Sobre la programación concurrente

Lecturas recomendadas:
G. Andrews, introducción parte 1
Ben-Ari, secciones 2.1 y 2.3
A. Burns, A. Wellings, capítulo 2 (hasta 2.4)
Andrews & Schneider, todo el artículo

Manuel Carro

Universidad Politécnica de Madrid

¿Qué es la concurrencia?

“Simultaneidad”

+

Sincronización

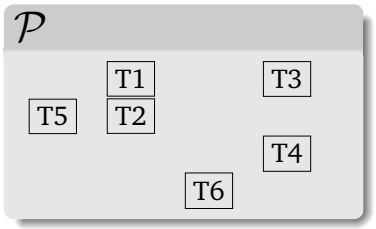
¿Qué es la programación concurrente?

- Procesos / tareas con control independientemente programable
- Es necesaria
  - Comunicación de datos / creación de estructuras de datos
  - Sincronización entre las tareas
- Distinguir *programa* de *tarea*:

  Un *programa*, varias *tareas*

  Metáfora: varios trabajadores construyen una pared.

  - ¿Cuándo y cómo se produce la sincronización en este caso?



- *Tarea* vs. *Proceso*

¿Dónde está la concurrencia?

- Navegador *web* accede una página mientras atiende al usuario
- Varios navegadores acceden a la misma página
- Acceso a disco mientras otras aplicaciones siguen funcionando
- El teléfono avisa de recepción de llamada mientras se habla
- Varios usuarios conectados a un mismo sistema
- Fase eliminatoria de *50 por 15*

La programación concurrente es...

- Difícil:
  - Interacción de tareas, comunicación de datos
  - Intuitivamente, muchos más estados
  - **¿Qué es un estado?**
  - **¿Cuántos estados tienen los siguientes fragmentos de programa?**

```
X := 2;
Y := 3;
X := X + Y;
Y := X - Y;
X := X - Y;
```

```
while True loop
    X := (X + 1) mod 1000;
end loop;
```

- Útil:
  - Permite funcionamiento determinados programas: sistemas **inherentemente concurrentes** con llamadas bloqueantes
  - Optimiza uso CPU: detener un proceso mientras se espera por un dato (`bio.adb`) → evitar *espera activa*
  - Simula simultaneidad (*time sharing*)
  - Simplifica muchos diseños

Tareas e interacción entre las mismas

Tarea: hilo de ejecución (y realización de instrucciones) independiente
(¿Cómo haríamos programas para un ordenador moderno si tuviesen que tener en cuenta qué otros programas están ejecutándose en un momento dado?)

Comunicación: a través de
- datos compartidos en memoria (multiprocesadores)
- datos enviados explícitamente (multicomputadores, sistemas distribuidos, Internet)

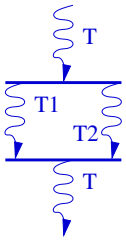
Objetivo: coordinación tareas, coherencia datos, cooperación hacia un fin

Ejemplo de interacción

Código fuente	⟶	Código compilado

X := X + X;		`(a) Load MemPos1, Accum`
		`(b) Add MemPos1, Accum`
		`(c) Sto Accum, MemPos1`

- T_1, T_2 mismo código
- Memoria compartida (`MemPos1`)
- `Accum` privado a cada tarea (*task switching* / registro del procesador)
- Inicialmente X = 1
- Objetivo: X = 4
- Instrucciones indivisibles (atómicas)



Posibles ejecuciones

Código fuente	⟶	Código compilado

X := X + X;		`(a) Load MemPos1, Accum`
		`(b) Add MemPos1, Accum`
		`(c) Sto Accum, MemPos1`

¿Cuáles son los resultados de las siguientes ejecuciones?

1. T_1: a; T_1: b; T_1: c; T_2: a; T_2: b; T_2: c
2. T_1: a; T_2: a; T_1: b; T_2: b; T_1: c; T_2: c
3. T_1: a; T_2: a; T_1: b; T_1: c; T_2: b; T_2: c

- **¿Cuántas ejecuciones diferentes hay? ¿Cuántas son correctas?**
- **No confiar en una planificación determinada**
- Para encontrar errores (en principio): buscar ejecuciones incorrectas
- Intentaremos expresar qué ejecuciones son correctas

Atomicidad y serializabilidad

Básicas para razonar sobre ejecuciones concurrentes

Acción atómica: desarrollada sin interferencia de otras tareas / procesos
(ej., cada una de las instrucciones en ensamblador del caso anterior)

Serializabilidad: ejecución concurrente equivalente a **algún** entrelazado de acciones atómicas
(ej., cada una de las ejecuciones del caso anterior)

- Nos servirán para estudiar las propiedades de los programas concurrentes

Propiedades de un programa concurrente

Sirven para analizar comportamientos

- Caracterización comportamiento: más complicado que en el caso secuencial
- A menudo (pero **no siempre**):
  - Tiempo (absoluto) o complejidad no importante
  - Resultados finales no existentes
  - ¡Terminación no esencial!
- ¿Qué es relevante?
  - Corrección de los resultados parciales
  - Existencia de resultados deseados
- Tres tipos de propiedades:
  - **Seguridad:** propiedades que deseamos que **siempre** se cumplan
  - **Vivacidad:** propiedades que deseamos que se cumplan **en algún momento**
  - **Prioridad**

Propiedades de seguridad

Deben cumplirse en todo momento

A menudo: situaciones que **no** queremos que ocurran:

Nunca debe cumplirse P

$\neg\exists t P(t)$ $(t \equiv tiempo)$

$\longleftrightarrow$

Siempre debe cumplirse $\neg P$

$\forall t \neg P(t)$

X: Natural;

$\mathbf{T}_1$	$\mathbf{T}_2$
loop	**loop**
X := X - 1;	**if** X = 0 **then** X := 1000; **end if**;
end loop;	**end loop;**

- ¿Cómo **garantizar** que X nunca es menor que 0?

Propiedades de seguridad (Contd.)

- Evitan situaciones indeseables
  - $\longrightarrow$ No se entra en ningún estado inconsistente / incorrecto
  - $\longrightarrow$ No se produce ningún dato erróneo (*siempre es verdad que no se produce un dato erróneo* o *todo dato producido es correcto*)
- También llamadas *estáticas*: no dependen de la historia anterior
- Corrección parcial: **siempre que** el proceso termine, su estado final es correcto

Propiedades de vivacidad

Si algo debe ocurrir, garantizamos que ocurrirá en algún momento

$$\exists t P(t)\ (t \equiv tiempo)$$

- A menudo: en **algún momento** se entra en un estado deseable o se produce un dato necesario

T_1	T_2
loop	**loop**
X := X + 1;	**if** X **mod** 2 = 0 **then** Put(X); **end if**;
end loop;	**end loop;**

- ¿Cómo **garantizar** que en algún momento se imprime un número?
- La **historia pasada** cuenta

Propiedades de vivacidad (Contd.)

- Garantizan situaciones deseables
- **¡No confiar en planificación del S.O. o del lenguaje!**
- Propiedades de vivacidad: corrección total (incluye terminación)

Inanición: procesos *conspiran* para que otro no progrese

Propiedades de prioridad

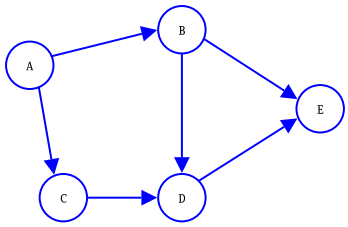
- Relacionadas con vivacidad
- De manera explícita, unas tareas puede tener más *derecho* a ejecutarse que otras
- Diferentes prioridades pueden dar inanición
- Pero a veces esto es lo requerido
- Propiedades **vivacidad/prioridad**

Hacia la corrección

- Asegurar (buenas) propiedades de seguridad/vivacidad/prioridad
- **Nunca** suponer una planificación favorable
- Prueba de corrección difícil: no determinismo
- Depuración convencional inadecuada: intromisión en planificación
- Trabajar con aserciones o imprimir mensajes para ver qué pasa puede ser mejor (pero también afecta planificación)
- ¿Cómo llegar a corrección?
  - Análisis cuidadoso de condiciones de concurrencia
  - Implementación **sistemática** de estas condiciones

Representación de procesos y datos

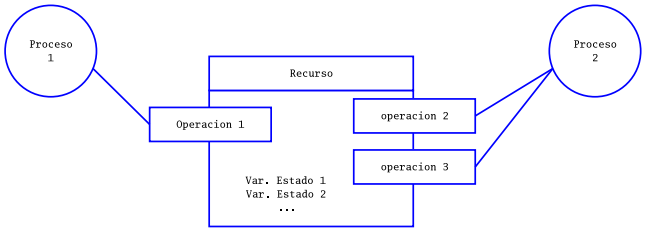
- Tareas sin comunicación: poco interesante
- Interacción entre procesos mediante comunicación datos
- Diagrama de procesos y flujo de datos: esquematiza interacción mediante comunicación datos
- Datos *fluyen* entre procesos



- Natural — pero no la única representación

Procesos y recursos

- Esquematiza interacción tareas concurrentes
- Procesos: acceden a recursos compartidos
- Similar (¡pero sólo similar!) a un TAD
- Recursos:
  - estado,
  - operaciones sobre estado,
  - condiciones sincronización



Transformación entre esquemas