Concurrente — Práctica 1

X=0 Y=0

1) 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;
     9)     10)   11)
```

a) 56 ✓ Acciones atómicas de grado fino

1) Store 4*2, Reg Acumulador,
2) Store Reg Acumulador, Pos Memoria Y
3) Load Pos Memoria y, Reg Acumulador
4) Add 2, Reg Acumulador
5) Store Reg Acumulador, Pos Memoria X

6) Load Pos Memoria X, Reg Acumulador
7) Add 1, Reg Acumulador
8) Store Reg Acumulador, Pos Memoria X
9) 10) y 11) Acciones atómicas de P3

56 es verdadero si los procesos se ejecutan en forma secuencial.

b) 1, 2, 9, 3, 4, 5, 10, 11, 6, 7, 8 Verdadero 22 ✓

c) 1, 2, 9, 3, 4, 5, 6, 7, 8, 10, 11 Verdadero 23 ✓

NOTA

2) Precondiciones:
- El arreglo debe estar correctamente definido y no ser nulo
- El valor de M debe ser > a 4
- N debe ser un valor válido (numérico)
- El número de procesos debe ser menor o igual a M

```
int total=0;   int arreglo[M]=...;   int N=...;

Process recorrer[id:0...3]
{ int inicio= id * 4;
  int fin = (id + 1) * 4;
  int parcial = 0;
  for i=inicio to fin -1
    { if (arreglo[i] =N) then
        parcial=parcial + 1;
    }
    <total = total + parcial;>
}
```

3) a)
El código no funciona, ya que, el acceso al buffer (recurso compartido) no se hace con exclusión mutua, lo que podría generar que el Productor y el consumidor intenten acceder al buffer al mismo tiempo. Además, un Consumidor podría intentar consumir un elemento del buffer antes que el Productor ingrese uno. [(pri_ocupada y pri_vacia) podrían ser var. locales]

```
Process Productor::
{ :

  <await (cant<N); cant++;
    buffer[pri_vacia]=elemento;>
  :
}
```

```
Process Consumidor::
{ :
  <await (cant>0); cant--;
   buffer[pri_ocupada];>
  :
}
```

b) Para que funcione para C consumidores y P productores:

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

Process Productor [0..P-1]
{ while (true)
  { produce elemento
    <await (cant<N); cant++;
     buffer[pri_vacia]=elemento;
     pri_vacia=(pri_vacia+1) mod N;>
  }
}

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

En este caso pri_ocupada y pri_vacia DEBEN ser var. globales o compartidas.

4) int cant = 5; recurso Q[5]; (cola de recursos)

```
process P [i:0...N]
{ while (true)
  { <await (cant > 0); cant--;
      r = pop (Q, recurso) >
    //usa recurso
    < push (Q, r); cant++; >
  }
```

< await (not empty (Q)) .... >
podr. cambiar preg. not empty Q ?

5) a) boolean libre = true;

```
Process Personas [id: 0 . N-1]
{ while (true)
  { <await (libre); libre = false; >
      Imprimir (documento);
      libre = true;
  }
}
```

imprime
1 sola vez c/ persona?

b) int siguiente = -1; colaEspecial C;

```
process Personas [id: 0..N-1]
{
  { < IF (siguiente = -1) siguiente = id
    else push (C, id) >;
    <await (siguiente = id) >;
    // Imprimir (documento);
    < IF (empty (C)) siguiente = -1
      else siguiente = pop (C, id) >;
  }
}
```

NOTA Si 1 persona imprime más de 1 documento, sg while (true)

c) igual que la solución b pero en vez de usar un push (c, id) usa un
Agregar Ordenado (c, id)

el pop(c, id) queda igual xq la cola ya fue ordenada en base a su id al Agregar Ordenado (c, id).

d) int siguiente = -1;

(termina siendo secuencial)

```
Process Personas[id=0..N-1]
  {
    { <await (siguiente = id-1)>
        Imprimir (documento);
        siguiente = id;
    }
  }
```

consultar

e) colaEspecial c; boolean libre=true; int siguiente=-1;

```
process Personas[id:0..N-1]
{ < push(c, id); >
  <await (siguiente=id)>
    Imprimir (documento);
    libre = true;
}
```

```
Process Cordinador
{ For i=0..N-1
  { <await notEmpty(c)>;
    <await (true)>
      libre = false;
      siguiente = pop(c, id);
  }
}
```

NOTA

6) int cant=0; colaEspecial C; colaResultados R;

process Alumnos [id: 0..P-1]
{ <cant = cant + 1>
<await (cant = P)>
Resuelve examen
<entrega examen
push(C, id); >

process Profesores [id: 0..2]
{ <await notEmpty(C);
id = pop(C, id);
Agarra el examen >
corrige el examen
<

¿Cómo le aviso al alumno que su examen ya está corregido y le doy la nota?
puedo tener un vector de 0 a 2 donde los profesores dicen la nota
y en alumnos tener:
<await (V[0] = id) OR (V[1] = id) OR (V[2] = id) >
recibir nota ?

¿Cómo le aviso al profesor que me fui así le da la nota al siguiente?

NOTA

6) int cant=0; cola Especial c; double notas[P-1]; // inicializadas en -1
int siguiente = -1; int restantes = P;

```
process Alumnos[id:0..P-1]
{ <cant = cant + 1>
  // espera a que llegen todos
  <await (cant = P)>
  // resuelve examen
  <// entrega examen
    push(c, id);>
  // espera mi nota
  <await (notas[id] <> -1)>
}
```

```
process Profesores[id:0..2]
{ while (restantes <> 0)
  { boolean corregir = False;
    // espera a que entreguen
    <await (not empty(c))
      or (restantes == 0))
    if (restantes <> 0)
    { siguiente = pop(c, id);
      restantes = restantes - 1;
      corregir = true; } >
    if (corregir == true) {
      // corregir
      // entregar examen
      notas[siguiente] = nota;
    }
  }
}
```

7) Condiciones requeridas:
1 - Exclusión mutua
2 - Ausencia de Deadlock
3 - Ausencia de Demora Innecesaria ✗
4 - Eventual Entrada (sin inanición) ✗

3) No se cumple con la propiedad de Ausencia de Demora Innecesaria, ya que, puede darse el escenario donde el SC1 esté listo para ejecutar la SC y a su vez el proceso SC2 no la necesite pero de todos modos siempre el proceso SC1 debe esperar a que el proceso SC2 cambie la variable turno, (o al revés). No hay coordinación entre la intención de entrar, pudiendo generar demora innecesaria.

4) Si uno de los procesos es mucho más rápido que el otro y el S.O. operativo SIEMPRE le da prioridad, puede ser que no se garantice la eventual entrada del otro proceso.
[No hay manera de expresar intención, ni ver si el otro realmente necesita la SC. Podría quedarse esperando para SIEMPRE.]

Corrección: la eventual entrada sí se cumple.
La única condición que no se cumple es la Ausencia de Demora Innecesaria!

NOTA

8) int N=...; int actual = -1; bool peticiones[N]; // inicializados en false

```
Process SC [i=0..N-1]
{ while (true)
  { peticiones[i] = true;
    while (actual <> i) skip;
      // Sección crítica
      actual = -1;
  }
}

Process Cordinador
{ while (true)
  { for (i=0 to N-1)
    { if (peticiones[i])
      { peticiones[i] = false;
        actual = i;
        while (actual <> -1) skip;
      }
    }
  }
}
```