

Práctica 3 de Sistemas Operativos

Raúl Zambrano Maestre

I.T. Informática de Gestión
Convocatoria de Febrero

ÍNDICE

Memoria explicativa de la práctica _____	pág. 03
Código fuente comentado _____	pág. 04
Código fuente sin comentar _____	pág. 06
Manual de usuario _____	pág. 10
Sistemas Concurrentes _____	pág. 11
Juego de pruebas _____	pág. 15
Bibliografía _____	pág. 15

MEMORIA EXPLICATIVA DE LA PRÁCTICA

La práctica consiste en la sincronización de procesos concurrentes, implementando soluciones basadas en semáforos que permitan hacer cumplir la exclusión mutua a dichos procesos.

El objetivo es resolver el supuesto práctico, que consistirá en la simulación de una cadena de montaje formada por una cinta transportadora y tres robots.

Por la cadena de montaje circulan tres productos: A, B y C. En la parte final de la cadena existen tres robots, que se encargan de retirar y empaquetar estos elementos.

El robot 1 se encarga de empaquetar en una misma caja el lote número 1 formado por los productos A y B, el robot 2 empaqueta el lote número 2 con los productos A y C, y el robot 3 empaqueta los productos B y C formando el lote número 3.

Continuamente llegan al final de la cadena dos productos distintos, dependiendo del lote que se forme deberá ser retirado y empaquetado por uno u otro robot.

Se deberá construir un programa que sincronice los tres robots, de forma que en todo momento sólo un robot retire los dos productos. La cinta no podrá insertar dos nuevos productos en la parte final de la cadena de producción hasta que no hayan sido retirados por el robot correspondiente.

La resolución de este problema se implementará mediante la utilización de BACI (Intérprete Concurrente de Ben-Ari), que simula la ejecución concurrente de procesos, soporta semáforos enteros y binarios, así como monitores.

Para la resolución del ejercicio se tendrán en cuenta las siguiente consideraciones:

Que existe un proceso cinta con acceso a la función

- InsertarProductos (producto1, producto2)

Que cada robot ejecuta un proceso con acceso a dos funciones:

- RetirarProductos (producto1, producto2)
- EmpaquetarProductos (producto1, producto2)

CÓDIGO FUENTE COMENTADO

La implementación del problema se ha dividido en varios procesos que se ejecutarán de forma concurrente en la sección “Cobegin”, donde se listan los procesos que van a ser ejecutados:

- Proceso Cinta (Genera los productos y los inserta en la parte final de la cadena)
- Proceso Robot_1 (Retira los productos AB de la cinta y los empaqueta)
- Proceso Robot_2 (Retira los productos AC de la cinta y los empaqueta)
- Proceso Robot_3 (Retira los productos BC de la cinta y los empaqueta)

Respecto al uso de semáforos se van a necesitar tres semáforos binarios, uno por cada robot encargado de retirar y empaquetar los lotes de productos: “robot1”, “robot2” y “robot3”. Estos tres semáforos se inicializarán a cero, ya que no podrán retirar nada de la cadena de montaje, hasta que la cinta transportadora coloque algún lote de productos al final de la cadena.

Además se necesitará un semáforo binario que controle que la cinta no inserte nuevos lotes de productos, hasta que el robot correspondiente los retire y los empaquete. Este semáforo se denominará “cinta”, y se inicializará con valor 1, para que inserte un lote en el final de la cadena, y espere a que sea retirado.

Finalmente haremos uso de un semáforo que controle el dispositivo de pantalla como una sección crítica, para evitar que varios procesos intenten escribir al mismo tiempo, para lo que nos serviremos de un semáforo binario que llamaremos “ts”.

PROCESO CINTA

El proceso se inicia con una orden wait para el semáforo cinta que al estar inicializado con valor uno, permite continuar con la ejecución del proceso insertando un lote, y cerrando el semáforo para evitar que se inserten nuevos lotes por parte del proceso cinta.

Seguidamente, se generan dos productos y se llama a la función de inserción de productos, la cual devuelve los valores uno, dos o tres si se generó un lote correcto, o cero si se generó un lote incorrecto. En caso de que el lote de productos sea incorrecto, se volverán a generar los productos.

Dependiendo del lote correcto generado, el proceso Cinta emitirá una orden signal para activar el semáforo del robot correcto.

PROCESOS ROBOT

Los procesos de los tres robots son iguales, con la particularidad de cada proceso robot actúa sobre el semáforo que le pertenece.

El proceso se inicia con una orden wait para el semáforo de ese robot, que al estar inicializado a cero quedará bloqueado hasta que el proceso Cinta lo ponga a uno mediante la orden signal. Al activarse el semáforo por parte del proceso Cinta significa que ese robot en concreto debe retirar y empaquetar los productos que se ha insertado al final de la cadena de montaje.

Finalmente activará el semáforo “cinta” para que el proceso Cinta pueda volver a insertar un nuevo lote de productos en la cadena de montaje.

CÓDIGO FUENTE SIN COMENTAR

/*****

Alumno: Raúl J. Zambrano Maestre

Asignatura: Sistemas Operativos (Práctica 3)

Titulación: I. T. Informática de Gestión

Curso: 2009-2010

Fecha de creación: 21 de Octubre de 2009

Última modificación: 3 de Noviembre de 2009

Compilador Utilizado: Interprete Concurrente Ben-Ari

*****/

```
binarysem cinta;    // Impide a Cinta colocar prod. mientras que un robot no retire
binarysem robot1;   // Impide a Robot_1 retirar nuevos prod. si no se lo indica Cinta
binarysem robot2;   // Impide a Robot_2 retirar nuevos prod. si no se lo indica Cinta
binarysem robot3;   // Impide a Robot_3 retirar nuevos prod. si no se lo indica Cinta
binarysem ts;       // ts-> sección critica que controla la pantalla como recurso
```

```
const int rango=3;
```

```
int lote=0;
```

```
int prod1, prod2;
```

```
int InsertarProductos(int p1, int p2)
```

```
{
```

```
    int tmp=0;
```

```
    if ((p1!=2)&&(p2!=0))
```

```
    {
```

```
        if ((p1==0)&&(p2==1))
```

```
            tmp=1;
```

```
        if ((p1==0)&&(p2==2))
```

```
            tmp=2;
```

```
        if ((p1==1)&&(p2==2))
```

```
            tmp=3;
```

```
    }
```

```
    return tmp;
```

```
}
```

```
void Cinta()
```

```
{
```

```
    for(;;)
```

```
    {
```

```
        wait(cinta);
```

```
        lote=0;
```

```
        while (lote==0)
```

```
        {
```

```

        prod1=random(rango);
        prod2=random(rango);
        lote=InsertarProductos(prod1,prod2);
    }

    if (lote==1)
    {
        wait(ts);
        cout << "Cinta genera los productos AB"<< endl;
        signal(ts);
        signal(robot1);
    }
    if (lote==2)
    {
        wait(ts);
        cout << "Cinta genera los productos AC"<< endl;
        signal(ts);
        signal(robot2);
    }
    if (lote==3)
    {
        wait(ts);
        cout << "Cinta genera los productos BC"<< endl;
        signal(ts);
        signal(robot3);
    }
}

void RetirarProductos(int pr1, int pr2)
{
    wait(ts);

    if ((pr1==0)&&(pr2==1))
        cout << "Robot 1 retira los productos AB"<< endl;

    if ((pr1==0)&&(pr2==2))
        cout << "Robot 2 retira los productos AC"<< endl;

    if ((pr1==1)&&(pr2==2))
        cout << "Robot 3 retira los productos BC"<< endl;
    signal(ts);
}

```

```

void EmpaquetarProductos(int pr1, int pr2)
{
    wait(ts);

    if ((pr1==0)&&(pr2==1))
        cout << "Robot 1 empaqueta los productos AB"<< endl;

    if ((pr1==0)&&(pr2==2))
        cout << "Robot 2 empaqueta los productos AC"<< endl;

    if ((pr1==1)&&(pr2==2))
        cout << "Robot 3 empaqueta los productos BC"<< endl;

    signal(ts);
}

```

```

void Robot_1()
{
    for(;;)
    {
        wait(robot1);
        RetirarProductos(prod1, prod2);
        EmpaquetarProductos(prod1, prod2);

        signal(cinta);
    }
}

```

```

void Robot_2()
{
    for(;;)
    {
        wait(robot2);
        RetirarProductos(prod1, prod2);
        EmpaquetarProductos(prod1, prod2);

        signal(cinta);
    }
}

```



```

void Robot_3()
{
    for(;;)
    {
        wait(robot3);
        RetirarProductos(prod1, prod2);
        EmpaquetarProductos(prod1, prod2);

        signal(cinta);
    }
}

main()
{
    initialsem(cinta,1);
    initialsem(robot1,0);
    initialsem(robot2,0);
    initialsem(robot3,0);
    initialsem(ts,1);

    cobegin
    {
        Cinta();
        Robot_1();
        Robot_2();
        Robot_3();
    }
}

```

MANUAL DE USUARIO

A continuación veremos los pasos a seguir para la ejecución de la práctica.

Comenzaremos abriendo un terminal de Linux, y accedemos al directorio donde se encuentra el Baci mediante el uso del comando “cd”, en nuestro caso se llama “p3”. Seguidamente compilaremos el programa fuente “p3.cm” mediante el uso del comando “bacc p3.cm”, generandose un programa objeto en pcode “p3.pco”, y un programa intermedio donde aparecen los pasos de compilación y errores, denominado “p3.lst”.

Por último tenemos que ejecutar el programa con el interprete de comandos mediante la instrucción “bainterp p3.pco”

El usuario no tiene que usar ningún comando más, ya que se visualizarán por pantalla todos los mensajes, simulando el funcionamiento de la cadena de producción.

Para detener la ejecución en el terminal bastará con que se pulse Ctrl + c.

SISTEMAS CONCURRENTES

El diseño de sistemas operativos está estrechamente ligado a la gestión de procesos e hilos, como sucede en:

- Multiprogramación: Múltiples procesos dentro de un sistema monoprocesador.
- Multiprocesamiento: Múltiples procesos dentro de un multiprocesador.
- Procesamiento distribuido: Múltiples procesos sobre múltiples sistemas.

La concurrencia es fundamental en todas estas áreas y en el diseño del sistema operativo. La concurrencia abarca varios aspectos, entre los cuales están la comunicación entre procesos, la competencia por recursos, la sincronización de actividades de múltiples procesos y la reserva de tiempo de procesador para los procesos.

Así pues, la computación concurrente consiste en la simultaneidad en la ejecución de múltiples tareas interactivas. Estas tareas pueden ser un conjunto de procesos o hilos de ejecución creados por un único programa. Las tareas se pueden ejecutar en una sola unidad central de proceso (multiprogramación), en varios procesadores (multiprocesamiento) o en una red de computadores (procesamiento distribuido).

La programación concurrente está relacionada con la programación paralela, pero enfatiza más la interacción entre tareas. Así, la correcta secuencia de interacciones o comunicaciones entre los procesos, y el acceso coordinado de recursos que se comparten por todos los procesos o tareas son las claves de esta disciplina.

Un sistema con un único procesador debe entrelazar en el tiempo los distintos procesos para ofrecer la apariencia de ejecución simultánea, aunque no se consigue un procesamiento paralelo real. La multiprogramación permite obtener importantes beneficios en la eficiencia del procesamiento y en la estructuración de los programas.

En el caso de un sistema de múltiples procesadores no sólo es posible entrelazar la ejecución de múltiples procesos, sino también solaparla.

Aunque pueda parecer que entrelazar y solapar procesos presentan problemas diferentes, ambas técnicas pueden verse como ejemplos de procesamiento concurrente y ambas presentan los mismos problemas:

- Compartir los recursos globales del sistema conlleva peligros, ya que si varios procesos usan la misma variable, y ambos la modifican y la consultan, el orden en que se realice cada operación resultará crítico.
- Gestionar los recursos de forma óptima por parte del sistema operativo puede resultar tremendamente complicado, ya que puede conducir a una situación de interbloqueo.

Así pues, es necesario proteger las variables y recursos compartidos, controlando las secciones de código crítico, es decir, la parte del código que accede a esos recursos. El acceso a estas secciones críticas por parte de un proceso, implica que deben ejecutarse hasta que se completen, antes de que sean accesibles por otro proceso.

En un sistema multiprocesador, aparecen los mismos problemas que en un sistema con un único procesador, ya que los problemas los generan los recursos compartidos (memoria, ficheros, ...), por lo que funcionarán las mismas soluciones.

Para solucionar estos problemas, el sistema operativo deberá tomar las siguientes precauciones en caso de que exista concurrencia:

1. Deberá ser capaz de seguir la pista de varios procesos, mediante el uso de bloques de control.
2. Tendrá que ubicar y desubicar varios recursos para cada proceso activo, como pueden ser tiempos del procesador en sistemas multiprogramados, memoria, ficheros y dispositivos de entrada/salida.
3. Deberá proteger los datos y recursos físicos de cada proceso frente a interferencias involuntarias de otros procesos.
4. El funcionamiento y el resultado que produzca, debe ser independiente de la velocidad a la que suceda su ejecución en relación con la velocidad de otros procesos concurrentes.

Los procesos de un sistema concurrente pueden clasificarse en base al grado de percepción de existencia que tienen respecto a los demás procesos, pudiéndose diferenciar los siguientes grupos:

1. Procesos que no se perciben entre sí, que son aquellos que no se pretenden que trabajen juntos. Aunque no estén trabajando juntos, el sistema operativo deberá preocuparse por la competencia por los recursos.
2. Procesos que se perciben indirectamente entre sí, que aunque no están al tanto de la presencia de otros procesos mediante su identificador, comparten accesos a algún objeto, como puede ser un buffer de entrada/salida.
3. Procesos que se perciben directamente entre sí, los cuales se comunican entre ellos mediante sus identificadores de procesos, y que han sido diseñados para trabajar conjuntamente en alguna tarea.

Los procesos concurrentes entran en conflicto entre ellos cuando compiten por el uso de un mismo recurso. En el caso de procesos en competencia deben afrontarse tres problemas de control.

Primero está la necesidad de exclusión mutua. Supongase que dos o más procesos requieren acceso a un recurso único y no compartible, nos referiremos a tal recurso como un recurso crítico, y al trozo de programa que lo usa como sección crítica, la cual se define como la sección de código dentro de un proceso que requiere acceso a recursos compartidos, y que no puede ser ejecutada mientras otro proceso esté en una sección de código correspondiente.

La aplicación de exclusión mutua crea dos problemas de control adicionales: el interbloqueo y la inanición.

El interbloqueo se produce cuando dos o más procesos son incapaces de actuar, porque cada uno de ellos, está esperando que alguno de los otros haga algo, quedando los procesos bloqueados mutuamente.

La inanición es la situación que se produce cuando un proceso preparado para avanzar es soslayado indefinidamente por el planificador; aunque es capaz de avanzar, nunca se le escoge.

Para que se cumpla la exclusión mutua deben cumplirse los siguientes requisitos:

1. Sólo se debe permitir un proceso a la vez dentro de su sección crítica, de entre todos los procesos que tienen secciones críticas para el mismo recurso u objeto.
2. Un proceso que se pare en su sección crítica debe hacerlo sin interferir en otros procesos.
3. No se debe permitir que un proceso que solicite acceso a una sección crítica sea postergado indefinidamente: ni interbloqueo, ni inanición.
4. Cuando ningún proceso esté en una sección crítica, a cualquier proceso que solicite entrar en su sección crítica debe permitírsele entrar sin demora.
5. No se hacen suposiciones sobre las velocidades relativas de los procesos ni sobre el número de procesadores.
6. Un proceso permanecerá dentro de su sección crítica durante un tiempo finito.

De los mecanismos que posee el sistema operativo y el lenguaje de programación para proporcionar recurrencia, hemos usado durante esta práctica los semáforos. Estos mecanismos resultan muy eficientes y fiables para dar soporte a la cooperación entre procesos.

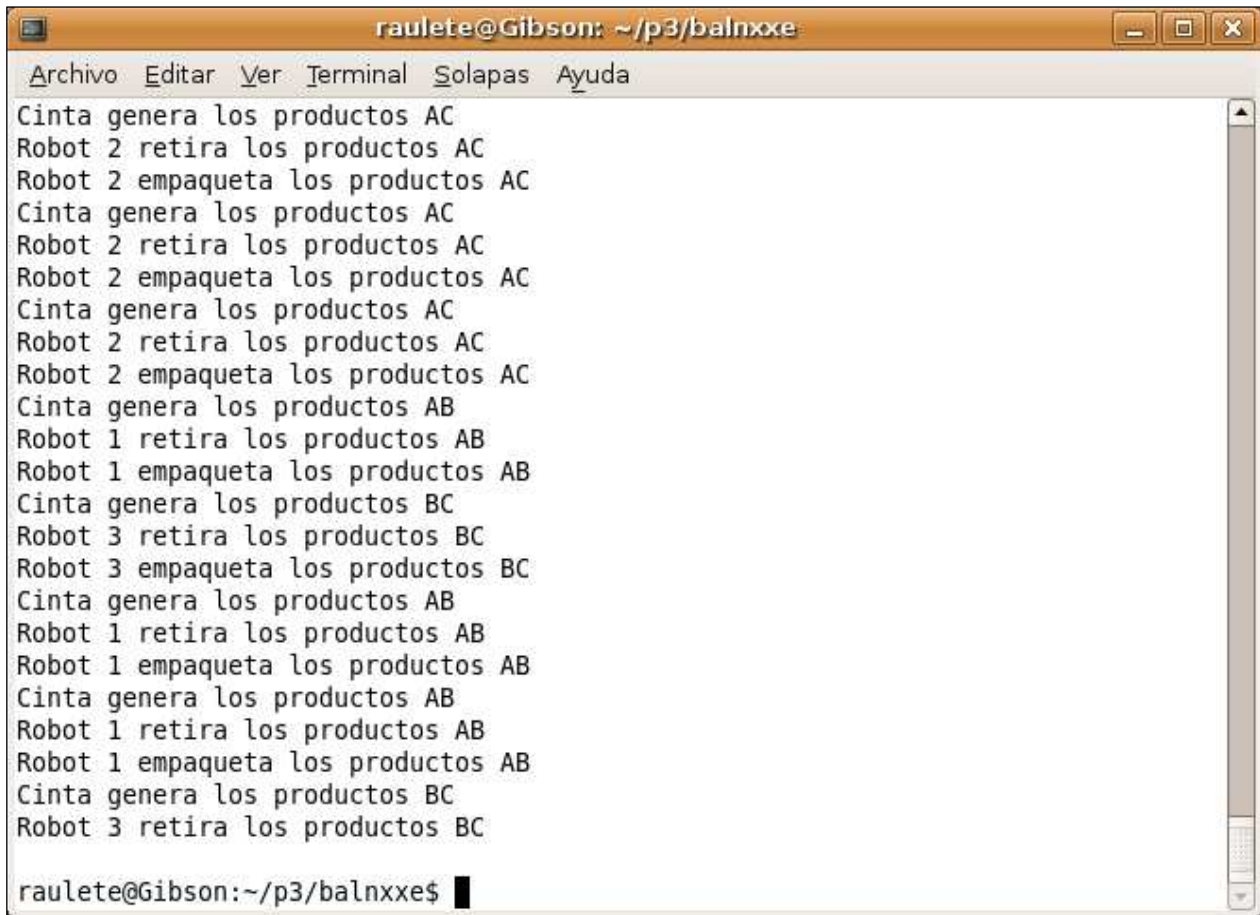
Los semáforos son variables especiales protegidas que contienen un valor entero, que constituyen un método clásico para restringir o permitir el acceso a recursos compartidos, y sobre las cuales hay definidas tres operaciones:

- Un semáforo se puede inicializar a un valor no negativo.
- La operación `semWait` decrementa el valor del semáforo. Si el valor pasa a ser negativo, entonces el proceso que está ejecutando la operación quedará bloqueado. En otro caso, el proceso continúa su ejecución
- La operación `semSignal` incrementa el valor del semáforo. Si el valor es menor o igual que cero, entonces se desbloquea uno de los procesos bloqueados en la operación `semWait`.

Dentro de los semáforos podemos diferenciar aquellos que sólo pueden tomar los valores 0 ó 1, y que se denominan semáforos binarios o mutex.

Para ambos, semáforos con contador y semáforos binarios, se utiliza una cola para mantener los procesos esperando por el semáforo.

JUEGO DE PRUEBAS



```
raulete@Gibson: ~/p3/balnxxe
Archivo  Editar  Ver  Terminal  Solapas  Ayuda
Cinta genera los productos AC
Robot 2 retira los productos AC
Robot 2 empaqueta los productos AC
Cinta genera los productos AC
Robot 2 retira los productos AC
Robot 2 empaqueta los productos AC
Cinta genera los productos AC
Robot 2 retira los productos AC
Robot 2 empaqueta los productos AC
Cinta genera los productos AB
Robot 1 retira los productos AB
Robot 1 empaqueta los productos AB
Cinta genera los productos BC
Robot 3 retira los productos BC
Robot 3 empaqueta los productos BC
Cinta genera los productos AB
Robot 1 retira los productos AB
Robot 1 empaqueta los productos AB
Cinta genera los productos AB
Robot 1 retira los productos AB
Robot 1 empaqueta los productos AB
Cinta genera los productos BC
Robot 3 retira los productos BC
raulete@Gibson:~/p3/balnxxe$
```

BIBLIOGRAFÍA

- [WIKIPEDIA] Wikipedia, la enciclopedia libre (<http://www.wikipedia.org/>).
- [PEARSON] Silberschartz Galvin: “Sistemas Operativos”
- [PEARSON] William Stallings: “Sistemas Operativos: Aspectos internos y principios de diseño”