Tema 3: Sincronización

1. Sincronización entre procesos.

1. Regiones críticas.

Es el conjunto de instrucciones en las que un proceso accede a un recurso compartido y se ejecutarán de forma exclusiva con respecto a otros procesos.

Al final de cada sección crítica, el recurso debe ser liberado para que puedan utilizarlo otros procesos.

1. Categoría de proceso cliente-suministrador.

Es un proceso que requiere o solicita información o servicios que proporciona otro proceso.

Entre un cliente y un suministrador (ojo, empecemos con un proceso de cada), se establece sincronismo entre ellos, por medio de intercambio de mensajes o a través de un recurso compartido.

Entre procesos cliente y suministrador debemos disponer de mecanismos de sincronización que permitan que:

- Un cliente no debe poder leer un dato hasta que no haya sido completamente suministrado.
- Un suministrador irá produciendo su información, que en cada instante, no podrá superar un volumen de tamaño máximo establecido.

2. Semáforos.

Un semáforo, es un componente de bajo nivel de abstracción que permite arbitrar los accesos a un recurso compartido en un entorno de programación concurrente.

Dependiendo del conjunto de datos que pueda tomar un semáforo, tendremos:

- **Semáforos binarios:** Aquellos que pueden tomar solo valores 0 ó 1.
- **Semáforos generales:** Pueden tomar cualquier valor Natural (entero no negativo).

En cualquier caso, los valores que toma un semáforo representan:

- **Valor igual a 0:** Indica que el semáforo está cerrado.
- **Valor mayor de 0:** El semáforo está abierto.

Cualquier semáforo permite dos operaciones seguras:

- **objSemaforo.wait():** Si el semáforo no es nulo (está abierto) decrementa en uno el valor del semáforo. Si el valor del semáforo es nulo (está cerrado), el proceso que lo ejecuta se suspende y se encola en la lista de procesos en espera del semáforo.

- **objSemaforo.signal():** Si hay algún proceso en la lista de procesos del semáforo, activa uno de ellos para que ejecute la sentencia que sigue al wait que lo suspendió. Si no hay procesos en espera en la lista incrementa en 1 el valor del semáforo.

Para utilizar semáforos, seguiremos los siguientes pasos:

1. Un proceso padre creará e inicializará el semáforo.
2. El proceso padre creará el resto de procesos hijo pasándoles el semáforo que ha creado. Esos procesos hijos acceden al mismo recurso compartido.
3. Cada proceso hijo, hará uso de las operaciones seguras wait y signal respetando este esquema:
   - **objSemaforo.wait();** Para consultar si puede acceder a la sección crítica.
   - **Sección crítica;** Instrucciones que acceden al recurso protegido por el semáforo objSemaforo.
   - **objSemaforo.signal();** Indicar que abandona su sección y otro proceso podrá entrar.

3. Monitores.

Los monitores, nos ayudan a resolver las desventajas que encontramos en el uso de semáforos. El problema en el uso de semáforos es que, recae sobre el programador o programadora la tarea implementar el correcto uso de cada semáforo para la protección de cada recurso compartido y sigue estando disponible el recurso para utilizarlo sin la protección de un semáforo.

Un monitor, es un componente de alto nivel de abstracción destinado a gestionar recursos que van a ser accedidos de forma concurrente.

Los monitores encierran en su interior los recursos o variables compartidas como componentes privadas y garantizan el acceso a ellas en exclusión mutua.

Las ventajas que proporciona el uso de monitores son:

- **Uniformidad:** El monitor provee una única capacidad, la exclusión mutua. No existe la confusión de los semáforos.
- **Modularidad:** El código que se ejecuta en exclusión mutua está separado, no mezclado con el resto del programa.
- **Simplicidad:** El programador o programadora no necesita preocuparse de las herramientas para la exclusión mutua.
- **Eficiencia de la implementación:** La implementación subyacente puede limitarse fácilmente a los semáforos.

Y, la desventaja:

- **Interacción de múltiples condiciones de sincronización:** Cuando el número de condiciones crece, y se hacen complicadas, la complejidad del código crece de manera extraordinaria.

4. Cola de mensajes.

El paso de mensajes es una técnica empleada en programación concurrente para aportar sincronización entre procesos y permitir la exclusión mutua, de manera similar a como se hace con los semáforos, monitores, etc. Su principal característica es que no precisa de memoria compartida.

Los elementos principales que intervienen en el paso de mensajes son el proceso que envía, el que recibe y el mensaje.

Dependiendo de si el proceso que envía el mensaje espera a que el mensaje sea recibido, se puede hablar de paso de mensajes síncrono o asíncrono:

- En el paso de mensajes asíncrono, el proceso que envía, no espera a que el mensaje sea recibido, y continúa su ejecución, siendo posible que vuelva a generar un nuevo mensaje y a enviarlo antes de que se haya recibido el anterior.
- En el paso de mensajes síncrono, el proceso que envía el mensaje espera a que un proceso lo reciba para continuar su ejecución.

2. Requisitos: seguridad, vivacidad, eficiencia y reusabilidad.

Todo programa concurrente debe satisfacer dos tipos de propiedades:

- **Propiedades de seguridad:** En cada instante de la ejecución no debe haberse producido algo que haga entrar al programa en un estado erróneo:
  - Dos procesos no deben entrar simultáneamente en una sección crítica.
  - Se respetan las condiciones de sincronismo.
- **Propiedades de vivacidad:** Cada sentencia que se ejecute conduce en algún modo a un avance constructivo para alcanzar el objetivo funcional del programa:
  - No deben producirse bloqueos activos (livelock).
  - Aplazamiento indefinido (starvation): Consiste en el estado al que puede llegar un programa que aunque potencialmente puede avanzar de forma constructiva.
  - Interbloqueo (deadlock): se produce cuando los procesos no pueden obtener los recursos necesarios para finalizar su tarea.

Es evidentemente, que también nos preocuparemos por diseñar nuestras aplicaciones para que sean eficientes:

- No utilizarán más recursos de los necesarios.
- Buscaremos la rigurosidad en su implementación: toda la funcionalidad esperada de forma correcta y concreta.

En cuanto a la reusabilidad, debemos tenerlo ya, muy bien aprendido:

- Implementar el código de forma modular: definiendo clases, métodos, funciones…
- Documentar correctamente el código y el proyecto.

1. ***Dificultades en la depuración.***

   Nos damos cuenta de la complejidad que entraña depurar el comportamiento de aplicaciones concurrentes, es por ello, que al diseñarlas, tendremos en cuenta los patrones de diseño, que ya están diseñados resolviendo errores comunes de la concurrencia.

3. Programación paralela y distribuida.

Dos procesos se ejecutan de forma paralela, si las instrucciones de ambos se están ejecutando realmente de forma simultánea. Esto sucede en la actualidad en sistemas que poseen más de un núcleo de procesamiento.

Tanto en la programación paralela como distribuida, existe ejecución simultánea de tareas que resuelven un problema común. La diferencia entre ambas es:

- La programación paralela se centra en microprocesadores multinúcleo (en nuestros PC y servidores), o sobre los llamados supercomputadores, compuestos por gran cantidad de equipos idénticos interconectados entre sí, y que cuentan son sistemas operativos propios.
- La programación distribuida se centra en sistemas formados por un conjunto de ordenadores heterogéneos interconectados entre sí por redes de comunicaciones de propósito general: redes de área local, metropolitana; incluso, a través de Internet.

En la computación paralela y distribuida:

- Cada procesador tiene asignada la tarea de resolver una porción del problema.
- En programación paralela, los procesos pueden intercambiar datos, a través de direcciones de memoria compartidas o mediante una red de interconexión propia.
- En programación distribuida, el intercambio de datos y la sincronización se realizará mediante intercambio de mensajes.
- El sistema se presenta como una unidad ocultando la realidad de las partes que lo forman.