


# Practica 1

## Ejercicio 1

p11, p12, p13, p21, p22, p31 = 56 

p11, p12, p13, p31, p21, p22 = 52

p11, p31, p21, p22, p12, p13 = 10

p11, p31, p21, p12, p22, p13 = 10

p11, p31, p21, p12, p13, p22 = 11

p31, p21, p22 = 2

## Ejercicio 2

```
//Precondiciones el tamaño del arreglo es multiplo 2
```

```
int v[1,m]; int n; int cant = 0; int porcion = m/2  
int m;
```

```
Process buscar()[id = 1 to 2]:
```

```
  if id = 1 {  
    for i = 1 to porcion {  
      if v[i] = n  
        <cant++>  
    } else  
      for j = porcion+1 to m  
        if v[i] = n  
          <cant++>  
  }
```

```
//Precondiciones el tamaño del arreglo y la cantidad de procesos es multiplo  
2
```

```
int v[1,m]; int n; int total = 0; int porcion = m/p
int m; int p
```

```
Process buscar()[id = 0 to p-1]:
    int cant = 0;
    for (i=id*porcion; i<(id*porcion)+porcion; i++) {
        if (v[i] = n)
            cant++;
    }
    <total += cant>
}
```

3a\_ No funcionan porque puede pasar el caso en el que acceden a posiciones que no deben acceder y el productor aumenta la cantidad antes de asignar el elemento.

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

b\_

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

## Ejercicio 4

```
cola recursos = [1,5]
```

```
Process usarInstancia[id = 1..n]:  
    while(true) {  
        <await (not empty(recursos));  
        r = pop(recursos)>  
        usar(r);  
        <push(recursos, r)>  
    }  
}
```

## Ejercicio 5

a\_

```
Process persona(id=1..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;  
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\_

```

list esperando;

Process persona(id=1..n) {
  while(true) {
    <esperando.add(id)>
    <await (id == min(esperando));
    imprimir(documento);
    esperando.remove(id);>
  }
}

```

d\_

```

cola turnos = empty;
int autorizado = -1; //nadie esta autorizado

Process persona(id=1..n) {
  while(true) {
    <push(turnos, id)>
    <await (id == autorizado);
    imprimir(documento);
    pop(turnos);
    autorizado = -1;
  }
}

Process coordinador() {
  while(true) {
    <await (not empty(turnos));
    autorizado = front(turnos);
  }
}

```

```
>  
<await (autorizado == -1)>  
}  
}
```

## 6\_ Verificación de las 4 condiciones

### 1. Exclusión mutua

- Nunca pueden entrar los dos procesos a la vez, porque el turno solo vale 1 o 2.
- ✓ Cumple.

### 2. Ausencia de deadlock

- Siempre alguno puede entrar: si `turno = 1`, entra P1; si `turno = 2`, entra P2.
- No hay posibilidad de que ambos estén esperando eternamente.
- ✓ Cumple.

### 3. Ausencia de demora innecesaria

- Cuando un proceso entra y termina, si quiere volver a entrar no puede porque tiene que esperar forzosamente a que el otro proceso ingrese y termine.
- ✗ No cumple

### 4. Eventual entrada

- Si un proceso quiere entrar, eventualmente lo hará, pero **solo alternando**.
- (suponiendo que tanto SC como SNC son segmentos de código finitos, es decir que terminan en algún momento). Como terminan en algun momento, cualquiera de los dos procesos va a tener una eventual entrada, no se quedara esperando infinitamente.
- ✓ Cumple.

## 7\_

```

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
        SNC
        // avisar al cordinador que finalice
        llegue [id] = 0;
    }
}

```

```

process cordinador{
    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;
            }
        }
    }
}

```