Sincronización

Pedro O. Pérez M., PhD.

Diseño de sistemas embebidos avanzados
Tecnológico de Monterrey

pperezm@tec.mx

10-2021

- El modelo de proceso introducido anteriormente asumía que un proceso era un programa en ejecución con un solo hilo de control. Sin embargo, prácticamente todos los sistemas operativos modernos proporcionan características que permiten que un proceso contenga múltiples hilos de control.

Introducción

Un proceso cooperativo es aquel que puede afectar o verse afectado por otros procesos que se ejecutan en el sistema. Los procesos que cooperan pueden compartir directamente un espacio de direcciones lógicas (es decir, tanto código como datos) o se les permite compartir datos solo a través de archivos o mensajes. Sin embargo, el acceso simultáneo a los datos compartidos puede dar como resultado una inconsistencia en los datos.

Ya hemos visto que los procesos se pueden ejecutar al mismo tiempo o en paralelo. Vimos el papel de la programación de procesos y se describió cómo el programador del CPU cambia rápidamente entre procesos para proporcionar una ejecución concurrente. Esto significa que un proceso solo puede completar la ejecución parcialmente antes de que se programe otro proceso. De hecho, un proceso puede interrumpirse en cualquier punto de su flujo de instrucciones y el núcleo de procesamiento puede asignarse para ejecutar instrucciones de otro proceso.

Ver código de sincronización

Como pudieron observar, llegamos a este estado incorrecto porque permitimos que ambos hilos manipulen el contador de variables al mismo tiempo. Una situación como esta, en la que varios procesos acceden y manipulan los mismos datos al mismo tiempo y el resultado de la ejecución depende del orden particular en el que tiene lugar el acceso, se denomina **condición de carrera**. Para protegernos contra la condición de carrera anterior, debemos asegurarnos de que solo un proceso a la vez pueda manipular el contador de variables. Para hacer tal garantía, requerimos que los procesos estén sincronizados de alguna manera.

Si bien, situaciones como la que acabamos de describir ocurren con frecuencia en los sistemas operativos cuando diferentes partes del sistema manipulan los recursos. Ahora, la creciente importancia de los sistemas multinúcleo ha traído un mayor énfasis en el desarrollo de aplicaciones multiproceso. En tales aplicaciones, varios subprocesos, que posiblemente comparten datos, se ejecutan en paralelo en diferentes núcleos de procesamiento. Claramente, queremos que los cambios que resulten de tales actividades no interfieran entre sí. Debido a la importancia de este tema, en este tema hablaremos de la sincronización y coordinación de procesos.

Definición del problema

Considera un sistema que consta de N procesos P_0, P_1, ..., P_{n-1}. Cada proceso tiene un segmento de código, llamada sección crítica, en que el proceso puede cambiar variables comunes, actualizar una tabla, escribir un archivo, etc. La característica importante del sistema es que, cuando un proceso está ejecutando la sección crítica, ningún otro proceso puede ejecutarla. El problema de la sección crítica es diseñar un protocolo que los procesos puedan utilizar para cooperar.

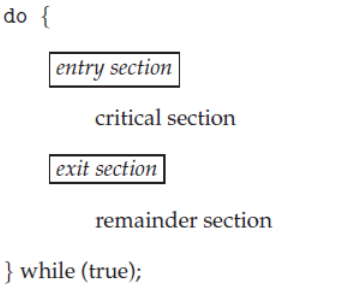

Figure 5.1 General structure of a typical process P_i.

Una solución al problema de la sección crítica debe satisfacer los siguientes tres requisitos:

1. **Exclusión mutua.** Si el proceso P_i está ejecutando su sección crítica, entonces ningún otro proceso puede ejecutar la suya.
2. **Progreso.** Si ningún proceso está ejecutando su sección crítica y algunos procesos desean ingresar a sus secciones críticas pueden participar para decidir cuál entrará a continuación en su sección crítica, y esta selección no se puede posponer indefinidamente.
3. **Espera limitada.** Existe un límite en la cantidad de veces que se permite que otros procesos ingresen a sus secciones críticas después de que un proceso haya realizado una solicitud para ingresar a su sección crítica y antes de que se otorgue dicha solicitud.

Semáforos

Un semáforo S es una variable entera a la que, además de la inicialización, se accede solo a través de dos operaciones atómicas estándar: *wait()* y *signal()*. La operación *wait()* se denominó originalmente P (del holandés proberen, "probar"); *signal()* originalmente se llamaba **V** (de verhogen, ïncrementar").

```
wait(S) {
    while (S <= 0)
       ; // busy wait
    S--;
}

signal(S) {
    S++;
}
```

Los sistemas operativos a menudo distinguen entre contadores y semáforos binarios.

- El valor de un semáforo binario sólo puede oscilar entre 0 y 1. Se utilizan para proporcionar exclusión mutua.
- Los semáforos contadores se pueden utilizar para controlar el acceso a un recurso determinado que consta de un número finito de instancias. El semáforo se inicializa con la cantidad de recursos disponibles. Cada proceso que desea utilizar un recurso realiza una operación de *wait()* en el semáforo (disminuyendo así el contador). Cuando un proceso libera un recurso, realiza una operación *signal()* (incrementando el contador). Cuando el recuento del semáforo llega a 0, se están utilizando todos los recursos. Después de eso, los procesos que deseen utilizar un recurso se bloquearán hasta que el recuento sea superior a 0.

Deadlocks

La implementación de un semáforo con una cola de espera puede resultar en una situación en la que dos o más procesos están esperando indefinidamente un evento que solo puede ser causado por uno de los procesos en espera. El evento en cuestión es la ejecución de una operación *signal()*. Cuando se alcanza tal estado, se dice que estos procesos están bloqueados.

P_0	P_1
`wait(S);`	`wait(Q);`
`wait(Q);`	`wait(S);`
.	.
.	.
.	.
`signal(S);`	`signal(Q);`
`signal(Q);`	`signal(S);`

Otro problema relacionado con los interbloqueos es el bloqueo indefinido o la inanición, una situación en la que los procesos esperan indefinidamente dentro del semáforo. El bloqueo indefinido puede ocurrir si eliminamos procesos de la lista asociada con un semáforo en orden LIFO (último en entrar, primero en salir).

Problema Productor-Consumidor

En computación, el problema del productor-consumidor es un ejemplo clásico de problema de sincronización de multiprocesos. El programa describe dos procesos, productor y consumidor, ambos comparten un buffer de tamaño finito. La tarea del productor es generar un producto, almacenarlo y comenzar nuevamente; mientras que el consumidor toma (simultáneamente) productos uno a uno. El problema consiste en que el productor no añada más productos que la capacidad del buffer y que el consumidor no intente tomar un producto si el buffer está vacío.

Problema Lectores-Escritores

Supón que una base de datos se va a compartir entre varios procesos concurrentes. Algunos de estos procesos pueden querer solo leer la base de datos, mientras que otros pueden querer actualizar (es decir, leer y escribir) la base de datos. Distinguimos entre estos dos tipos de procesos refiriéndonos a los primeros como lectores y a los segundos como escritores. Obviamente, si dos lectores acceden a los datos compartidos simultáneamente, no se producirán efectos adversos. Sin embargo, si un escritor y algún otro proceso (ya sea un lector o un escritor) acceden a la base de datos simultáneamente, puede producirse el caos.

Problema de los filósofos comedores

Considera a cinco filósofos que se pasan la vida pensando y comiendo. Los filósofos comparten una mesa circular rodeada de cinco sillas, cada una de las cuales pertenece a un filósofo. En el centro de la mesa hay un cuenco de arroz y la mesa se coloca con cinco palillos individuales. Cuando un filósofo piensa, no interactúa con sus colegas. De vez en cuando, un filósofo tiene hambre y trata de recoger los dos palillos que están más cerca de él (los palillos que están entre él y sus vecinos izquierdo y derecho). Un filósofo puede tomar solo un palillo a la vez. Evidentemente, no puede coger un palillo que ya está en la mano de un vecino. Cuando un filósofo hambriento tiene sus dos palillos al mismo tiempo, come sin soltar los palillos. Cuando termina de comer, deja ambos palillos y comienza a pensar de nuevo.

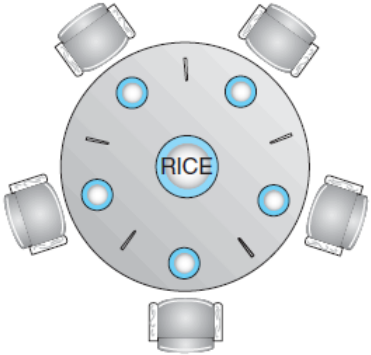

Figure 5.13 The situation of the dining philosopher

Actividad colaborativa

En grupos de 2 integrantes,

- Implementa, usando threads, el problema de los lectores-escritores.
- Sube tu código a Canvas.