Práctica 1 – 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.
   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::	**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;

3. Dada la siguiente solución de grano grueso, 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
	}
	}

2. Realice una solución concurrente de grano grueso (utilizando <> y/o <await B; S>) para el siguiente problema. Dado un numero N verifique cuantas veces aparece ese número en un arreglo de longitud M. Escriba las condiciones que considere necesarias.

4. Realice una solución concurrente de grano grueso (utilizando <> y/o <await B; S>) para el siguiente problema. 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.

5. Realice una solución concurrente de grano grueso (utilizando <> y/o <await B; S>) el siguiente problema. Se debe calcular el valor promedio de un vector de 1000 números por medio de 10 procesos. Al finalizar todos los procesos deben guardar en una variable local el resultado final.

6. 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*.

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

c) Modifique la solución de (b) para el caso en que se deba respetar el orden de llegada pero dando prioridad de acuerdo a la edad de cada persona (cuando la impresora está libre la debe usar la persona de mayor edad entre las que hayan solicitado su uso).

d) Modifique la solución de (a) para el caso en que se deba respetar estrictamente el orden dado por el identificador del proceso (la persona X no puede usar la impresora hasta que no haya terminado de usarla la persona X-1).

e) Modifique la solución de (c) para el caso en que además hay un proceso *Coordinador* que le indica a cada persona cuando puede usar la impresora.

7. Realice una solución concurrente de grano grueso (utilizando <> y/o <await B; S>) el siguiente problema. Una empresa de turismo posee UN micro con capacidad para 50 personas. Hay un único vendedor donde C pasajeros (C > 50) intentan comprar un pasaje (de acuerdo al orden de llegada); si aún hay lugar se dirige al micro para subir; en caso contrario se retira. El micro espera a que suban los 50 pasajeros (suben de a uno y sin importar el orden), luego realiza el viaje, y cuando llega al destino deja bajar a todos los pasajeros.

8. Realice una solución concurrente de grano grueso (utilizando <> y/o <await B; S>) el siguiente problema. En una playa hay 5 personas que deben juntar 15 monedas cada una y obtener el TOTAL de dinero juntado entre todos (la suma del valor de las 75 monedas que pueden ser de 1, 2 o 5 pesos). Al terminar todos los procesos deben guardarse en una variable local el TOTAL. ***Nota:*** maximizar la concurrencia. Suponga que existe una función *Moneda()* llamada por las personas que simula encontrar UNA moneda y retorna el valor de la misma. Cada persona trabaja sobre una zona diferente de la playa.

9. Realice una solución concurrente de grano grueso (utilizando <> y/o <await B; S>) el siguiente problema. En un entrenamiento de futbol hay 25 jugadores y un entrenador. Cuando los 25 jugadores han llegado el entrenador le da una charla técnica a cada uno de ellos (de a uno a la vez) en orden aleatorio (es decir que cuando el entrenador está libre elige a cualquiera de los jugadores con los que aún no habló). Cuando un jugador termino su charla técnica con el entrenador, corre durante 30 minutos y luego se retira. El entrenador se retira después de hablar con los 25 jugadores. ***Nota:*** maximizar concurrencia; suponga que existe una función *Siguiente* llamada por el entrenador que le devuelve un número aleatorio que indica el ID de un jugador con el que aún no hablo.

10. Realice una solución concurrente de grano grueso (utilizando <> y/o <await B; S>) el siguiente problema. En un examen de la secundaria hay ***un preceptor*** y ***una profesora*** que deben tomar un examen escrito a ***45 alumnos***. El preceptor se encarga de darle el enunciado

del examen a los alumnos cundo los 45 han llegado (es el mismo enunciado para todos). La profesora se encarga de ir corrigiendo los exámenes de acuerdo al orden en que los alumnos van entregando. Cada alumno al llegar espera a que le den el enunciado, resuelve el examen, y al terminar lo deja para que la profesora lo corrija y le dé la nota. ***Nota:*** maximizar la concurrencia; todos los procesos deben terminar; suponga que la profesora tiene una función *corregirExamen* que recibe un examen y devuelve un entero con la nota.

11. Realice una solución concurrente de grano grueso (utilizando <> y/o <await B; S>) el siguiente problema. En un examen final hay ***P alumnos*** y ***3 profesores***. Cuando todos los alumnos han llegado comienza el examen. Cada alumno resuelve su examen, lo entrega y espera a que alguno de los profesores lo corrija y le indique la nota. Los profesores corrigen los exámenes respectando el orden en que los alumnos van entregando.

12. En una salita médica hay ***un médico*** para atender a ***15 pacientes***. El médico atiende a los pacientes para indicarles el tratamiento a realizar; asegurándose de que no haya dos pacientes al mismo tiempo en el consultorio. **Nota:** todos los procesos deben terminar; sólo se pueden usar procesos que representen a los pacientes y al médico.
    a) Realice una solución concurrente de grano grueso (utilizando <> y/o <await B; S>) teniendo en cuenta que el médico atiende a los pacientes de acuerdo con el orden en que van llegando.
    b) Realice una solución concurrente de grano grueso (utilizando <> y/o <await B; S>) teniendo en cuenta que el médico atiende a los pacientes de acuerdo al turno de cada paciente (suponga que cada paciente ya conoce su turno).

13. Dada la siguiente solución de grano fino 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;
	  }
	}

14. 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 2 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 de *Variables Compartidas*.