Gestores de sincronización

Lecturas:
Transparencias y apuntes de la asignatura

Manuel Carro

Universidad Politécnica de Madrid

Gestores de sincronización

- Un caso común:
  - Exclusión mutua demasiado fuerte
  - Necesitamos acceso de varios procesos

  → Separar datos fuera recurso

  → Usar recurso sólo para sincronizar

$$Tipo\_Recurso = (Datos : \ldots \times \ Gestor : Tipo\_Gestor)$$

- Usar *Gestor* sólo para regular acceso

Gestores (Cont.)

- Ejemplo de uso:

```
Operacion_X(Gestor);
```

se traduce en

```
Iniciar_Operacion_X(Gestor);
...
Terminar_Operacion_X(Gestor);
```

- Iniciar_Operacion_X puede permitir el paso a varios procesos: uso de datos concurrente
- Suele corresponder a una operación de más alto nivel (y debe ser encapsulada como tal)
- Operación *protegida* debe ser reentrante

Ejemplo: lectores / escritores

- Procesos quieren leer y escribir datos
- Lectores acceden concurrentemente datos
- Escritores acceden en exclusión mutua:

OPERACIONES

ACCIÓN Leer: *TipoBaseDeDatos* . . .
ACCIÓN Escribir: *TipoBaseDeDatos* . . .

SEMÁNTICA
DOMINIO:
TIPO: *TipoBaseDeDatos = datos:* . . .

PROTOCOLOS: Leer
Escribir

CONCURRENCIA: Leer*

- Concurrencia: **A** | **A** ‖ **B** | **A***

Ejemplo: lectores / escritores (Cont.)

- Datos siendo leídos/escritos no relevantes: sólo importa gestor
- C-TADSOL / recurso no utilizable directamente
- Desdoblar:
  - Inicio / terminación lectura
  - Inicio / terminación escritura
- Cada una de ellas atómica

Lectores y escritores: seguridad

ACCIÓN IniciarLectura: *TipoGestor[es]*
ACCIÓN TerminarLectura: *TipoGestor[es]*
ACCIÓN IniciarEscritura: *TipoGestor[es]*
ACCIÓN TerminarEscritura: *TipoGestor[es]*

PROTOCOLOS: Leer: IniciarLectura ; TerminarLectura
Escribir: IniciarEscritura ; TerminarEscritura

CONCURRENCIA: Leer*
TIPO: *TipoGestor = (nLect:* $\mathbb{N}\times$ *Escrib:* $\mathbb{B}$*)*

INVARIANTE: $\forall$ g $\in$ TipoGestor • g.Escrib $\rightarrow$ g.nLect = 0

```
Gestor.IniciarLectura;
« L e e r »
Gestor.TerminarLectura;
```

```
Gestor.IniciarEscritura;
« E s c r i b i r »
Gestor.TerminarEscritura;
```

Lectores y escritores: seguridad (Cont.)

CPRE: ¬ g.Escrib
IniciarLectura (g)
POST: $g^{sal} = g^{ent} \setminus g^{sal}.nLect = g^{ent}.nLect + 1$

CPRE: cierto
TerminarLectura (g)
POST: $g^{sal} = g^{ent} \setminus g^{sal}.nLect = g^{ent}.nLect - 1$

CPRE: $g.nLect = 0 \wedge \neg g.Escrib$
IniciarEscritura (g)
POST: $g^{sal} = g^{ent} \setminus g^{sal}.Escrib$

CPRE: cierto
TerminarEscritura (g)
POST: $g^{sal} = g^{ent} \setminus \neg g^{sal}.nEscrib$

Turno en lectores y escritores

- Puntos a tratar:
  - Evitar llegada continua lectores
  - Decidir si entra lector o escritor al acabar un escritor
- Idea: limitar número lectores que entran sin interrupción
- Flexibilidad: no fijar este número
- Turno: resuelve empate entre lectores y escritores
- Cambio de turno:
  - El primer lector que termina
  - El (primer) escritor que termina

Funcionamiento

- **E:** escritor
- **Li:** lectores

Esperando	**Ejecutando**
L2, L3 L3 E	L1E L2L2 *turnoLectura*¬*turnoLect*

1. L2 y L3 pueden entrar; E **no**
2. L2 entra
3. L1 termina, cambia el turno, L3 no puede entrar
4. Ahora sólo puede entrar E
5. E entra
6. Al salir cambia el turno; L3 puede entrar

Lectores / escritores: vivacidad

- ¿Cuándo intenta entrar un escritor / lector?
- Lectores, escritores bloqueados: atributo *Count*
- IniciarLectura 'Count, IniciarEscritura 'Count
- Otros métodos en otros lenguajes / paradigmas de concurrencia (ej., monitores)
- Supondemos *nLectEsp*, *nEscEsp*
- Puede simularse:

  PROTOCOLOS: Leer: PermisoLect; IniciarLect; TerminarLect
  CONCURRENCIA: Leer*
- Veremos cómo quedan las precondiciones
- *nLectEsp* y *nEscEsp* pueden implementarse de varios modos (dependiendo, p.ej., del lenguaje)

Lectores / escritores: tipo

- Aumentamos tipo anterior
- Nuevo: número de procesos esperando, turno

TIPO: $TipoGestor = (nLect : \mathbb{N} \times Escrib : \mathbb{B} \times turnoLect : \mathbb{B} \times nLectEsp : \mathbb{N} \times nEscEsp : \mathbb{N})$
INVARIANTE: $\forall g \in TipoGestor.(g.Escrib \rightarrow g.nLect = 0)$

- Aumentamos también la sincronización
- Pero sin violar la seguridad:

$$\textbf{CPRE}_{\text{vivacidad}} \rightarrow \textbf{CPRE}_{\text{seguridad}}$$

- `lectsesc.adb` vs. `lectsesc_seg.adb`

Lectores / escritores: vivacidad

CPRE: $\neg g.nEscrib \wedge \underline{(g.turnoLect \vee g.nEscEsp = 0)}$
IniciarLectura (g)
POST: $g^{sal} = g^{ent} \setminus g^{sal}.nLect = g^{ent}.nLect + 1$

CPRE: *cierto*
TerminarLectura (g)
POST: $g^{sal} = g^{ent} \setminus g^{sal}.nLect = g^{ent}.nLect - 1 \wedge \underline{\neg g^{sal}.turnoLect}$

CPRE: $\neg g.nEscrib \wedge g.nLect = 0 \wedge \underline{(\neg g.turnoLect \vee g.nLectEsp = 0)}$
IniciarEscritura (g)
POST: $g^{sal} = g^{ent} \setminus g^{sal}.nEscrib$

CPRE: *cierto*
TerminarEscritura (g)
POST: $g^{sal} = g^{ent} \setminus \neg g^{sal}.nEscrib \wedge \underline{g^{sal}.turnoLect}$

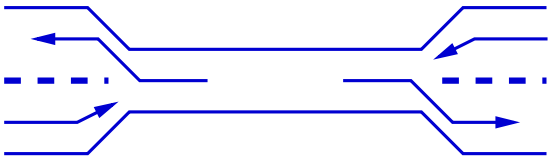
Uso de turnos

- Común: L/E sólo un ejemplo
- En esta caso sólo dos tipos de procesos
- Puede haber varios:
  - Diferenciados por la operación llamada
  - Diferenciados por los parámetros
- En general: pueden expresarse propiedades que lleven a una planificación explícita
- Pero necesitan conocimiento *especial*:
  - Número de procesos bloqueados,
  - Número de procesos en el sistema.
  - ...

  (¡como cualquier planificador!)

Ejemplo: puente estrecho

- Coches intentan cruzar un puente



- No hay espacio para ambas direcciones
- Similar a lectores/escritores
- Modelizar y programar (con vivacidad)
- Dificultad adicional:
  - Cada vehículo un peso diferente
  - Puente aguanta peso máximo (sumar el de los vehículos)