
Problemas de Sistemas Operativos

Dpto. ACYA, Facultad de Informática, UCM

Modulo 3.2 – Comunicación y Sincronización

1.- Estudie el siguiente código que trata de resolver, únicamente por software y sin mediación del sistema operativo, el problema de la exclusión mutua entre dos hilos:

```
turno = 0

Hilo 0
while(TRUE){
    //Espera ocupada
    while(turno!=0);
    sección_crítica_H1();
    turno = 1;
    otro_código_H1()
}

Hilo 1
while(TRUE){
    //Espera ocupada
    while(turno!=1);
    sección_crítica_H2();
    turno = 0;
    otro_código_H2()
}
```

¿Resuelve correctamente el problema de la sección crítica garantizando exclusión, progreso y espera limitada?. Justifique su respuesta

2.- El algoritmo de Peterson es una solución software correcta para el problema de la sección crítica. A continuación se muestra el algoritmo de Peterson para dos procesos/hilos. Justifique que, efectivamente, cumple los tres requisitos de cualquier solución correcta del problema de la sección crítica.

```
turno = 0
interesado = {false, false}

Hilo 0
interesado[0] = true
turno = 1
//no hace nada, espera ocupada
while(interésado[1]&&turno==1);
sección_crítica();
//fin de la sección crítica
interesado[0] = false
sección_no_crítica();

Hilo 1
interesado[1] = true
turno = 0
//no hace nada, espera ocupada
while(interésado[0]&&turno==0);
sección_crítica();
//fin de la sección crítica
interesado[1] = false
sección_no_crítica();
```

3.- El algoritmo de la panadería de Lamport es un algoritmo de computación creado por el científico en computación Dr Leslie Lamport, para implementar la exclusión mutua de N procesos o hilos de ejecución sin necesidad de ningún soporte HW específico. Se inspira en el funcionamiento normal de cualquier comercio con un tendero y múltiples clientes (ej. una panadería). En este escenario, un cliente que llega a la panadería coge un número con su turno y espera pacientemente a ser atendido. Sin embargo, obtener el número para el turno no es trivial en un computador debido a la posibilidad de que varios procesos reciban el mismo. El siguiente pseudo-código (fuente: wikipedia) ilustra la solución de Lamport a dicho problema:

```
// Variables globales
Num[N] = {0, 0, 0, ..., 0};
Eligiendo[N] = {falso, falso, falso, ..., falso};

//Código del hilo i-ésimo
Hilo(i) {
    loop {
        //Calcula el número de turno
```

```

Eligiendo[i] = cierto;
Num[i] = 1 + max(Num[1],..., Num[N]);
Eligiendo[i] = falso;

//Compara con todos los hilos
for j in 1..N {
    //Si el hilo j está calculando su número, espera a que termine
    while( Eligiendo[j] ) {yield()}

    while( Num[j]!=0 && (Num[j]<Num[i]||(Num[j]==Num[i] && i<j)) ) {yield()}
}

// Sección crítica
...
// Fin de sección crítica

Número[i] = 0;

// Código restante
}
}

```

Discutir la validez de la solución de Lamport (seguridad, interbloqueo, inanición).

4.- Escriba un programa que cree tres hilos que se comunicarán entre ellos. El hilo 1 genera los 1000 primeros números pares y el hilo 2 los 1000 números impares. El hilo 3 irá leyendo esos números e imprimiéndolos en pantalla. Se debe garantizar que los números escritos por pantalla estén en orden: 1,2,3,4,5...

Implementar dicha sincronización mediante:

- a) Cerrojos y variables condicionales.
- b) Semáforos.

5.- Implementar la funcionalidad de un semáforo general a partir de cerrojos y variables condicionales. Para ello, defina un tipo de datos llamado `sem_t` (un struct en C con los campos necesarios), y las funciones `wait()` y `signal()` de acuerdo a la semántica estudiada en clase.

6.- Codificar el problema de los **lectores/escritores** de forma que sean los escritores los que tengan prioridad de acceso (conocido como “Segundo Problema de los Lectores/Escritores”). Emplear como método de comunicación/sincronización únicamente semáforos.

7.- En el problema de los **lectores/escritores**, si se prioriza a un colectivo, ya sea lectores o escritores, es posible que los procesos priorizados nieguen el acceso al recurso compartido al otro colectivo y que, por lo tanto, se produzca inanición (*starvation*). Codificar una solución para el problema de los lectores/escritores en la cual se otorgue acceso en función del orden de solicitud (conocido como “Tercer Problema de los Lectores/Escritores”). Emplear para ello semáforos.

8.- El Barbero Dormilón: una barbería está compuesta por una sala de espera, con n sillas, y la sala del barbero, que tiene un sillón para el cliente que está siendo atendido. Las condiciones de atención a los clientes son las siguientes:

- a) Si no hay ningún cliente, el barbero se va a dormir
- b) Si entra un cliente en la barbería y todas las sillas están ocupadas, el cliente abandona la barbería
- c) Si hay sitio y el barbero está ocupado, se sienta en una silla libre
- d) Si el barbero estaba dormido, el cliente le despierta
- e) Una vez que el cliente va a ser atendido, el barbero invocará `cortarPelo()` y el cliente `recibirCortePelo()`

Escriba un programa que coordine al barbero y a los clientes utilizando mutex y semáforos POSIX para la sincronización entre procesos.

9.- Un **monitor** lo podemos entender como un objeto cuyos métodos son ejecutados con exclusión mutua, por lo que un hilo/proceso que invoque un método del monitor se puede bloquear a la espera de un evento generado por otro hilo/proceso a través del mismo monitor. Dicha exclusión mutua y espera selectiva se puede implementar mediante cerrojos y variables condicionales.

Resolver el problema de la **cena de los filósofos** codificando un monitor (usando cerrojos y variables condicionales) basándose en el siguiente ejemplo de filósofo situado en la posición i -ésima de la mesa. El monitor debe incluir la implementación de `cogerPalillosMonitor()` y `dejarPalillosMonitor()` así como la declaración de variables (privadas al monitor) que sean necesarias.

```
Filósofo(int i){
    while(1){
        pensar();

        //Solicitamos al monitor la necesidad de coger los palillos
        cogerPalillosMonitor(i);
        comer();

        //Solicitud al monitor que queremos dejar los palillos
        dejarPalillosMonitor(i);
    }
}
```

10.- Considerar las siguientes primitivas de sincronización:

- a) **ADVANCE(A)**: incrementa el valor de la variable A en 1.
- b) **AWAIT(A,C)**: bloquea el proceso que ejecuta esta instrucción hasta que $A > C$

Usando estas primitivas, desarrollar una solución para el problema productor/consumidor para un único productor y consumidor con tamaño del almacén circular $n > 1$

11.- El Problema de los Fumadores [Suhas Patil]: considerar un sistema con tres procesos fumadores y un proceso agente. Continuamente cada fumador se prepara un cigarrillo y lo fuma. Para hacer un cigarrillo se necesitan tres ingredientes: **tabaco**, **papel** y **cerillas**. Uno de los procesos tiene infinito **papel**, otro **tabaco** y el tercero **cerillas**. El agente tiene provisión infinita de los tres ingredientes.

El agente coloca dos ingredientes distintos y de forma aleatoria en la mesa. El fumador que tiene el ingrediente que falta puede proceder a preparar y fumar un cigarrillo, haciendo una señal al agente cuando acaba, el agente en este instante repite la operación, pone otros dos de los tres ingredientes en la mesa. La operación se repite indefinidamente.

Escribir un programa que sincronice al agente y los fumadores mediante semáforos o mutexes y variables condicionales. Asíumase que el agente no tiene forma de consultar los ingredientes que posee cada fumador.

12.- Una tribu de salvajes se sirven comida de un caldero con M raciones de estofado de misionero. Cuando un salvaje desea comer, se sirve una ración del caldero a menos que esté vacío. Si está vacío deberá avisar al cocinero para que reponga otras M raciones de estofado, y entonces se podrá servir su ración. Un número arbitrario de salvajes se comportan del siguiente modo:

```
while (True){
    getServingFromPot()
    eat()
}
```

Y el concinero se compora de la siguiente forma:

```
while (True){
    putServingsInPot(M)
}
```

Sobre este código realice los siguiente:

- 1) Añada el código necesario para garantizar la correcta sincronización, usando semáforos POSIX y variables de tipo entero, booleano...

- 2) Añada el código necesario para garantizar la correcta sincronización, usando cerrojos y variables condicionales

Se deben cumplir las siguientes restricciones:

- a) Los salvajes no pueden invocar `getServingFromPot()` si el caldero está vacío
- b) El cocinero sólo puede invocar `putServingsInPot(M)` si el caldero está vacío

13.- El problema de la montaña rusa [Andrews's Concurrent Programming]: suponga que hay un hilo por cada pasajero, haciendo un total de n , y un hilo para el coche. Los pasajeros están constantemente esperando y subiéndose a la montaña rusa, que puede llevar C pasajeros ($C < n$). El coche sólo puede comenzar un viaje cuando está lleno. Se deben cumplir las siguientes restricciones:

- a) Los pasajeros deben invocar `board()` y `unboard()`
- b) El coche debe invocar `load()`, `run()` y `unload()`
- c) Los pasajeros no pueden hacer `board()` hasta que el coche haya invocado `load()`
- d) El coche no puede partir (`run()`) hasta que no haya C pasajeros en él.
- e) Los pasajeros no se pueden bajar hasta que el coche invoque `unload()`

A partir de las especificaciones realice los siguiente:

- 1) Escriba el código necesario usando semáforos POSIX generales (y variables enteras, booleanas...)
- 2) Escriba el código necesario usando cerrojos y variables condicionales (y variables enteras, booleanas...)

14.- El problema del sushi bar [Kenneth Reek]: supóngase un restaurante japonés con 5 asientos, si un cliente llega cuando hay un asiento libre puede sentarse inmediatamente. Sin embargo, si un cliente llega y encuentra los 5 asientos ocupados, supondrá que todos los clientes están cenando juntos y esperará hasta que todos ellos se levanten antes de tomar asiento. Además, los clientes son atendidos por orden. Codifique un programa que se comporte como un cliente según las especificaciones anteriores usando semáforos generales.

15.- El problema del cuidado de niños [Max Hailperin]: en cierta guardería se debe de cumplir que por cada tres niños debe de estar presente al menos un adulto. Codificar (con semáforos generales) el código correspondiente a los adultos de manera que se cumpla esta restricción y suponiendo que el número de niños se mantiene constante.

Amplíe la solución anterior para que tenga en cuenta la posible variación de la cantidad de niños. Codifique el hilo correspondiente a los niños.