4. (*2 p*) Dos procesos A y B se ejecutan concurrentemente en un determinado sistema. El proceso A ejecuta unas tareas ("Tareas 1") y alcanza un punto de encuentro. Posteriormente realiza otras tareas ("Tareas 2") y finaliza. Por su parte el proceso B ejecuta unas tareas ("Tareas 3") y llega al punto de encuentro. Posteriormente realiza otras tareas ("Tareas 4") y finaliza. El primer proceso que llega al punto de encuentro no puede continuar su ejecución hasta que no llegue el otro proceso. No se sabe que proceso comienza a ejecutarse primero o cual es el primero que termina. Escribir en pseudocódigo un programa de nombre `coordinación` que usando semáforos coordine la actividad de los procesos A y B. Dicho programa debe tener cuatro partes: declaración de variables y semáforos, código del proceso A, código del proceso B y código para inicializar los semáforos y lanzar la ejecución concurrente de ambos procesos.

Solución:

La solución que se propone para modelar el punto de encuentro entre los procesos A y B utiliza dos semáforos binarios S1 y S2. Ambos semáforos son inicializados al valor 0 ya que se utilizan para sincronizar.

```
program/module coordinación;
var
      S1, S2: semáforo;

process proceso_A;
begin
      /* Tareas 1*/
      señal(S2);
      espera(S1);
      /* Tareas 2*/
end

process proceso_B;
begin
      /* Tareas 3*/
      señal(S1);
      espera(S2);
      /* Tareas 4*/
end

begin
      inicializa (S1,0);
      inicializa (S2,0);
      cobegin
            proceso_A, proceso_B;
      coend
end coordinación;
```

SOLUCION EXAMEN SEPTIEMBRE 2010

1. Conteste **razonadamente** a los siguientes apartados:

a) *(1 p)* Explique como es posible determinar el número de procesos bloqueados en la cola de un semáforo general o generalizado.

b) *(1 p)* Explique el funcionamiento del algoritmo MLFQ de planificación de procesos.

Solución:

a) Un semáforo generalizado es un tipo especial de dato sobre el que únicamente se puede realizar operaciones que tengan garantizada su indivisibilidad o atomicidad, es decir, que las instrucciones máquina que las implementan tiene garantizadas que se ejecutan hasta completarse sin ser interrumpidas. En principio las operaciones básicas para operar sobre un semáforo generalizado son tres: inicializar, espera y señal que se pueden definir por ejemplo de la forma indicada en el Cuadro 1 (pp. 122 del libro base del curso 2009-2010).

operación de inicializar

```
inicializa (S; V)
poner el valor del semáforo S al valor de V
numero_suspendidos := 0
```

operación de espera

```
espera (S)
if S > 0 then S := S - 1
else
    numero_supendidos := numero_supendidos +1
    suspender la tarea que hace la llamada
    y ponerla en la cola de tareas
```

operación de señal

```
señal (S)
if numero_suspendidos > 0 then
    numero_supendidos := numero_supendidos - 1
    pasar al estado preparado un proceso suspendido
else S := S + 1
```

Cuadro 1. Posible definición de algunas operaciones sobre un semáforo generalizado

Una posible forma de conocer el número de procesos bloqueados en la cola de un semáforo general o generalizado sería definir e implementar una operación adicional que devuelva el valor del número de procesos suspendidos en la cola del semáforo. Si se llama a dicha operación `info`, una posible definición de la misma sería la siguiente:

```
info(S,H)
{
      H= número_suspendidos;
}
```

Donde H es la dirección de memoria donde se desea almacenar el número de procesos suspendidos en la cola del semáforo en un cierto instante de tiempo, para que pueda ser

4. (*2 p*) En un restaurante autoservicio inicialmente vacío existen S puestos para comer. Cuando un cliente llega al restaurante lo primero que hace es buscar un puesto libre, si no encuentra ninguno se marcha. Si encuentra algún puesto libre lo reserva dejando allí sus cosas. A continuación coge, sin necesidad de esperar ninguna cola, lo que desea comer de unos mostradores. Luego se pone en una cola para que le cobre un dependiente. Finalmente vuelve a su puesto para comer. El dependiente sólo atiende a los clientes de uno en uno y debe ser avisado por el cliente para que le cobre. Cuando no está cobrando a los clientes el dependiente se dedica a reponer los mostradores. Escribir en pseudocódigo un programa de nombre `autoservicio` que usando semáforos coordine la actividad de los clientes y del dependiente. Dicho programa debe tener cuatro partes: declaración de variables y semáforos, código del proceso `cliente`, código del proceso `dependiente` y código para inicializar los semáforos y lanzar la ejecución concurrente de ambos procesos.

Solución:

La solución que se propone para este problema tiene en cuenta lo siguiente:

- Cuando un cliente llega al restaurante, lo primero que hace es buscar un puesto libre, de los S disponibles, sino encuentra ninguno se marcha. Para modelar esta situación se va a utilizar una variable global de nombre `puestos_libres` que será consultada por los procesos cliente. Inicialmente `puestos_libres= S`. Si `puestos_libres=0` entonces el cliente se marcha, en caso contrario dicha variable debe ser decrementada en una unidad. Cuando el cliente termine de comer y se levante para irse la variable se incrementará en una unidad. Puesto que se trata de una variable que puede ser leída o escrita por diferentes procesos clientes es necesario asegurar la exclusión mutua en su uso. Para ello se va a utilizar el semáforo binario `ex_mut1` que se inicializa con el valor 1.

- Una vez que el cliente coge lo que quiere comer se pone en una cola para pagar, esta situación se puede modelar con un semáforo binario de nombre `a_pagar`. Inicialmente se inicializa a 1, ya que el primero que llega no tiene que esperar cola.

- Cada cliente debe avisar al dependiente para que le cobre, ya que este puede estar realizando tareas de reposición. Por tarea de reposición se entiende por ejemplo sacar un producto de una caja y colocarlo en un mostrador, llevar una caja de un sitio a otros, etc. Una tarea de reposición se ejecuta rápidamente, por lo que cuando el dependiente es avisado por el cliente, bien de viva voz o pulsando un llamador, el dependiente tarda poco en estar listo para cobrarle.

- Nótese que el dependiente no permanece sin hacer nada, es decir, bloqueado en espera de que le avise un cliente, sino que realiza una espera activa. Por ello no es posible utilizar un semáforo para avisar al cocinero. Se puede utilizar una variable global boleana de nombre `aviso_dependiente`, el cliente si desea ser atendido debe hacer

`aviso_dependiente=true`. El dependiente cuando termina de atender a un cliente hace `aviso_dependiente=false`. Inicialmente está variable se configura a `false`, ya que el dependiente está haciendo tareas de reposición. Puesto que se trata de una variable que puede ser leída o escrita por el clientes en el mostrador o el dependiente es necesario asegurar la exclusión mutua en su uso. Para ello se va a utilizar el semáforo binario `ex_mut2` que se inicializa con el valor 1.

- Finalmente es necesario usar un semáforo binario en el que espere el cliente hasta ser cobrado por el dependiente. Dicho semáforo se va a denotar como `finalizar_pago`, que debe inicializar a 0, el cliente debe hacer una operación `esperar(finalizar_pago)` mientras que corresponde al dependiente hacer una operación `señal(finalizar_pago)`.

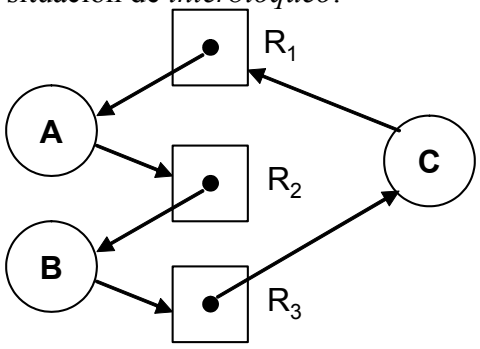
```
program/module autoservicio;
var
      puestos_libres: integer;
      aviso_dependiente: boolean;
      a_pagar, ex_mut1, ex_mut2, finalizar_pago: semáforo;

process cliente;
begin
      /* Entra en el restaurante*/
      esperar(ex_mut1);
      if puestos_libres>0 then
            begin
                  /*Dejar pertenencias en el puesto para reservarlo*/
                  puestos_libres:= puestos_libres -1;
                  señal(ex_mut1);
                  /*Coger comida*/
                  esperar(a_pagar);
                  esperar(ex_mut2)
                  aviso_dependiente:= true;
                  señal(ex_mut2);
                  esperar(finalizar_pago);
                  señal(a_pagar);
                  /*Comer*/
                  esperar(ex_mut1);
                  puestos_libres:= puestos_libres +1;
                  señal(ex_mut1);
                  /*Abandona el restaurante*/
            end
      else
            begin
                  señal(ex_mut1);
                  /*Abandona el restaurante*/
            end
end
```

```
process dependiente;
begin
      loop
            esperar(ex_mut2);
            if aviso_dependiente==true then
                  begin
                        /*Cobrar cliente*/
                        señal(finalizar_pago);
                        aviso_dependiente:= false;
                        señal(ex_mut2);
                  end
            else
                  begin
                        señal(ex_mut2);
                        /* Hacer una tarea de reposición*/
                  end
            end
      end
end


begin
      puestos_libres :=S;
      aviso_dependiete :=false;
      inicializa (a_pagar,1);
      inicializa (ex_mut1,1);
      inicializa (ex_mut2,1);
      inicializa (finalizar_pago,0);
      cobegin
            clientes, dependiente;
      coend
end autoservicio;
```

3. (*2 p*) Supóngase un sistema que permite retención y espera, exclusión mutua y expropiación en sus recursos. En un determinado instante de tiempo el grafo de asignación de los recursos R1, R2 y R3 del sistema (representados por cuadrados y cada instancia por un punto negro) a los procesos A, B y C (representados por circulos) es el que se muestra en la Figura. Explicar **razonadamente** si se cumplen las condiciones para que se produzca una situación de *interbloqueo*.



Solución:

Para que se produzca una situación de interbloqueo se deben de cumplir de forma simultánea las siguientes cuatro condiciones:

1) Exclusión mutua.
2) Retención y espera.
3) No existencia de expropiación.
4) Espera circular.

En el enunciado se afirma que se cumplen las dos primeras condiciones: exclusión mutua y retención y espera. Sin embargo, de acuerdo con el enunciado, la tercera condición no se cumple ya que el sistema permite la expropiación de recursos, es decir, el sistema operativo puede expropiar los recursos a los procesos que los retienen.

En consecuencia como no se cumple una de las cuatro condiciones necesarias **no se puede dar una situación de interbloqueo**.

Aunque ya no es necesario, analizando el grafo de asignación de recursos se observa la existencia del siguiente ciclo

$$A \rightarrow R2 \rightarrow B \rightarrow R3 \rightarrow C \rightarrow R1 \rightarrow A$$

Como sólo existe una instancia de cada recurso la existencia de dicho ciclo es una condición necesaria y suficiente para garantizar la existencia de espera circular.

4. (*2 p*) El baño de caballeros de un centro comercial posee una capacidad para seis caballeros. Cuando el servicio está completo los caballeros que desean pasar deben esperar fuera haciendo cola al lado de la puerta. Además si el operario de limpieza está limpiando el baño no puede pasar ningún caballero. Por otra parte, el operario sólo pasa a limpiar el baño si éste está vacío. Escribir en pseudocódigo un programa de nombre `baño_caballeros` que usando **semáforos binarios** coordine la actividad de los caballeros y del operario de limpieza. Dicho programa debe tener cuatro partes: declaración de variables y semáforos, código del proceso `caballero`, código del proceso `operario_limpieza` y código para inicializar los semáforos y lanzar la ejecución concurrente de los procesos.

Nota: Antes de escribir el pseudocódigo se debe explicar **adecuadamente** el significado de cada uno de los semáforos y variables que se van a utilizar en el mismo.

Solución:

La solución que se propone para este problema utiliza las siguientes variables globales y semáforos binarios:

- `contador`: Variable global de tipo entero para llevar la cuenta del número de caballeros.
- `sem1`. Semáforo binario que se utiliza para garantizar la exclusión mutua en el uso de la variable global `contador`.
- `sem2`. Semáforo binario que se utiliza para sincronizar la actividad del operario de limpieza y los caballeros. El primer caballero que desee entrar al baño debe comprobar que no está el operario realizando una operación `esperar(sem2)`. Asimismo el último caballero que salga del baño quedando éste vacío debe avisar al operario de esta circunstancia mediante la realización de una operación `señal(sem2)`. Por su parte el operario cuando desea entrar al baño a limpiarlo debe realizar una operación `esperar(sem2)` y cuando termine de limpiarlo debe realizar una operación `señal(sem2)`.
- `sem3`. Semáforo binario que se utiliza para sincronizar la entrada al baño de los caballeros.

El pseucódigo del programa `baño_caballeros` se muestra a continuación. Nótese que se ha utilizado un pseudocódigo distinto al usado en el libro base de teoría, en concreto este pseudocódigo está inspirado en el lenguaje de programación C.

```
/*Pseudocódigo del programa baño_caballeros*/

/*Definición variables y semáforos*/
semáforo sem1, sem2, sem3;
int contador=0;

void caballero() /*Proceso caballero*/
{
      espera(sem1);
      contador=contador+1;
      if(contador==1)
      {
            espera(sem2);
            señal(sem1);
      }
      else  if(contador>6){
            señal(sem1);
            esperar(sem3);
      }
      else señal(sem1);

      /* Usar baño */

      señal(sem3);
      espera(sem1);
      contador=contador-1;
      if (contador==0) señal(sem2);
      señal(sem1);
}

void operario_limpieza()/*Proceso operario_limpieza*/
{
            esperar(sem2);
            /* Limpiar Baño */
            señal(sem2);
}

void main() /*Inicialización de semáforos y ejecución concurrente*/
{
      inicializar(sem1,1);
      inicializar(sem2,1);
      inicializar(sem3,1);
      ejecución_concurrente(caballero,..,caballero,operario_limpieza);
}
```

5. (*2 p*) Un alumno de la asignatura Sistemas Operativos I ha escrito el pseudocódigo (ver Figura 1) de un programa de nombre `Prueba` que intenta resolver el problema de la exclusión mutua. Explicar **razonadamente** si el alumno ha conseguido su objetivo.

```
program/module Prueba;

var X: boolean;

process P1
begin
  loop
       while X = true do
             /* Espera a que el dispositivo se libere */
       end;
       X := true;
       /* Uso del recurso Sección Crítica */
       X := false;
       /* resto del proceso */
  end
end P1;

process P2
begin
  loop
       while X = true do
             /* Espera a que el dispositivo se libere */
       end;
       X := true;
       /* Uso del recurso Sección Crítica */
       X := false;
       /* resto del proceso */
  end
end P2;

begin /* Prueba*/
  X := false;
  cobegin
       P1;
       P2;
  coend
end Prueba;
```

Figura 1: Pseudocódigo del programa `Prueba`

Solución:

La solución que propone el alumno para resolver el problema de la exclusión mutua hace uso de una variable compartida X de tipo booleano que se suele denominar *indicador* (flag). Se asocia a cada recurso sobre el que se desea la exclusión mutua un indicador. Antes de acceder al recurso, un proceso debe examinar el indicador asociado que podrá tomar dos valores (`true` o `false`) que indicarán, respectivamente, si el recurso está siendo utilizado o que está disponible. La acción de bloqueo se realiza con la activación del indicador (`X:=true`) y la de desbloqueo con su desactivación (`X:=false`).

El uso de un único indicador, como propone el alumno, no resuelve el problema de la exclusión mutua ya que al ser la comprobación y la puesta del indicador a falso operaciones separadas, puede ocurrir que se entrelace el uso del recurso por ambos procesos.

Sólo si el computador dispusiera de una instrucción que permitiera comprobar el estado y modificarlo simultáneamente, el programa propuesto permitiría el uso del recurso sin el problema del entrelazado.

3. Conteste **razonadamente** a los siguientes apartados:

a) (*1 p*) Enumere y explique las condiciones que se deben cumplir de forma simultánea para que se de una situación de interbloqueo.

b) (*0.5 p*) ¿Qué condiciones de las enumeradas en el apartado anterior pueden ser detectadas utilizando un grafo de asignación de recursos?

c) (*0.5 p*) Explique como se detectan interbloqueos en un grafo de asignación de recursos.

Solución:

a) Para que se de una situación de interbloqueo se deben cumplir de forma simultánea las cuatro condiciones siguientes:

- *Exclusión mutua*. Los procesos utilizan de forma exclusiva los recursos que han adquirido. Si otro proceso pide el recurso debe esperar a que este sea liberado.
- *Retención y espera*. Los procesos retienen los recursos que han adquirido mientras esperan para adquirir otros recursos que están siendo retenidos por otros procesos.
- *No existencia de expropiación*. Los recursos no se pueden quitar a los procesos que los tienen; su liberación se produce voluntariamente una vez que los procesos han finalizado su tarea con ellos.
- *Espera circular*. Existe una cadena circular de procesos en la que cada uno retiene al menos un recurso que se solicita por el siguiente proceso de la cadena.

b) *Un grafo de asignación de recursos* es un grafo dirigido que indica las asignaciones de los recursos a los procesos y las peticiones que éstos realizan. Los nodos del grafo son procesos y recursos y cada arco conecta el nodo de un proceso con el nodo de un recurso.

En un grafo de asignación de recursos de las cuatro condiciones que se deben cumplir para que se produzca interbloqueo puede detectarse visualmente (o usando algún algoritmo de análisis de grafos) la condición de espera circular y de retención y espera. Cuando se usa un grafo de asignación de recursos para detectar interbloqueos se presupone que las otras dos condiciones (exclusión mutua y no existencia de expropiación) ya se cumplen, estas condiciones quedan fijadas por el sistema operativo.

c) En un grafo de asignación de recursos los interbloqueos se detectan siguiendo las siguientes reglas:

- Si el grafo no contiene ningún ciclo no hay ningún proceso que esté bloqueado, pero si lo contiene puede existir un interbloqueo.
- Si sólo hay un elemento por cada tipo de recurso la existencia de un ciclo es una condición necesaria y suficiente para que haya un interbloqueo.
- Si cada tipo de recurso tiene varios elementos, entonces la condición de existencia de un ciclo es necesaria pero no es suficiente para asegurar que existe un interbloqueo. En este caso una condición suficiente es la existencia de un ciclo en el que no hay ningún camino que salga de alguno de los nodos que lo forman que a su vez no sea ciclo.