Trabajo Práctico N°1

1.- Dadas las siguientes propuestas de pseudocódigos de solución al problema de la sección crítica, analice detalladamente si cumplen con cada una de las condiciones para que sean solución.

a)

```
int turno = 1;  // corresponde ingresar a P1.
P0:                                          P1:
   while(true)                                  while(true)
   {                                            {
       while(turno = 1);                            while(turno = 0);
       /*SeccCritica_0*/;                           /*SeccCritica_1*/;
       turno = 1;                                   turno = 0;
       /*programa0*/;                               /*programa1*/;
   }                                            }
```

b)

```
boolean estado[2] = {false, false}; // valor inicial de variables compartidas.
P0:                                          P1:
   while(true)                                  while(true)
   {                                            {
       estado[0] = true;                            estado[1] = true;
       while(estado[1]);                            while(estado[0]);
       /*SeccCritica_0*/;                          /*SeccCritica_1*/;
       estado[0] = false;                           estado[1] = false;
       /*programa0*/;                               /*programa1*/
   }                                            }
```

c)

```
boolean estado[2] = {false, false}; // valor inicial de variables compartidas.
P0:                                          P1:
   while(true)                                  while(true)
   {                                            {
       estado[0] = true;                            estado[1] = true;
       if(estado[1])                                if(estado[0])
          estado[0] = false;                           estado[1] = false;
       else                                         else
       {                                            {
          /*SeccCritica_0*/;                          /*SeccCritica_1*/;
          estado[0] = false;                         estado[1] = false;
          /*programa0*/;                             /*programa1*/;
       }                                            }
   }                                            }
```

2.- Considere un sistema operativo en cual se encuentran en ejecución dos procesos concurrentes, P1 y P2, que intentan entrar en su sección crítica la cual, en ambos procesos, tiene una cantidad de tres instrucciones. Luego de finalizar la sección crítica, el proceso P1 ejecuta en su programa siete instrucciones y P2, cinco. Se indican en la tabla a continuación los instantes en que los procesos invocan a la primitiva P(s) al intentar ingresar en su sección crítica.
Utilizando las implementaciones de las primitivas P y V que se indican, complete la tabla indicando los valores que toman en cada paso las variables ***mutex***, ***delay*** y ***s***, e indique en cada caso qué proceso modifica éstas variables y cuál de ellos queda en espera. Complete, además, el código que ejecuta cada proceso, asumiendo que cada instrucción se ejecuta en un sólo paso, que la primitiva P tiene prioridad sobre la V, y que las primitivas tienen prioridad frente a las otras instrucciones. Para indicar código de sección crítica utilice SC1/SC2, y para programa, Prog1/Prog2.

Operación P(s):

```
Pb(mutex);
s = s - 1;
if s < 0 then
      Vb(mutex);
      Pb(delay);
Vb(mutex);
```

Operación V(s):

```
Pb(mutex);
s = s + 1;
if s <= 0 then
      Vb(delay);
else
      Vb(mutex);
```

Paso	Proceso 1	Proceso 2	mutex	delay	s	Espera
0			True	False	1	
1	P(s)					
2						
3		P(s)				
4						
5						
…	…	…	…	…	…	…

3.- Resolver con el mismo criterio del ejercicio anterior, pero para tres procesos, P1, P2 y P3, donde todos tienen tres instrucciones en la sección crítica, P1 tiene seis instrucciones de programa, P2 tiene cuatro y P3 tiene cinco.

Paso	Proceso 1	Proceso 2	Proceso 3	mutex	delay	s	Espera
0				True	False	1	
1	P(s)						
2							
3							
4							
5							
6			P(s)				
7							
8							
9							
10							
11							
12							
13		P(s)					
14							
…	…	…	…	…	…	…	…

4.- Considere la solución propuesta en el ejemplo del buffer limitado visto en teoría. Asuma además que cada instrucción o primitiva se ejecuta en un instante de tiempo, y que las primitivas tienen prioridad frente a las otras instrucciones. Teniendo en cuenta que para el proceso productor la generación del próximo registro y el agregado en el buffer requiere en ambos casos una instrucción mientras que, en el proceso consumidor, extraer un registro del buffer requiere una instrucción y procesar el registro extraído, tres instrucciones, realice las siguientes consignas:

a) Identifique las rutinas *Comienzo_SC* y *Fin_SC*.
b) Identifique la sección crítica de cada proceso, productor y consumidor.
c) Complete las tablas a continuación, indicando los valores que toman los semáforos en cada paso, qué proceso modifica cada uno de ellos y cuál de ellos se queda en espera. Determine si se producen errores por buffer desbordado o buffer vacío.

Paso	Proceso 1	Proceso 2	e	f	s	Espera
0			3	0	1	
1	Productor					
2						
3						
4						
5						
6						
7		Consumidor				
8	Productor					
9						
10						
11						
12						
13						
14						
15						
16		Consumidor				
17						
18	…	…	…	…	…	…

5.- Dada la implementación de la primitiva V(s) sin espera ocupada:

```
V(s):
        Pb(mutex);
        s.value = s.value + 1;
        if s <=0 then       //activar un proceso.
           q = delete(s.L); //sacar proceso de cola asociada a s.
           if (hay CPU libre) then //apropiado para SMP.
              comenzar a ejecutar q en un CPU libre;
           else
              insert(q, Cola_Ready); //si CPU's ocupados, a cola ready.
        Vb(mutex);
```

Proponga un control para evitar errores al acceder a la cola de procesos asociados al semáforo *s*, por ejemplo, en el caso que se intente sacar un elemento y la misma esté vacía. ¿Qué ventajas y desventajas tendría respecto de la implementación original?