**.
- Se usa como una cola circular:
  - `pri_vacia` → índice donde se guarda el próximo elemento (cola).
  - `pri_ocupada` → índice donde se saca el próximo elemento (cabeza).
  - `cant` → cantidad actual de elementos en el buffer.

4. Resolver con SENTENCIAS AWAIT (<> y). 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.

   Respuesta :

```
Inicializar colaEspecial C con 5 instancias de recurso;

process consumidor[id: 0..n]{

   while (true){
       recurso instancia;
       <await (not c.isEmpty()) ; instancia = c.pop() >
       hace uso del recurso
       < c.push(instancia) >
   }
}
```

5. En cada ítem debe realizar una solución concurrente de grano grueso (utilizando <> y/o) para el siguiente problema, teniendo en cuenta las condiciones indicadas en el ítem.
   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.
   b) Modifique la solución de (a) para el caso en que se deba respetar el orden de llegada.

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).
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.

Respuesta :

```
a.

boolean libre = true

process persona[id: 0..N-1]{


     < await (not libre); libre = false >
      Imprimir(documento) > ???? -- atomico
      < libre = true >

}

--> Tamb se puede hacer <await (not libre); libre = false>
```

```
b.

ColaLlegada c;
int sig = -1

process persona[id: 0..N-1]{

     < if (sig == -1) then sig = id
       else c.add(id)>
     < await (sig == id) >

     imprimir(documento)

     < if (c.isEmpty()) then sig = -1
      else sig = c.pop() >

}
}

-> Mi duda acá es cuando me desencolo yo ? sera despues del await ?
```

```
    -> Acá no hace falta el uso de una variable de ocupado ? ya que al usarlo
       de a uno la impresora lo controlo con la variable sig.
```

```
    c.

    ColaLlegada c;
    int sig = -1

    process persona[id: 0..N-1]{

         < if (sig == -1) then sig = id
           else c.addOrdenado(id)>
         < await (sig == id) >

         imprimir(documento)

         < if (c.isEmpty()) then sig = -1
          else sig = c.pop() >

    }
    }

    ->  Lo que hice fue solo que se inserta ordenado en la cola, despues lo
trate
        igual que el anteriror.
```

```
    d. TERMINAR

    ColaLlegada c;
    int sig = -1;
    boolean enUso = false;

    process coordinador{

        while (true){

            < await (not enUso) ;
                if ( not c.isEmpty() ) ;


        }

    }
```

```
process persona[id: 0..N-1]{

      < if (sig == -1) then sig = id
        else c.add(id)>
      < await (sig == id) >

      imprimir(documento)

      < if (c.isEmpty()) then sig = -1
       else sig = c.pop() >

}
}
```

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::
{ while (true)
{ while (turno == 2) skip;
SC;
turno = 2;
SNC;
}
}
	Process SC2::
	{ while (true)
	{ while (turno == 1) skip;
	SC;
	turno = 1;
	SNC;
	}
	}

Respuesta :

- Exclusión mutua : Se cumple ya que
- Ausencia de deadlock : Esto se cumple ya que el while se encarga de que alguno de los dos procesos entren a su sección critica mediante la variable turno.
- Ausencia de demora innecesaria : No cumple ya que antes de entrar a su sección no critica se encarga de cambiar el valor de la variable para que el otro proceso pueda entrar a su sección critica. (No se cumple si turno esta después de SNC). (Al arrancar el programa esta el problema, de que si o si tiene que arrancar el el 1).
- Eventual entrada : Si en este caso son dos procesos que se cambian mutuamente al momento de finalizar uno.

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.