UNIVERSIDAD AMERICANA

Programación Concurrente

Recopilación de teoría referente a la materia

Ing. Luis Müller

Esta es una recopilación de la teoría referente a la asignatura Programación Concurrente, a ser estudiada en clases con los alumnos, y que servirá como base para su aplicación también en clases en ejercicios prácticos en el lenguaje de programación Java.

Contenido

Introducción 3
Sistemas Concurrentes 3
Objetivos 4
Ejecución concurrente 4
Núcleo de ejecución (RTS) 5
Procesos e hilos 5
Lenguajes concurrentes 6
Clasificación de los procesos 6
Procesos anidados 7
Los procesos y la programación orientada a objetos 7
Representación de procesos 8
Declaración explícita de procesos 8
Sincronización 8
Exclusión mutua y Sincronización condicional 9
Sincronización en Java 9

Programación Concurrente y Sincronización

Introducción

• La mayoría de los sistemas de tiempo real son concurrentes por naturaleza

• Las actividades del entorno evolucionan en paralelo

- Paralelismo físico

• Programación concurrente:

- Marco para expresar de forma abstracta la ejecución de actividades potencialmente paralelas
- y sin ocuparse de los detalles de implementación

Sistemas Concurrentes

• Un sistema concurrente se compone de un conjunto de procesos autónomos que se ejecutan, desde un punto de vista lógico, en paralelo

• Los lenguajes de programación concurrente incorporan la noción de proceso:

- Cada proceso tiene un flujo de control independiente
- Las instrucciones de los procesos se ejecutan intercalándose unas con otras

• Paralelismo lógico

Objetivos

• Ilustrar los principios de la programación concurrente

• Repasar los diferentes modelos de creación y ejecución de procesos concurrentes

• Mostrar cómo crear procesos concurrentes en Java y C/POSIX

Ejecución concurrente

• En la práctica, la implementación de una colección de procesos puede tomar diversas formas:

- Multiprogramación

  • si multiplexan sus ejecuciones en un solo procesador

- Multiproceso

  • en un multiprocesador con memoria compartida (fuertemente acoplado)

- Procesamiento distribuido

  •en un sistema con varios procesadores que no comparten memoria (débilmente acoplado)

Núcleo de ejecución (RTS)

estrategia en tiempo real o **RTS** (siglas en inglés de *real-time strategy*)

• Los procesos concurrentes se ejecutan con la ayuda de un núcleo de ejecución (run-time support system)

  – Planificador (scheduler) del sistema operativo

• Se encarga de la creación, terminación y multiplexado de los procesos

• Opciones del núcleo:

  – Desarrollado como parte de la aplicación (Modula-2)
  – Incluido en el entorno de ejecución del lenguaje (Ada, Java)
  – Parte de un sistema operativo de tiempo real (POSIX)
  – Microprogramado en el procesador (occam2)

• El método de planificación utilizado afecta al comportamiento temporal del sistema

Procesos e hilos

• Todos los sistemas operativos soportan procesos

  – Cada proceso se ejecuta en una máquina virtual distinta

• Algunos sistemas operativos soportan procesos ligeros (hilos o threads)

  – Todos los hilos de un proceso comparten la misma máquina virtual
  – Tienen acceso al mismo espacio de memoria
  – El programador o el lenguaje deben proporcionar mecanismos para evitar interferencias
  – La concurrencia puede estar soportada por

• el lenguaje: Java, Ada, occam2

Lenguajes concurrentes

• Tienen elementos de lenguaje para:

  – Crear procesos concurrentes
  – Sincronizar su ejecución
  – Comunicarse entre sí

• Las procesos pueden:

  – Ser independientes
  – Cooperar para un fin común
  – Competir por el uso de los recursos

• Si cooperan o compiten necesitan comunicarse y/o sincronizar sus actividades

Clasificación de los procesos

• Estructura:

  – Estática
  – Dinámica

• Granularidad

  – Gruesa
  – Fina

• Iniciación:

  – Paso de parámetros

• Terminación

  – Al completar la ejecución
  – Auto-terminación
  – Aborto explícito
  – Excepción no prevista
  – Si ya no es necesario
  – Nunca

Procesos anidados

• Es posible establecer jerarquías de procesos y establecer relaciones entre procesos

• Relación padre-hijo:

- Un proceso crea otro proceso
- El padre puede tener que esperar mientras el hijo es creado e inicializado

• Relación tutor-pupilo:

- (guardian-dependent)
- Un proceso es afectado por la terminación de otro proceso
- El tutor no puede terminar hasta que lo hayan hecho todos sus pupilos

Los procesos y la programación orientada a objetos

• Objetos activos:

- Ejecutan acciones espontáneamente
- Contienen uno o más hilos

• Objetos reactivos:

- Sólo ejecutan acciones cuando son invocados por otros objetos
- Objetos pasivos:

• Sin control de acceso (se ejecutan sin restricciones)

- Recursos:

• Con control de acceso según su estado interno

• Necesitan un agente de control:

- Recursos protegidos: agente pasivo

» Ej: semáforo

- Servidores: agente activo

Representación de procesos

• Hay varias formas de expresar la ejecución concurrente en los lenguajes de programación:

- Corrutinas
- Bifurcación y unión (fork/join)
- cobegin/coend
- Declaración explícita de procesos
- Declaración implícita de procesos (ej: Ada)

Declaración explícita de procesos

•Los procesos son unidades de programa (como los procedimientos)

•Esto permite que la estructura del programa sea más clara

•Nótese que esto no indica cuándo se ejecutarán

•Ejemplos: Java, Ada, C/POSIX

Sincronización

• El comportamiento correcto de un programa concurrente

- formado por procesos que compiten y/o cooperan
- depende de la sincronización y comunicación entre dichos procesos

• La comunicación es el paso de información entre procesos

• La sincronización es la satisfacción de restricciones en el orden en que se ejecutan algunas de sus acciones

• Ambos conceptos están relacionados

• Formas de abordarlos:

- Datos compartidos
- Mensajes

Exclusión mutua y Sincronización condicional

• Una secuencia de instrucciones que se debe ejecutar de forma aparentemente indivisible se denomina región crítica

  – las operaciones de dos regiones críticas sobre unos mismos datos no se entremezclan
  – la forma de sincronización que se usa para proteger una región crítica se llama exclusión mutua

• suponemos que el acceso a una celda individual de memoria es atómico

• Cuando una acción de un proceso sólo se puede realizar de forma segura si otros procesos están en determinado estado o han ejecutado ciertas acciones, se usa la sincronización condicional

• Ej: productor/consumidor con búfer limitado

  – No se debe hacer Put cuando el búfer está lleno
  – No se debe hacer Get cuando el búfer está vacío
  – Además, hay exclusión mutua en el acceso al búfer

Sincronización en Java

• Exclusión mutua:

- en Java, cada objeto tiene asociado un bloqueo

  • que no es accesible directamente desde el programa

  • pero se utiliza de forma indirecta mediante:

- el modificador de método synchronized

  » sólo se puede acceder al método si se ha obtenido el bloqueo asociado con el objeto

  » los métodos tiene acceso mutuamente exclusivo a los datos encapsulados por el objeto

- la sincronización de bloque

• Sincronización condicional:

public void wait ();

• bloquea al hilo invocador y libera el bloqueo sobre el objeto

public void notify ();

• despierta a un hilo de los que esperan

• pero no libera el bloqueo

public void notifyAll ();

• despierta a todos los hilos que esperan

– los 3 métodos sólo se pueden llamar desde métodos que mantienen el objeto bloqueado

• si no, lanzan la excepción IllegalMonitorStateException