Tema 1. Introducción a la Programación Concurrente

Jorge García Duque
http://www.det.uvigo.es/~jgd
Depto. Enxeñería Telemática
Universidad de Vigo

Conceptos Básicos (I)

- Programa:
  - Conjunto de declaraciones, asignaciones y sentencias de control en un lenguaje de programación.
  - El compilador las traduce a instrucciones en código máquina que se ejecutan ***secuencialmente***.
- Proceso:
  - Instancia de un programa en ejecución.
  - Ejecución secuencial del conjunto de instrucciones atómicas que forman un programa.
- Paralelismo: conjunto de procesos que se ejecutan simultáneamente, cada uno en un procesador diferente.
- Sistema Distribuido: procesamiento paralelo en ordenadores diferentes.
- ***Concurrencia***: paralelismo "*potencial":*
  - Ejecución *entrelazada arbitrariamente* de las instrucciones atómicas de cada uno de los procesos.
  - Multitarea: característica de un sistema operativo que permite la ejecución concurrente de varios procesos compartiendo uno o varios procesadores.
  - Escenario (*computation):* ejecución o entrelazamiento concreto.

¿Es suficiente con la implementación de un scheduler para soportar la programación concurrente?

Conceptos Básicos (II)

- ¿Por qué ejecutar procesos concurrentemente?
  - Procesos independientes: ventajas de un sistema en tiempo compartido.
  - Cooperación entre procesos concurrentes:
    - Sincronización: el avance en la ejecución de un proceso depende de otro/s.
    - Comunicación: la ejecución de un proceso necesita datos aportados por otro/s procesos.

¿Ejemplos? Puede haber sincronización sin comunicación, pero ¿puede haber comunicación sin sincronización?

- ¿Qué necesitamos?:
  - Identificar las partes de los procesos que no son independientes: secciones críticas.
  - Herramientas de programación concurrente que permitan sincronizar y comunicar procesos.
  - Soluciones válidas para todo entrelazamiento arbitrario (*escenario)*: algoritmos y protocolos para sincronizar y comunicar procesos concurrentes.

    Asumiremos la existencia de memoria compartida: varios procesos pueden acceder a una misma posición de memoria (variables de color rojo en los ejemplos)

Instrucciones Atómicas

- Ejecución concurrente: entrelazamiento arbitrario de las instrucciones atómicas de varios procesos.

```
Process P() {
    int k = 1;

    int r = 2;
p1: n = k;
p2: n = r;
}
```

```
Process Q() {
    int m = 2;


q1: n = m;
}
```

- Instrucciones atómicas: si dos se ejecutaran simultáneamente el resultado sería el mismo que si se ejecutan secuencialmente en cualquier orden.

¿Puede considerarse un autoincremento [n = n + 1] una instrucción atómica?

- Asumiremos como instrucciones atómicas: asignaciones y comprobaciones de condiciones booleanas en sentencias de control.

Grafos de Precedencia (I)

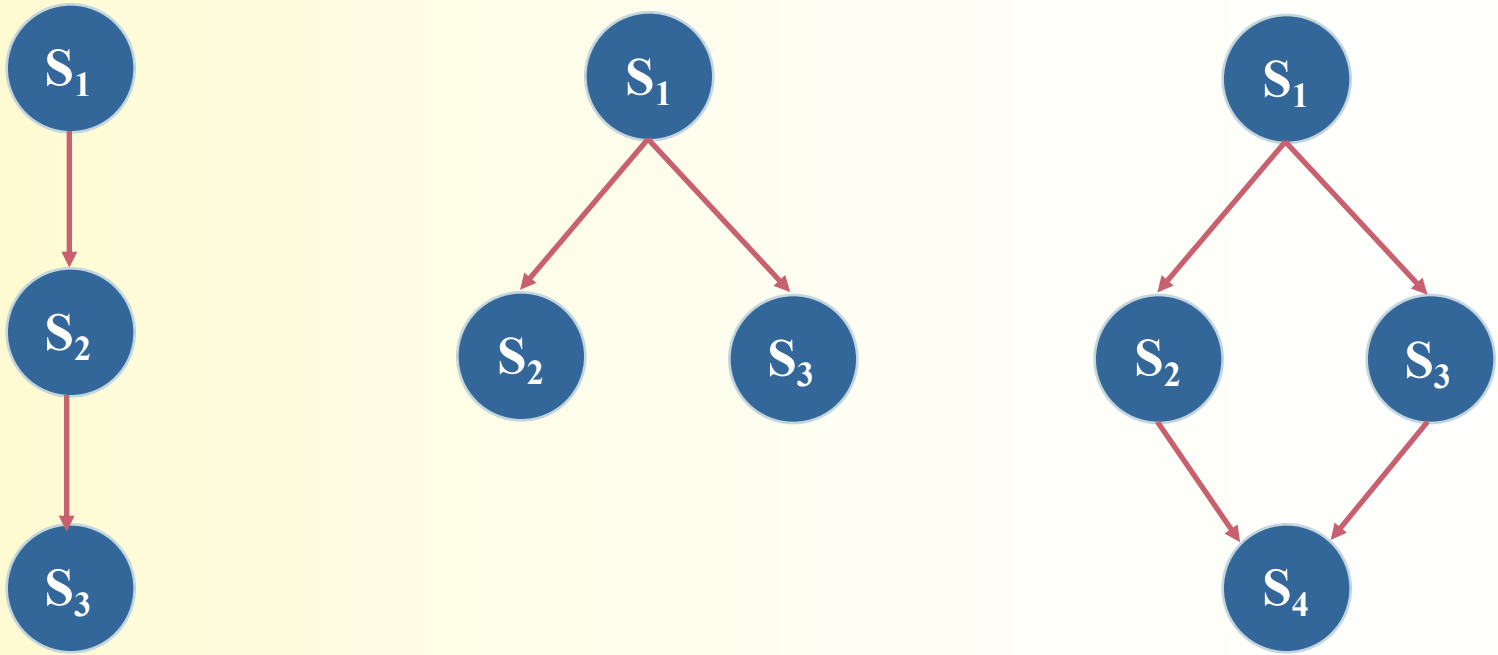
- Diseño de un sistema con varios procesos concurrentes:
  - Determinar el conjunto de tareas a llevar a cabo: especificación del problema.
  - Determinar qué conjunto de tareas son secuenciales y cuáles son concurrentes: ¿cómo se expresa?
    - Grafos de Precedencia.
    - Instrucciones específicas: FORK y JOIN, COBEGIN COEND;
  - Asignar dichas tareas a procesos concurrentes en función del paso anterior.

¿Es importante el número de procesos elegido?

Grafos de Precedencia (II)

- Los grafos de precedencia permiten expresar la concurrencia existente entre un conjunto de sentencias.
  - Cada sentencia es un conjunto de acciones atómicas ejecutadas secuencialmente.
- En concreto, especifican las restricciones a la concurrencia entre dichas sentencias:
  - La ejecución de la sentencia s1 debe preceder a la ejecución de la sentencia s2.
  - El resto es concurrente.
- Se utiliza un grafo para expresar dichas restricciones:
  - Cada sentencia se corresponde con un estado del grafo.
  - Se crean transiciones entre las sentencias para las que existen restricciones de precedencia.

Grafos de Precedencia (III)



¿Cuántos procesos serían necesarios para implementar la concurrencia expresada por cada uno de los grafos?