Solución Ejercicios T4. Semáforos

Ejercicio 1

En un comedor de un colegio hay n alumnos. Cuando un alumno, que normalmente está estudiando en su clase, tiene hambre va al comedor y se acerca a alguna de las tres barras para recoger el menú. En estos menús, que varían cada día, sólo se puede elegir o cambiar el postre o café. Para el postre hay dos mostradores mientras que para el café hay un único mostrador. Realizar un programa que, usando semáforos, coordine las peticiones de los alumnos en el comedor.

Solución

Este es un problema en el que el número de recursos es limitado y definido, tres barras, dos de postre y una de café.

Como podemos observar, se utilizan cuatro semáforos: el semáforo *comedor* asociado a la sala y que se inicializa a n, indicando que tiene una capacidad de n comensales; *menu* inicializado a 3, representando a las tres barras donde se sirven menús; *postre* inicializado a 2, indicando las dos barras donde se sirven postres; y el semáforo *cafe* inicilizado a 1, simulando la única barra donde existe café.

```
#define alumnos n
TSemaforo comedor, menu, postre, cafe;
bool postre_cafe;

void alumno()
{
  while (true)
  {
    P(comedor);
    entrar_en_comedor();
    P(menu);
    coger_menu();
    V(menu);
    decidir_postre_cafe();
    if (postre_cafe == postre)
    {
      P(postre);
      coger_postre();
      V(postre);
    }
    else
    {
      P(cafe);
      coger_cafe();
      V(cafe);
    }
    comer();
    V(comedor);
  }
}

Void main()
{
  inicializar(comedor, n);  inicializar(menu, 3);
  inicializar(postre, 2);  inicializar(cafe, 1);
  cobegin
    alumno1();alumno2(); … alumnom();
  coend;
}
```

Observemos como en los semáforos *menú*, *postre* y *cafe* se ejecuta la operación *P* justa antes de llevar a cabo la acción correspondiente y la operación *V* inmediatamente después, para liberar la barra y que otro compañero pueda acceder. En cambio, en el semáforo *comedor* se ejecuta la operación *P* al entrar en el comedor y la operación *V* al salir del mismo, asegurando de este modo que no se superará la capacidad del comedor.

Ejercicio 2

Un parque de atracciones decide usar semáforos en sus actividades de mayor afluencia de público. En concreto en las pistas blandas (con 5 pistas) y el kamikaze(con dos pistas). Además, por exigencias de seguridad, se obliga a que no haya más de 2000 personas en el parque, todos deben de ducharse antes de entrar en estas actividades que se realizarán en el orden que se han descrito. Realizar un programa con semáforos que cumpla las exigencias descritas del parque acuático.

Solución

Como es fácilmente observable, este problema guarda muchas similitudes con el anterior, también es un problema de recursos limitados y definidos, 2000 personas de capacidad máxima en el parque, cinco pistas blandas, dos kamikaze y número de duchas indefinido.

Como en el caso anterior, asociaremos un semáforo a cada uno de los recursos anteriores y trabajaremos con la inicialización de mismos. También como en el problema anterior, se puede ver como en la utilización de las atracciones el semáforo correspondiente se cierra y se libera (operaciones *P* y *V*) antes y después del uso de la atracción. En cambio, en el semáforo parque la operación *P* se ejecuta al entrar y la operación *V* al salir, garantizando en todo momento no exceder la capacidad del parque.

Observemos la estructura secuencial del código que garantiza que las actividades se realizan en el orden en que se han descrito, en cumplimiento de la definición del problema.

```
#define duchas n
TSemaforo blandas, kamikaze, parque, duchas;

void clientei()
{
  while (true)
  {
    P(parque);
    entrar_en_el_parque();
    P(duchas);
    ducharse();
    V(duchas);
    P(blandas);
    atraccion_pistasblandas();
    V(blandas);
    P(kamikaze);
    Atraccion_kamikaze();
    V(kamikaze);
    V(parque);
  }
}

Void main()
{
  inicializar(parque, 2000);
  inicializar(duchas, n);
  inicializar(blandas, 5);
  inicializar(kamikaze, 2);
  cobegin
    cliente1(); cliente2(); … clientem();
  coend;
}
```

Ejercicio 3

En el almacén de una empresa de logística existen distintos toritos que pueden ser utilizados por personal del almacén que mueve los bultos internamente y por personal de los muelles que carga los camiones de transporte. Supongamos que la gestión de los toritos se implementa de la siguiente forma:

```
TSemáforo sc, mutex;
int numero;
```

```
void muelle_i()
{
  . . .
  P(sc);
  coger_torito();
  V(sc);
  . . .
}
```

```
void almacen_j()
{
  . . .
  P(mutex);
    numero++;
    if (numero==1) P(sc);
  V(mutex)
  coger_toriro();
  P(mutex);
    numero--;
    if (numero==0) V(sc);
  V(mutex)
  . . .
}
```

```
void main()
{
  inicializar(mutex, 1);
  inicializar(sc, 1);
  numero=0;
  cobegin
    muelle_1();…; muelle_n(); almacen_1(); … ; almacen_m();
  coend;
}
```

1) ¿Pueden utilizar a la vez los toritos el personal del almacén y del muelle?
2) Si hay un señor del almacén utilizando un torito, ¿puede otro del almacén utilizar algún torito simultáneamente?
3) Si una persona de los muelles está cargando, ¿puede otra persona de los muelles cargar al mismo tiempo?
4) Indicar si existe algún tipo de prioridad. En caso afirmativo, realizar las modificaciones necesarias para que no exista prioridad.

Solución

En el apartado 1 la respuesta es negativa, no pueden utilizar ambos tipos de personas los toritos simultáneamente. Simplemente tenemos que observar el código que ejecuta el personal del muelle, antes de coger un torito todos ejecutan una operación *P* sobre el semáforo asociado a la sección crítica *sc* y el semáforo se ha inicializado a 1.

En el apartado 2 la respuesta es afirmativa, varias personas del almacén pueden estar utilizando toritos al mismo tiempo. Si observamos el código correspondiente al almacén, podemos observar que tan sólo el primer señor del almacén que quiere coger un torito ejecuta la operación *P* sobre el semáforo de la sección crítica, mientras que el resto no realizarán la citada operación mientras haya alguna persona del almacén utilizando los toritos. Es la técnica utilizada para permitir que varios procesos del mismo tipo estén simultáneamente en la sección crítica.

No ocurre lo mismo en el apartado 3, no pueden haber dos personas de los muelles usando toritos al mismo tiempo. Todas las personas del muelle tienen que ejecutar obligatoriamente la operación P sobre el semáforo de la sección crítica inicializado a 1, por lo que sólo podrán coger un torito en un momento dado.

Finalmente, indicar que sí existe prioridad a favor en este caso del personal del almacén. Observemos que la prioridad está implícita en el código y no existe ningún semáforo asociado al tratamiento de ésta. La prioridad se puede eliminar simplemente añadiendo un nuevo semáforo para el tratamiento de la prioridad y utilizándolo de la misma manera que se usó en el problema de lectores y escritores sin prioridad.

```
TSemáforo sc, mutex, fifo;
int numero;
```

```
void muelle_i()
{
  . . .
  P(fifo);
  P(sc);
  V(fifo);
  coger_torito();
  V(sc);
  . . .
}
```

```
void almacen_j()
{
  . . .
  P(fifo);
  P(mutex);
  numero++;
    if (numero==1) P(sc);
  V(mutex);
  V(fifo);
  coger_toriro();
  P(mutex);
    numero--;
    if (numero==0) V(sc);
  V(mutex)
  . . .
}
```

```
void main()
{
  inicializar(mutex, 1);
  inicializar(sc, 1);
  inicializar(fifo, 1);
  numero=0;
  cobegin
    muelle_1();…; muelle_n(); almacen_1(); … ; almacen_m();
  coend;
}
```

Ejercicio 4

En una nave industrial existe una máquina de inyectar que deja cada pieza producida en una cinta transportadora de tamaño limitado. Existe un robot que recoge las piezas de la cinta (una cada vez) y las deja en las cajas de embalaje. Finalmente, tenemos un operario que, de vez en cuando, recoge 3 piezas para realizar el control de calidad, si no hay tres piezas en la cinta lo intentará más tarde. Resuelve el escenario anterior mediante semáforos.

```
#define tamaño_cinta N
TSemáforo e, s, n;        // s exclusión mutua para la variable piezas
```

```
void Inyector()
{
  while (true)
  {
    Producer_pieza();
    P(e);
    P(s);
    piezas++;
    añadir_cinta();
    V(s);
    V(n);
  }
}
```

```
void Robot()
{
  while (true)
  {
    P(n);
    P(s);
    coger_ cinta();
    piezas--;
    V(s);
    V(e);
    Robot_coge_pieza_y_embala();
  }
}
```

```
void Operario()
{
  while (true)
  {
   P(s);
   If (piezas>=3)
   {
```

```
        Piezas=piezas-3;
        V(e); V(e); V(e);
        Operario_coge_piezas();
         };
         V(s);
       }
    }
```

```
void main()
{
  inicializar(s, 1);
  inicializar(n, 0);
  inicializar(e, N);
  piezas=0;
  inicializar(e, tamaño_cinta);
  cobegin
     Inyector ();
     Robot ();
     Operario();
  coend;
}
```