Lenguajes de Programación

Concurrencia

Ma. Laura Cobo

Universidad Nacional del Sur
Departamento de Ciencias e Ingeniería de la Computación
2018

Motivación

Un programa se dice concurrente si puede tener mas de un contexto de ejecución activo.

Motivaciones generales:

- Capturar la estructura lógica del problema
- Aprovechar los procesadores (para maximizar velocidad de computo)
- para hacer frente a dispositivos físicos separados

En general la concurrencia caracteriza sistemas en los cuales dos o mas tareas están en ejecución al mismo tiempo (se intercalan en puntos de ejecución no predecibles)

2

Niveles de paralelismo

Es inherente a cada nivel de los sistemas computacionales.

Cuanto mas alto es el nivel de abstracción mas difícil se vuelve explotar esta característica.

Están los procesadores y las capas de software que corren sobre ellos, la granularidad del paralelismo (tamaño y complejidad de las tareas) aumenta en cada nivel y se vuelva mas difícil determinar como un trabajo debe ser realizado (responsabilidad de cada tarea y coordinación)

En general la concurrencia caracteriza sistemas en los cuales dos o mas tareas están en ejecución al mismo tiempo (se intercalan en puntos de ejecución no predecibles)

3

Niveles de abstracción

El programador necesita entender concurrencia con diferentes niveles de detalles y utilizarla de maneras diferentes.

- Modo "caja negra": es el mas abstracto, la concurrencia se incorpora al lenguaje a través de librerías.
- Modo intermedio: requiere el conocimiento de que ciertas tareas son mutuamente independientes. Se incorpora en el lenguaje a través de constructores dedicados y habilidades de sincronización.
- Modo experto: el programador puede necesitar una comprensión detallada de hardware y sistemas de run-time para implementar los mecanismos de sincronización

4

Procesos e hilos

El uso de estos dos términos no es consistente. Los diferentes lenguajes y autores utilizan los términos con semánticas diferentes.

Una tarea o proceso es una unidad que puede ser ejecutada en forma concurrente con otras unidades

Características que diferencian una *tarea* de una *unidad ordinaria*:

- Una tarea puede ser comenzada en forma implícita
- Cuando una unidad comienza la ejecución de una tarea, no necesita suspenderse necesariamente
- Cuando se completa la ejecución de una tarea el control puede o no retornar a la unidad que comenzó su ejecución

5

Concurrencia a nivel unidad: categorías

Categorías de tareas:

- Heavyweight task: se ejecutan en su propio espacio de direcciones y tienen sus propias pilas de ejecución
- Lightweight task: corren todas en el mismo espacio de direcciones y utilizan la misma pila de ejecución

Si una tarea no se comunica, ni afecta la ejecución de otras tareas (de ninguna manera) se dice *disjunta*.

Aspectos fundamentales de la programación concurrente

- Comunicaciones y sincronización
  - Comunicación: Mecanismo que permite a una tarea o hilo conseguir información producida por otra/o. en los lenguajes imperativos esta generalmente basada en pasaje de mensajes o memoria compartida.
  - Sincronización: Mecanismo que permite al programados controlar el orden relativo en el las operaciones tienen en lugar en diferentes tareas o hilos.

7

Comunicación y sincronización

Son conceptos cruciales para la programación concurrente

La *comunicación* se refiere a cualquier mecanismo que permite a un hilo-tarea obtener información producida por otroa).

Los mecanismos de comunicación para programas imperativos están basados en:

- Memoria compartida: algunas de las variables son accesibles para varios hilos-tareas.
- Pasaje de mensajes: los hilos-tareas no tienen estado común. Para comunicarse uno debe realizar un pedido en forma explicita (send) para transmitir datos a otro

8

Comunicación y sincronización

La *sincronización* se refiere a los mecanismos que permiten al programador controlar el orden relativo en el cual las tareas-hilos se ejecutan (implica alguna forma de pasaje de mensajes)

Tipos de sincronización:

- Sincronización cooperativa:

  La tarea A debe esperar que la tarea B complete una actividad específica antes que la tarea A puede continuar si ejecución.

  Ejemplo el problema productor-consumidor

- Sincronización competitiva:

  Dos o mas tareas deben usar un recurso que no puede ser utilizado en forma simultánea.

  Ejemplo: contador compartido

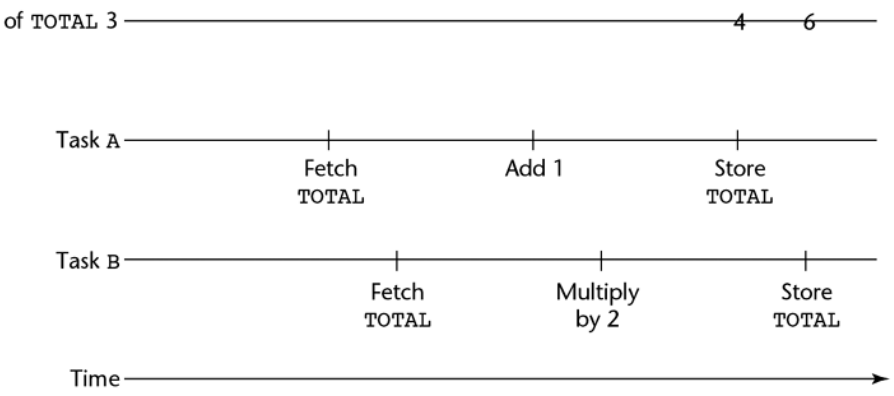
  La competición es usualmente provista por acceso mutuamente excluyente 9

Sincronización

La sincronización en general no puede darse en forma implícita en un contexto de memoria compartida

Figure 13.1

The need for competition synchronization



10

Semáforos

Un *semáforo* es una estructura de datos, que consiste de un contador y una cola para almacenar los descriptores de las tareas

Observación

- Los semáforos fueron desarrollados por Dijkstra en 1965
- Pueden utilizarse para implementar guardas sobre el código que accede a estructuras de datos compartidas.
- Solo utilizan dos operaciones *wait* and *release*.
- Se pueden utilizar tanto para sincronización cooperativa, como para sincronización competitiva

11

Evaluación de los semáforos

El mal uso de los semáforos puede causar fallas en la sincronización cooperativa. Puede producirse un overflow en el buffer si se omite accidentalmente el *wait* del semáforo *fullspots*.

De la misma manera pueden producirse fallas en la sincronización competitiva, conduciendo a deadlocks en caso de omitirse el release del semáforo.

"Los semáforos son una herramienta de sincronización elegante ideal para los programadores que no cometen errores" (Brinch Hansen, 1973)

12

Monitores

La idea es encapsular los datos compartidos y sus operaciones para restringir el acceso.

Un monitor es un tipo de dato abstracto para datos compartidos

Muchos lenguajes proveen la facilidad de definir monitores, algunos de ellos son: Concurrent Pascal, Modula, Mesa, Ada, Java, C#

13

Monitores: sincronización competitiva

Los datos están ahora dentro del monitor en lugar de en las unidades clientes

Todos los accesos a los datos dentro del monitor se realizan a través de sus operaciones

- La implementación del monitor asegura el acceso sincronizado, permitiendo solo una acceso a la vez.
- Las llamadas a las unidades del monitor son implícitamente encoladas si el monitor está ocupado en el momento de realizar la llamada.

Monitores: sincronización cooperativa

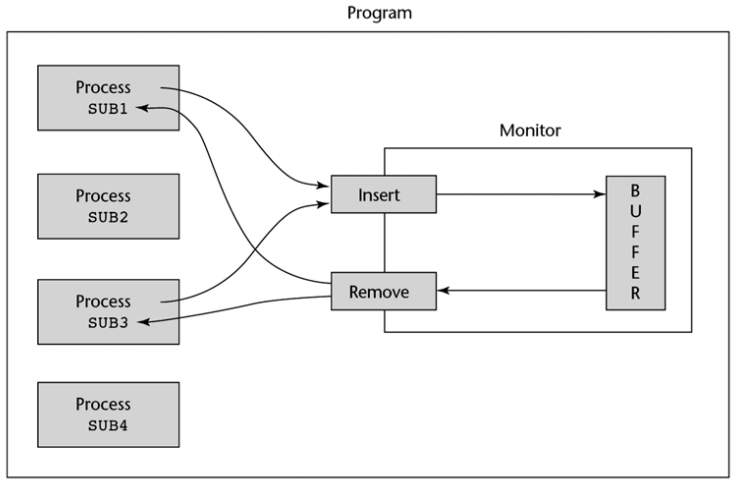
La cooperación entre procesos sigue siendo un trabajo de programación.

El programador debe seguir garantizando, en la implementación, que no se produzca overflow o underflow del buffer (en el ejemplo anterior)

Los lenguajes proveen diferentes maneras de realizar sincronización cooperativa, todos terminan estando relacionadas con los semáforos.

Figure 13.2

A program using a monitor to control access to a shared buffer



15

Monitores: evaluación

- Brindan un mejor soporte para sincronización competitiva que los semáforos
- Los semáforos pueden utilizarse para implementar monitores
- Los monitores se pueden utilizar para implementar semáforos
- El soporte para sincronización cooperativa es muy similar al que se provee con semáforos, por lo que tiene los mismo inconvenientes.

16

Pasaje de mensajes

- El pasaje de mensajes es un modelo general para concurrencia
  - Puede modelar tanto semáforos como monitores.
  - No es solo para sincronización competitiva.

- Idea central: la comunicación entre tareas es como una persona esperando por una llamada importante. A veces no queda otra que sentarse y esperar. En el caso de las tareas, si la tarea A quiere enviarle un mensaje a la tarea B, y B está esperando el mensaje, el mensaje es enviado. Esta transmisión se conoce como **rendezvous**.
- Para que el **rendezvous** se produzca el emisor y el receptor del mensaje deben querer que la comunicación se produzca.

17

Pasaje de mensajes: rendezvous

- Un mecanismo que permite a la tarea indicar cuando esta disponible para aceptar mensajes.
- Las tareas necesitan algún modo de recordar quién está esperando por sus mensajes y una manera "justa" de elegir el próximo mensaje.
- El rendezvous se produce en la transmisión del mensaje; cuando el mensaje de la tarea emisora es aceptado por la tarea receptora

Ejemplificaremos el pasaje de mensajes con el lenguaje Ada.

Ada 83 soporta pasaje de mensajes sincrónicos, Ada 95 agrega soporte para pasaje de mensaje asincrónico.

18

Aspectos fundamentales de la programación concurrente

- Lenguajes y librerías

  La concurrencia a nivel hilo puede proveerse en forma de:

  - Lenguajes concurrente explicito
  - Extensiones soportadas a nivel compilador
  - Paquete librería ajenos a la definición del lenguaje

19

Lenguajes y Librerías

La concurrencia puede brindarse al programador en forma de:

- *Lenguaje concurrente:* La concurrencia está considerada en el diseño del lenguaje.
- *Extensiones a los lenguajes secuencias tradicionales soportados por el compilador:* El lenguaje no se diseño con esa característica pero se agregó a un nivel que resulta tranparente para el programador.
- *Librerías o paquetes fuera del lenguaje propiamente dicho:* el lenguaje no considera esta característica en su diseño, ni en su implementación

20

Creación de hilos-tareas

Es fácil imaginar a un programa concurrente como una collección de hilos-tareas creada por la implementación del lenguaje.

Sin embargo esta aproximación es estática y como forma de concurrencia muy restrictiva

Un lenguaje de programación concurrente permite al programador crear hilos-tareas en tiempo de ejecución. La sintaxis y semántica varia considerablemente de lenguaje en lenguaje .

21

Creación de hilos-tareas

Las opciones más comunes son:

- **Co-begin:** tiene el comportamiento del begin-end pero para indicar que lo ubicado en ese bloque puede ejecutarse concurrente mente (ej par begin – par end)
- **Ciclos paralelos:** las iteraciones de estos ciclos pueden darse en forma concurrentes
- **Launch-at-elaboration:** el código del hilo puede ser declarado con una sintaxis que semeja la declaración de una unidad sin parámetros. Cuando se elabora la declaración se crea el hilo para ejecutar el código

22

Creación de hilos-tareas

Las opciones más comunes son:

- **fork:** Esta aproximación es mas general: hace la creación en forma explicita a través de una sentencia ejecutable.
  - **Recepción implícita**
  - **Respuesta temprana**

23

Concurrencia a nivel unidad: diseño

Aspectos de diseño

- Sincronización competitiva y cooperativa
- Cómo controlar la planificación de tareas
- Como y cuando comenzar las tareas y terminar su ejecución
- Como y cuando son creadas las tareas

Tres de las soluciones alternativas a los problemas de diseño para concurrencia:

- Semáforos
- Monitores y
- Pasaje de mensajes

24

Concurrencia a nivel unidad: propiedades

Liveness o *vivacidad* significa que si un evento (terminación del programa en forma completa) se supone que suceda, eventualmente ocurrirá

En programación secuencial significa que eventualmente la unidad se completa.

En programación concurrente, las tareas fácilmente pierden la propiedad de *vivacidad*.

Si todas las tareas en un ambiente concurrente pierden su *vivacidad*, el sistema se encuentra en deadlock

25