UADER | FCyT

Sistemas Operativos

Procesos

Clase 6 - Unidad II

Lic. Alexis Sostersich
sostersich.alexis@uader.edu.ar

Teoría – Sistemas Operativos

Procesos

Unidad II – Clase 6

- Concurrencia.
- Exclusión mutua y sincronización
- Principios de la concurrencia
- Exclusión mutua: soporte hardware
- Semáforos
- Monitores
- Paso de mensajes

Concurrencia. Exclusión mutua y sincronización

Concurrencia. Exclusión mutua y sincronización

Los temas centrales del diseño de sistemas operativos están todos relacionados con la gestión de procesos e hilos:

Multiprogramación. Gestión de múltiples procesos dentro de un sistema monoprocesador.

Multiprocesamiento. Gestión de múltiples procesos dentro de un multiprocesador.

Procesamiento distribuido. Gestión de múltiples procesos que ejecutan sobre múltiples sistemas de cómputo distribuidos.

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 y la compartición de, o competencia por, recursos, la sincronización de actividades de múltiples procesos y la reserva de tiempo de procesador para los procesos.

La concurrencia aparece en tres contextos

Múltiples aplicaciones. La multiprogramación fue ideada para permitir compartir dinámicamente el tiempo de procesamiento entre varias aplicaciones activas.

Aplicaciones estructuradas. Como extensión de los principios del diseño modular y de la programación estructurada, algunas aplicaciones pueden ser programadas eficazmente como un conjunto de procesos concurrentes.

Estructura del sistema operativo. Las mismas ventajas constructivas son aplicables a la programación de sistemas y, de hecho, los sistemas operativos son a menudo implementados en sí mismos como un conjunto de procesos o hilos.

Términos clave relacionados con la concurrencia

sección crítica (critical section) 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.

Interbloqueo (deadlock) Situación en la cual dos o más procesos son incapaces de actuar porque cada uno está esperando que alguno de los otros haga algo.

círculo vicioso (livelock) Situación en la cual dos o más procesos cambian continuamente su estado en respuesta a cambios en los otros procesos, sin realizar ningún trabajo útil.

exclusión mutua (mutual exclusion) Requisito de que cuando un proceso esté en una sección crítica que accede a recursos compartidos, ningún otro proceso pueda estar en una sección crítica que acceda a ninguno de esos recursos compartidos.

Términos clave relacionados con la concurrencia

condición de carrera (race condition) Situación en la cual múltiples hilos o procesos leen y escriben un dato compartido y el resultado final depende de la coordinación relativa de sus ejecuciones.

Inanición (starvation) Situación en la cual un proceso preparado para avanzar es soslayado indefinidamente por el planificador; aunque es capaz de avanzar, nunca se le escoge.

Principios de la concurrencia

Principios de la concurrencia

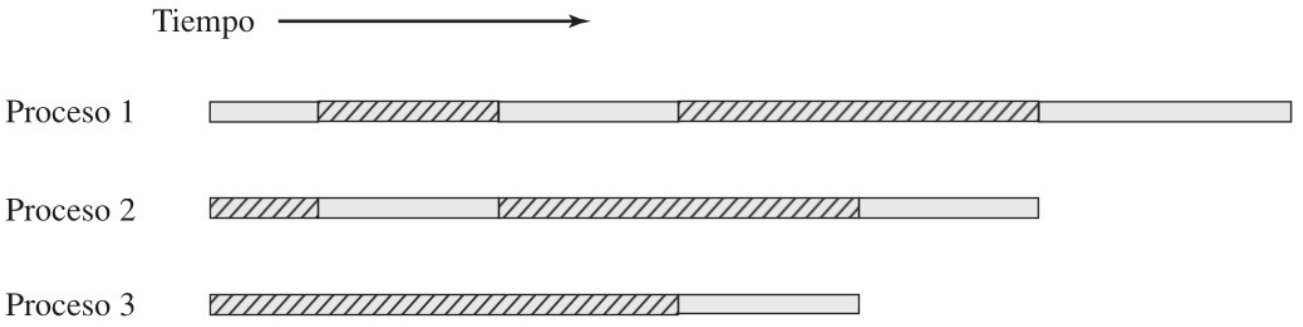
En un sistema multiprogramado de **procesador único, los procesos se entrelazan en el tiempo para ofrecer la apariencia de ejecución simultánea**. Aunque no se consigue procesamiento paralelo real, e ir cambiando de un proceso a otro supone cierta sobrecarga, la ejecución entrelazada proporciona importantes beneficios en la eficiencia del procesamiento y en la estructuración de los programas.

En un sistema de múltiples procesadores no sólo es posible entrelazar la ejecución de múltiples procesos sino también solaparlas. En primera instancia, puede parecer que el entrelazado y el solapamiento representan modos de ejecución fundamentalmente diferentes y que presentan diferentes problemas. De hecho, ambas técnicas pueden verse como ejemplos de procesamiento concurrente y ambas presentan los mismos problemas.

Principios de la concurrencia

En el caso de un monoprocesador, los problemas surgen de una característica básica de los sistemas multiprogramados: no puede predecirse la velocidad relativa de ejecución de los procesos. Ésta depende de la actividad de los otros procesos, de la forma en que el sistema operativo maneja las interrupciones y de las políticas de planificación del sistema operativo.

Multiprogramación y multiproceso



(a) Intercalado (multiprogramación, un procesador)

(b) Intercalado y solapamiento (multiproceso; dos procesadores)

Bloqueado Ejecutando

Dificultades mono y multi procesador

Se plantean las siguientes dificultades:

1. La compartición de recursos globales está cargada de peligros.
2. Para el sistema operativo es complicado gestionar la asignación de recursos de manera óptima
3. Llega a ser muy complicado localizar errores de programación porque los resultados son típicamente no deterministas y no reproducibles.

Todas estas dificultades se presentan también en un sistema multiprocesador, porque aquí tampoco es predecible la velocidad relativa de ejecución de los procesos. Un sistema multiprocesador debe bregar también con problemas derivados de la ejecución simultánea de múltiples procesos. Sin embargo, fundamentalmente los problemas son los mismos que aquéllos de un sistema monoprocesador.

Condición de carrera

Una condición de carrera sucede cuando múltiples procesos o hilos leen y escriben datos de manera que el resultado final depende del orden de ejecución de las instrucciones en los múltiples procesos.

Consideremos dos casos sencillos.

Como primer ejemplo, suponga que dos procesos, P1 y P2, comparten la variable global a. En algún punto de su ejecución, P1 actualiza a al valor 1 y, en el mismo punto en su ejecución, P2 actualiza a al valor 2. Así, las dos tareas compiten en una carrera por escribir la variable a.

En este ejemplo el «perdedor» de la carrera (el proceso que actualiza el último) determina el valor de a.

Condición de carrera

Para nuestro segundo ejemplo, considere dos procesos, P3 y P4, que comparten las variables globales b y c, con valores iniciales b = 1 y c = 2. En algún punto de su ejecución, P3 ejecuta la asignación b = b + c y, en algún punto de su ejecución, P4 ejecuta la asignación c = b + c. Note que los dos procesos actualizan diferentes variables. Sin embargo, los valores finales de las dos variables dependen del orden en que los dos procesos ejecuten estas dos asignaciones. Si P3 ejecuta su sentencia de asignación primero, entonces los valores finales serán b = 3 y c = 5. Si P4 ejecuta su sentencia de asignación primero, entonces los valores finales serán b = 4 y c = 3.

Preocupaciones del sistema operativo

¿Qué aspectos de diseño y gestión surgen por la existencia de la concurrencia? Pueden enumerarse las siguientes necesidades:

1. El sistema operativo debe ser capaz de seguir la pista de varios procesos.

2. El sistema operativo debe ubicar y desubicar varios recursos para cada proceso activo. Estos recursos incluyen:

- Tiempo de procesador. Esta es la misión de la planificación.
- Memoria. La mayoría de los sistemas operativos usan un esquema de memoria virtual.
- Ficheros.
- Dispositivos de E/S.

Preocupaciones del sistema operativo

3. El sistema operativo debe proteger los datos y recursos físicos de cada proceso frente a interferencias involuntarias de otros procesos. Esto involucra técnicas que relacionan memoria, ficheros y dispositivos de E/S.

4. El funcionamiento de un proceso 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.

Interacción de procesos

Podemos clasificar las formas en que los procesos interaccionan en base al grado en que perciben la existencia de cada uno de los otros.

- **Procesos que no se perciben entre sí.**
- **Procesos que se perciben indirectamente entre sí**.
- **Procesos que se perciben directamente entre sí**.

Las condiciones no serán siempre tan claras. Mejor dicho, algunos procesos pueden exhibir aspectos tanto de competición como de cooperación. No obstante, es constructivo examinar cada uno de los tres casos de esta lista y determinar sus implicaciones para el sistema operativo.

Interacción de procesos

- **Procesos que no se perciben entre sí.** Son procesos independientes que no se pretende que trabajen juntos. El mejor ejemplo de esta situación es la multiprogramación de múltiples procesos independientes. Estos bien pueden ser trabajos por lotes o bien sesiones interactivas o una mezcla. Aunque los procesos no estén trabajando juntos, el sistema operativo necesita preocuparse de la competencia por recursos. Por ejemplo, dos aplicaciones independientes pueden querer ambas acceder al mismo disco, fichero o impresora. El sistema operativo debe regular estos accesos.

Interacción de procesos

- **Procesos que se perciben indirectamente entre sí**. Son procesos que no están necesariamente al tanto de la presencia de los demás mediante sus respectivos ID de proceso, pero que comparten accesos a algún objeto, como un buffer de E/S. Tales procesos exhiben cooperación en la compartición del objeto común.
- **Procesos que se perciben directamente entre sí**. Son procesos capaces de comunicarse entre sí vía el ID del proceso y que son diseñados para trabajar conjuntamente en cierta actividad. De nuevo, tales procesos exhiben cooperación.

Interacción de procesos

Grado de percepción	Relación	Influencia que un proceso tiene sobre el otro	Potenciales problemas de control
Procesos que no se perciben entre sí	Competencia	• Los resultados de un proceso son • independientes de la acción de los otros • La temporización del proceso puede verse afectada	• Exclusión mutua • Interbloqueo (recurso renovable) • Inanición
Procesos que se perciben indirectamente entre sí (por ejemplo, objeto compartido)	Cooperación por compartición	• Los resultados de un proceso pueden depender de la información obtenida de otros • La temporización del proceso puede verse afectada	• Exclusión mutua • Interbloqueo (recurso renovable) • Inanición • Coherencia de datos
Procesos que se perciben directamente entre sí (tienen primitivas de comunicación a su disposición)	Cooperación por comunicación	• Los resultados de un proceso pueden depender de la información obtenida de otros • La temporización del proceso puede verse afectada	• Interbloqueo (recurso consumible) • Inanición

Competencia entre procesos por recursos

Los procesos concurrentes entran en conflicto entre ellos cuando compiten por el uso del mismo recurso. En su forma pura, puede describirse la situación como sigue. Dos o más procesos necesitan acceso a un recurso durante el curso de su ejecución. Ningún proceso se percata de la existencia de los otros procesos y ninguno debe verse afectado por la ejecución de los otros procesos. Esto conlleva que cada proceso debe dejar inalterado el estado de cada recurso que utilice. Ejemplos de recursos son los dispositivos de E/S, la memoria, el tiempo de procesador y el reloj.

No hay intercambio de información entre los procesos en competencia. No obstante, la ejecución de un proceso puede afectar al comportamiento de los procesos en competencia. En concreto, si dos procesos desean ambos acceder al mismo recurso único, entonces, el sistema operativo reservará el recurso para uno de ellos, y el otro tendrá que esperar. Por tanto, el proceso al que se le deniega el acceso será ralentizado. En un caso extremo, el proceso bloqueado puede no conseguir nunca el recurso y por tanto no terminar nunca satisfactoriamente.

Competencia entre procesos por recursos

En el caso de procesos en competencia, deben afrontarse tres problemas de control. Primero está **la necesidad de exclusión mutua**. Supóngase que dos o más procesos requieren acceso a un recurso único no compartible, como una impresora. Durante el curso de la ejecución, cada proceso estará enviando mandatos al dispositivo de E/S, recibiendo información de estado, enviando datos o recibiendo datos. Nos referiremos a tal recurso como un **recurso crítico**, y a la porción del programa que lo utiliza como la sección crítica del programa. **Es importante que sólo se permita un programa al tiempo en su sección crítica**. No podemos simplemente delegar en el sistema operativo para que entienda y aplique esta restricción porque los detalles de los requisitos pueden no ser obvios. En el caso de una impresora, por ejemplo, queremos que cualquier proceso individual tenga el control de la impresora mientras imprime el fichero completo. De otro modo, las líneas de los procesos en competencia se intercalarían.

Competencia entre procesos por recursos

La aplicación de la exclusión mutua crea dos problemas de control adicionales. Uno es el del **interbloqueo**. Por ejemplo, considere dos procesos, P1 y P2, y dos recursos, R1 y R2. Suponga que cada proceso necesita acceder a ambos recursos para realizar parte de su función. Entonces es posible encontrarse la siguiente situación: el sistema operativo asigna R1 a P2, y R2 a P1. Cada proceso está esperando por uno de los dos recursos. Ninguno liberará el recurso que ya posee hasta haber conseguido el otro recurso y realizado la función que requiere ambos recursos. Los dos procesos están interbloqueados.

Competencia entre procesos por recursos

Un último problema de control es la **inanición**. Suponga que tres procesos (P1, P2, P3) requieren todos accesos periódicos al recurso R. Considere la situación en la cual P1 está en posesión del recurso y P2 y P3 están ambos retenidos, esperando por ese recurso. Cuando P1 termine su sección crítica, debería permitírsele acceso a R a P2 o P3. Asúmase que el sistema operativo le concede acceso a P3 y que P1 solicita acceso otra vez antes de completar su sección crítica. Si el sistema operativo le concede acceso a P1 después de que P3 haya terminado, y posteriormente concede alternativamente acceso a P1 y a P3, entonces a P2 puede denegársele indefinidamente el acceso al recurso, aunque no suceda un interbloqueo.

Cooperación entre procesos vía compartición

El caso de cooperación vía compartición cubre **procesos que interaccionan con otros procesos sin tener conocimiento explícito de ellos**.

Por ejemplo, múltiples procesos pueden tener acceso a variables compartidas o a ficheros o bases de datos compartidas. Los procesos pueden usar y actualizar los datos compartidos sin referenciar otros procesos pero saben que otros procesos pueden tener acceso a los mismos datos.

Así, los procesos deben cooperar para asegurar que los datos que comparten son manipulados adecuadamente. Los mecanismos de control deben asegurar la integridad de los datos compartidos.

Cooperación entre procesos vía compartición

Dado que los datos están contenidos en recursos (dispositivos, memoria), los problemas de control de exclusión mutua, interbloqueo e inanición están presentes de nuevo. La única diferencia es que los **datos individuales pueden ser accedidos de dos maneras diferentes, lectura y escritura, y sólo las operaciones de escritura deben ser mutuamente exclusivas.**

Sin embargo, por encima de estos problemas, **surge un nuevo requisito: el de la coherencia de datos**. Como ejemplo sencillo, considérese una aplicación de contabilidad en la que pueden ser actualizados varios datos individuales. Supóngase que dos datos individuales a y b han de ser mantenidos en la relación a = b. Esto es, cualquier programa que actualice un valor, debe también actualizar el otro para mantener la relación.

Cooperación entre procesos vía compartición

Considérense ahora los siguientes dos procesos:

```
// P1:                  // P2:
a = a + 1;              b = 2 * b;
b = b + 1;              a = 2 * a;
```

Si el estado es inicialmente consistente, cada proceso tomado por separado dejará los datos compartidos en un estado consistente.

Ahora considere la siguiente ejecución concurrente en la cual los dos procesos respetan la exclusión mutua sobre cada dato individual (a y b).

```
a = a + 1;
b = 2 * b;
b = b + 1;
a = 2 * a;
```

Al final de la ejecución de esta secuencia, la condición a = b ya no se mantiene.

Por ejemplo, si se comienza con a = b = 1, al final de la ejecución de esta secuencia, tendremos a = 4 y b = 3. El problema puede ser evitado declarando en cada proceso la secuencia completa como una sección crítica.

Así se ve que el concepto de sección crítica es importante en el caso de cooperación por compartición.

Cooperación entre procesos vía comunicación

En los dos primeros casos que se han tratado, cada proceso tiene su propio entorno aislado que no incluye a los otros procesos. Las interacciones entre procesos son indirectas. En ambos casos hay cierta compartición. En el caso de la competencia, hay recursos compartidos sin ser conscientes de los otros procesos. En el segundo caso, hay compartición de valores y aunque cada proceso no es explícitamente consciente de los demás procesos, es consciente de la necesidad de mantener integridad de datos.

Cuando los procesos cooperan vía comunicación, en cambio, los diversos procesos involucrados participan en un esfuerzo común que los vincula a todos ellos. La comunicación proporciona una manera de sincronizar o coordinar actividades varias.

Típicamente, la comunicación se fundamenta en mensajes de algún tipo. Las primitivas de envío y recepción de mensajes deben ser proporcionadas como parte del lenguaje de programación o por el núcleo del sistema operativo.

Cooperación entre procesos vía comunicación

Dado que en el acto de pasar mensajes los procesos no comparten nada, la exclusión mutua no es un requisito de control en este tipo de cooperación. Sin embargo, los problemas de interbloqueo e inanición están presentes.

Como ejemplo de interbloqueo, dos procesos pueden estar bloqueados, cada uno esperando por una comunicación del otro.

Como ejemplo de inanición, considérense tres procesos, P1, P2 y P3, que muestran el siguiente comportamiento. P1 está intentando repetidamente comunicar con P2 o con P3, y P2 y P3 están ambos intentando comunicar con P1.

Podría suceder una secuencia en la cual P1 y P2 intercambiasen información repetidamente, mientras P3 está bloqueado esperando comunicación de P1. No hay interbloqueo porque P1 permanece activo, pero P3 pasa hambre.

Exclusión mutua: soporte hardware

Requisitos para la exclusión mutua

Cualquier mecanismo o técnica que vaya a proporcionar exclusión mutua debería cumplimentar los siguientes requisitos:

1. **La exclusión mutua debe hacerse cumplir**: sólo se permite un proceso al tiempo dentro de su sección crítica, de entre todos los procesos que tienen secciones críticas para el mismo recurso u objeto compartido.
2. **Un proceso que se pare en su sección no crítica debe hacerlo sin interferir con otros procesos**.
3. **No debe ser posible que un proceso que solicite acceso a una sección crítica sea postergado indefinidamente**: ni interbloqueo ni inanición.

Requisitos para la exclusión mutua

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 permanece dentro de su sección crítica sólo por un tiempo finito.**

Requisitos para la exclusión mutua

Hay varias maneras de satisfacer los requisitos para la exclusión mutua. Una manera es delegar la responsabilidad en los procesos que desean ejecutar concurrentemente. Esos procesos, ya sean programas del sistema o programas de aplicación, estarían obligados a coordinarse entre sí para cumplir la exclusión mutua, sin apoyo del lenguaje de programación ni del sistema operativo.

Podemos referirnos a esto como soluciones software. Aunque este enfoque es propenso a una alta sobrecarga de procesamiento y a errores, sin duda es útil examinar estas propuestas para obtener una mejor comprensión de la complejidad de la programación concurrente.

Un segundo enfoque es proporcionar cierto nivel de soporte dentro del sistema operativo o del lenguaje de programación.

Deshabilitar interrupciones

En una máquina monoprocesador, los procesos concurrentes no pueden solaparse, sólo pueden entrelazarse. Es más, un proceso continuará ejecutando hasta que invoque un servicio del sistema operativo o hasta que sea interrumpido. Por tanto, para garantizar la exclusión mutua, basta con impedir que un proceso sea interrumpido. Esta técnica puede proporcionarse en forma de primitivas definidas por el núcleo del sistema para deshabilitar y habilitar las interrupciones.

Deshabilitar interrupciones

Un proceso puede cumplir la exclusión mutua del siguiente modo:

```
while (true)
{
/* deshabilitar interrupciones */;
/* sección crítica */;
/* habilitar interrupciones */;
/* resto */;
}
```

Deshabilitar interrupciones

Dado que la sección crítica no puede ser interrumpida, se garantiza la exclusión mutua. El precio de esta solución, no obstante, es alto. La eficiencia de ejecución podría degradarse notablemente porque se limita la capacidad del procesador de entrelazar programas.

Un segundo problema es que **esta solución no funcionará sobre una arquitectura multiprocesador**. Cuando el sistema de cómputo incluye más de un procesador, es posible (y típico) que se estén ejecutando al tiempo más de un proceso. En este caso, deshabilitar interrupciones no garantiza exclusión mutua.

Instrucciones máquina especiales

En una configuración multiprocesador, varios procesadores comparten acceso a una memoria principal común. En este caso no hay una relación maestro/esclavo; en cambio los procesadores se comportan independientemente en una relación de igualdad. No hay mecanismo de interrupción entre procesadores en el que pueda basarse la exclusión mutua.

A un nivel hardware, como se mencionó, el acceso a una posición de memoria excluye cualquier otro acceso a la misma posición. Con este fundamento, los diseñadores de procesadores han propuesto varias instrucciones máquina que llevan a cabo dos acciones atómicamente, como leer y escribir o leer y comprobar, sobre una única posición de memoria con un único ciclo de búsqueda de instrucción.

Instrucciones máquina especiales

Durante la ejecución de la instrucción, el acceso a la posición de memoria se le bloquea a toda otra instrucción que referencie esa posición. Típicamente, estas acciones se realizan en un único ciclo de instrucción.

Instrucción Test and Set. La instrucción test and set (comprueba y establece)

Instrucción Exchange. La instrucción exchange (intercambio)

Propiedades de la solución instrucción máquina

El uso de una instrucción máquina especial para conseguir exclusión mutua tiene ciertas ventajas:

- Es aplicable a cualquier número de procesos sobre un procesador único o multiprocesador de memoria principal compartida.
- Es simple y, por tanto, fácil de verificar.
- Puede ser utilizado para dar soporte a múltiples secciones críticas: cada sección crítica puede ser definida por su propia variable.

Propiedades de la solución instrucción máquina

Hay algunas desventajas:

• **Se emplea espera activa**. Así, mientras un proceso está esperando para acceder a una sección crítica, continúa consumiendo tiempo de procesador.

• **Es posible la inanición**. Cuando un proceso abandona su sección crítica y hay más de un proceso esperando, la selección del proceso en espera es arbitraria. Así, a algún proceso podría denegársele indefinidamente el acceso.

• **Es posible el interbloqueo.** Considérese el siguiente escenario en un sistema de procesador único. El proceso P1 ejecuta la instrucción especial (por ejemplo, testset, exchange) y entra en su sección crítica. Entonces P1 es interrumpido para darle el procesador a P2, que tiene más alta prioridad. Si P2 intenta ahora utilizar el mismo recurso que P1, se le denegará el acceso, dado el mecanismo de exclusión mutua. Así caerá en un bucle de espera activa. Sin embargo, P1 nunca será escogido para ejecutar por ser de menor prioridad que otro proceso listo, P2.

Dados los inconvenientes de ambas soluciones software y hardware, es necesario buscar otros mecanismos.

Semáforos

Semáforos

El primer avance fundamental en el tratamiento de los problemas de programación concurrente ocurre en 1965 con el tratado de Dijkstra. Dijkstra estaba involucrado en el diseño de un sistema operativo como una colección de procesos secuenciales cooperantes y con el desarrollo de mecanismos eficientes y fiables para dar soporte a la cooperación. Estos mecanismos podrían ser usados fácilmente por los procesos de usuario si el procesador y el sistema operativo colaborasen en hacerlos disponibles.

Semáforos

El principio fundamental es éste: **dos o más procesos pueden cooperar por medio de simples señales, tales que un proceso pueda ser obligado a parar en un lugar específico hasta que haya recibido una señal específica.** Cualquier requisito complejo de coordinación puede ser satisfecho con la estructura de señales apropiada. Para la señalización, se utilizan unas variables especiales llamadas semáforos. Para transmitir una señal vía el semáforo s, el proceso ejecutará la primitiva semSignal(s).

Para recibir una señal vía el semáforo s, el proceso ejecutará la primitiva semWait(s); si la correspondiente señal no se ha transmitido todavía, el proceso se suspenderá hasta que la transmisión tenga lugar.

Semáforos

Para conseguir el efecto deseado, **el semáforo puede ser visto como una variable que contiene un valor entero** sobre el cual sólo están definidas tres operaciones:

1. Un semáforo puede ser inicializado a un valor no negativo.
2. La operación semWait decrementa el valor del semáforo. Si el valor pasa a ser negativo, entonces el proceso que está ejecutando semWait se bloquea. En otro caso, el proceso continúa su ejecución.
3. 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.

Aparte de estas tres operaciones no hay manera de inspeccionar o manipular un semáforo.

Semáforos

Una versión más restringida, conocida como semáforo binario o mutex que sólo puede tomar los valores 0 y 1 y se puede definir por las siguientes tres operaciones:

1. Un semáforo binario puede ser inicializado a 0 o 1.

2. La operación semWaitB comprueba el valor del semáforo. Si el valor es cero, entonces el proceso que está ejecutando semWaitB se bloquea. Si el valor es uno, entonces se cambia el valor a cero y el proceso continúa su ejecución.

3. La operación semSignalB comprueba si hay algún proceso bloqueado en el semáforo. Si lo hay, entonces se desbloquea uno de los procesos bloqueados en la operación semWaitB. Si no hay procesos bloqueados, entonces el valor del semáforo se pone a uno.

Semáforos

En principio debería ser más fácil implementar un semáforo binario, y puede demostrarse que tiene la misma potencia expresiva que un semáforo general.

Para contrastar los dos tipos de semáforos, el semáforo no-binario es a menudo referido como semáforo con contador o semáforo general.

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

Semáforo débil/fuerte

Para ambos, semáforos con contador y semáforos binarios, se utiliza una cola para mantener los procesos esperando por el semáforo. Surge la cuestión sobre el orden en que los procesos deben ser extraídos de tal cola. La política más favorable es FIFO (primero-en-entrar-primero-en-salir): el proceso que lleve más tiempo bloqueado es el primero en ser extraído de la cola; un semáforo cuya definición incluye esta política se denomina semáforo fuerte. Un semáforo que no especifica el orden en que los procesos son extraídos de la cola es un semáforo débil.

Los semáforos fuertes garantizan estar libres de inanición mientras que los semáforos débiles no.

Los semáforos fuertes son más convenientes y es la forma típica del semáforo proporcionado por los sistemas operativos.

Monitores

Monitores

Los semáforos proporcionan una herramienta potente y flexible para conseguir la exclusión mutua y para la coordinación de procesos.

Sin embargo, puede ser difícil producir un programa correcto utilizando semáforos. La dificultad es que las operaciones semWait y semSignal pueden estar dispersas a través de un programa y no resulta fácil ver el efecto global de estas operaciones sobre los semáforos a los que afectan.

Monitores

El monitor es una construcción del lenguaje de programación que proporciona una funcionalidad equivalente a la de los semáforos pero es más fácil de controlar. La construcción monitor ha sido implementada en cierto número de lenguajes de programación, incluyendo Pascal Concurrente, Pascal-Plus, Modula-2, Modula-3 y Java.

También ha sido implementada como una biblioteca de programa. Esto permite a los programadores poner cerrojos monitor sobre cualquier objeto. En concreto, para algo como una lista encadenada, puede quererse tener un único cerrojo para todas las listas, por cada lista o por cada elemento de cada lista.

Monitor con señal

Un monitor es un módulo software consistente en uno o más procedimientos, una secuencia de inicialización y datos locales.

Las principales características de un monitor son las siguientes:

1. Las variables locales de datos son sólo accesibles por los procedimientos del monitor y no por ningún procedimiento externo.
2. Un proceso entra en el monitor invocando uno de sus procedimientos.
3. Sólo un proceso puede estar ejecutando dentro del monitor al tiempo; cualquier otro proceso que haya invocado al monitor se bloquea, en espera de que el monitor quede disponible.

Monitor con señal

Las dos primeras características guardan semejanza con las de los objetos en el software orientado a objetos. De hecho, en un sistema operativo o lenguaje de programación orientado a objetos puede implementarse inmediatamente un monitor como un objeto con características especiales.

Al cumplir la disciplina de sólo un proceso al mismo tiempo, **el monitor es capaz de proporcionar exclusión mutua fácilmente.** Las variables de datos en el monitor sólo pueden ser accedidas por un proceso a la vez. Así, una estructura de datos compartida puede ser protegida colocándola dentro de un monitor. Si los datos en el monitor representan cierto recurso, entonces el monitor proporciona la función de exclusión mutua en el acceso al recurso.

Paso de mensajes

Paso de mensajes

Cuando los procesos interaccionan entre sí, deben satisfacerse dos requisitos fundamentales: sincronización y comunicación.

Los procesos necesitan ser sincronizados para conseguir exclusión mutua; los procesos cooperantes pueden necesitar intercambiar información. Un enfoque que proporciona ambas funciones es el paso de mensajes.

El paso de mensajes tiene la ventaja añadida de que se presta a ser implementado tanto en sistemas distribuidos como en multiprocesadores de memoria compartida y sistemas monoprocesador.

Los sistemas de paso de mensajes se presentan en varias modalidades.

Paso de mensajes

La funcionalidad real del paso de mensajes se proporciona normalmente en forma de un par de primitivas:

```
send(destino, mensaje)
```

```
receive(origen, mensaje)
```

Este es el conjunto mínimo de operaciones necesarias para que los procesos puedan entablar paso de mensajes.

El proceso envía información en forma de un mensaje a otro proceso designado por destino.

El proceso recibe información ejecutando la primitiva receive, indicando la fuente y el mensaje.

Características de diseño en sistemas de mensajes

Sincronización

- *Send*
  - Bloqueante
  - No bloqueante
- *Receive*
  - Bloqueante
  - No bloqueante
  - Comprobación de llegada

Formato

- Contenido
- Longitud
  - Fija
  - Variable

Direccionamiento

- Directo
  - *Send*
  - *Receive*
    - Explícito
    - Implícito
- Indirecto
  - Estático
  - Dinámico
  - Propiedad

Disciplina de cola

- FIFO
- Prioridad

Sincronización

La comunicación de un mensaje entre dos procesos implica cierto nivel de sincronización entre los dos: el receptor no puede recibir un mensaje hasta que no lo haya enviado otro proceso. En suma, tenemos que especificar qué le sucede a un proceso después de haber realizado una primitiva send o receive.

Considérese primero la primitiva send. Cuando una primitiva send se ejecuta en un proceso, hay dos posibilidades: o el proceso que envía se bloquea hasta que el mensaje se recibe o no se bloquea. De igual modo, cuando un proceso realiza la primitiva receive, hay dos posibilidades:

Sincronización

Así, ambos **emisor y receptor pueden ser bloqueantes o no bloqueantes**. Tres son las combinaciones típicas, si bien un sistema en concreto puede normalmente implementar sólo una o dos de las combinaciones:

- **Envío bloqueante**, recepción bloqueante. Ambos emisor y receptor se bloquean hasta que el mensaje se entrega; a esto también se le conoce normalmente como rendezvous.
- **Envío no bloqueante**, recepción bloqueante. Aunque el emisor puede continuar, el receptor se bloqueará hasta que el mensaje solicitado llegue. Esta es probablemente la combinación más útil.
- **Envío no bloqueante**, recepción no bloqueante. Ninguna de las partes tiene que esperar.

Direccionamiento

Claramente, es necesario tener una manera de especificar en la primitiva de envío qué procesos deben recibir el mensaje.

Los diferentes esquemas para especificar procesos en las primitivas send y receive caben dentro de dos categorías: direccionamiento directo y direccionamiento indirecto.

Con el direccionamiento directo, la primitiva send incluye un identificador específico del proceso destinatario. La primitiva receive puede ser manipulada de dos maneras. Una posibilidad es que el proceso deba designar explícitamente un proceso emisor. Así, el proceso debe conocer con anticipación de qué proceso espera el mensaje. Esto suele ser lo más eficaz para procesos concurrentes cooperantes.

Direccionamiento

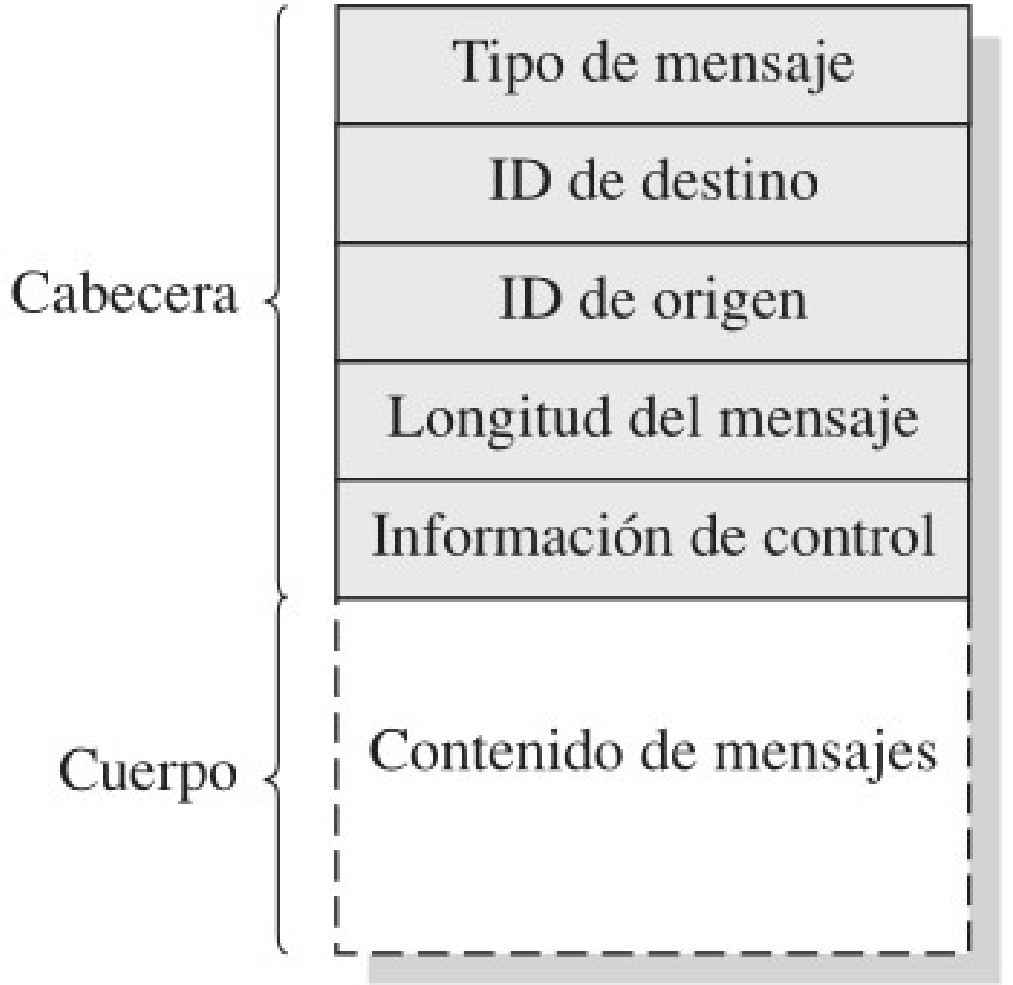
El otro esquema general es **el direccionamiento indirecto, donde los mensajes no se envían directamente por un emisor a un receptor sino que son enviados a una estructura de datos compartida que consiste en colas que pueden contener mensajes temporalmente**. Tales colas se conocen generalmente como buzones (mailboxes). Así, para que dos procesos se comuniquen, un proceso envía un mensaje al buzón apropiado y otro proceso toma el mensaje del buzón.

Formato de mensaje

El formato del mensaje depende de los objetivos de la facilidad de mensajería y de cuándo tal facilidad ejecuta en un computador único o en un sistema distribuido.

En algunos sistemas operativos, los diseñadores han preferido mensajes cortos de longitud fija para minimizar la sobrecarga de procesamiento y almacenamiento. Si se va a transferir una gran cantidad de datos, los datos pueden estar dispuestos en un archivo y el mensaje puede simplemente indicar el archivo. Una solución más sencilla es permitir mensajes de longitud variable.

Formato de mensaje



Exclusión mutua

Se asume el uso de la primitiva receive bloqueante y de la primitiva send no bloqueante. Un conjunto de procesos concurrentes comparten un buzón que pueden usar todos los procesos para enviar y recibir. El buzón se inicializa conteniendo un único mensaje de contenido nulo. El proceso que desea entrar en su sección crítica primero intenta recibir un mensaje.

Si el buzón está vacío, el proceso se bloquea. Cuando el proceso ha conseguido el mensaje, realiza su sección crítica y luego devuelve el mensaje al buzón. Así, el mensaje se comporta como un testigo que va pasando de un proceso a otro.

Exclusión mutua

La solución precedente asume que si más de un proceso realiza la operación de recepción concurrentemente, entonces:

- Si hay un mensaje, se le entregará solo a uno de los procesos y los otros se bloquearán
- Si la cola de mensajes está vacía, todos los procesos se bloquearán; cuando haya un mensaje disponible solo uno de los procesos se activará y tomará el mensaje.

Estos supuestos son prácticamente ciertos en todas las facilidades de paso de mensajes.

Exclusión mutua usando mensajes

```
/* programa exclusión mutua */
const int n = /* número de procesos */;
void P(int i)
{
     message carta;
     while (true)
     {
          receive (buzon, carta);
          /* sección crítica */;
          send (buzon, carta);
          /* resto */;
     }
}
void main()
{
          create_mailbox (buzon);
          send (buzon, null);
          paralelos (P(1), P(2), . . ., P(n));
}
```

Bibliografía

- **Tanenbaum, A. S. (2009)**. Sistemas operativos modernos (3a ed.) (pp. 1-18). México: Pearson Educación.
- **Stallings, W. (2005)**. Sistemas operativos (5a ed.) (pp. 54-67). Madrid: Pearson Educación.