Secciones críticas y exclusión mutua

Lecturas:
Andrews, secciones 3.1, 3.2, 3.3, 3.4
Ben-Ari, sección 2.2

Manuel Carro

Universidad Politécnica de Madrid

El primer problema a resolver

Código fuente $\longrightarrow$ Código compilado

X := X + X;

```
(a) Load MemPos1, Accum
(b) Add MemPos1, Accum
(c) Sto Accum, MemPos1
```

- Regular acceso a X := X + X
- Sólo una tarea a la vez: *exclusión mutua*
- *Sección crítica* del programa (reduce número ejecuciones posibles)

Esquema

Protocolo de entrada
Sección crítica
Protocolo de salida

Sección crítica

- Soluciones *software* y *hardware*
  - *Hardware*: instrucciones especializadas (*Test and Set* y *Swap*)
  - *Software*: algoritmos exclusión mutua / soporte S.O.
- Muchos algoritmos de exclusión mutua
  - Peterson (lo veremos)
  - Dekker (Ben-Ari, Cáp. 3)
  - Panadería, Tickets (Andrews, Cáp. 3)
  - etc.

Exclusión mutua

- Ejemplo de necesidad de exclusión mutua:
  `shvar_without_mutex.adb`
  - Variables *vistas* por varios procesos
  - Accesos indiscriminados $\longrightarrow$ resultados erróneos
- Intentaremos implementación protocolos entrada / salida
- Supondremos atomicidad sentencias Ada
- Suposición: procesos no mueren en
  - Sección crítica
  - Protocolos entrada / salida

Protocolo de entrada
Sección crítica
Protocolo de salida

Intentos de algoritmo de exclusion mutua

(`andes{1,2,3,4,5}.adb`)

- Dos tareas intentan acceder al mismo recurso compartido
- Diferentes velocidades / recorridos
- Sincronización inicio / final acceso a recurso

Primer intento de exclusión mutua

```
type Tipo_Turno is (Dcha, Izq);
T: Tipo_Turno:= Dcha;
```

```
loop
   while (T /= Izq) loop
      null;
   end loop;
   ....
   T := Dcha;
end loop;
```

```
loop
   while (T /= Dcha) loop
      null;
   end loop;
   ....
   T := Izq;
end loop;
```

Segundo intento de exclusión mutua

```
Entra_Izq : Boolean := False;
loop
   while Entra_Dcho loop
      null;
   end loop;
   Entra_Izq := True;
   ....
   Entra_Izq := False;
end loop;
```

```
Entra_Dcho: Boolean := False;
loop
   while Entra_Izq loop
      null;
   end loop;
   Entra_Dcho := True;
   ....
   Entra_Dcho := False;
end loop;
```

- Entra_i = True **sii** proceso i en S.C.

Tercer intento de exclusión mutua

```
Entra_Izq: Boolean := False;
loop
   Entra_Izq := True
   while Entra_Dcho loop
      null;
   end loop;
   ....
   Entra_Izq := False;
end loop;
```

```
Entra_Dcho: Boolean := False;
loop
   Entra_Dcho := True
   while Entra_Izq loop
      null;
   end loop;
   ....
   Entra_Dcho := False;
end loop;
```


Cuarto intento de exclusión mutua

```
Entra_Izq: Boolean := False;
loop
   Entra_Izq := True;
   while Entra_Dcho loop
      Entra_Izq := False;
      Entra_Izq := True;
   end loop;
   ....
   Entra_Izq := False;
end loop;
```

```
Entra_Dcho: Boolean := False;
loop
   Entra_Dcho := True;
   while Entra_Izq loop
      Entra_Dcho := False;
      Entra_Dcho := True;
   end loop;
   ....
   Entra_Dcho := False;
end loop;
```

Quinto intento de exclusión mutua

```
Turno : Tipo_Turno := Dcha;
```

```
Entra_Izq: Boolean := False;

loop
   Entra_Izq:= True;
   Turno:= Der;
   while Entra_Der and
         (Turno = Der) loop
      null;
   end loop;
   -- Sección crítica
   Entra_Izq := False;
end loop;
```

```
Entra_Dcho: Boolean := False;

loop
   Entra_Dcho:= True;
   Turno:= Izq;
   while Entra_Izq and
         (Turno = Izq) loop
      null;
   end loop;
   -- Sección crítica
   Entra_Dcho := False;
end loop;
```


Prueba

Observaciones:

- Proceso i en s.c. $\longrightarrow$ **Entra**$_i$ = ***True***
- **Turno** resuelve conflictos
- Algún proceso es el último en cambiar **Turno**

```
Entra_Dcho: Boolean := False;

loop
   Entra_Dcho:= True;
   Turno:= Izq;
   while Entra_Izq and
         (Turno = Izq) loop
      null;
   end loop;
   -- Sección crítica
   Entra_Dcho := False;
end loop;
```

Exclusión mutua:

- Supongamos ambos procesos en sección crítica
- Entonces Entra_Izquierdo = Entra_Derecho = True
- La condición de entrada es sólo cierta para uno
- Éste debe haber entrado primero
- Y al estar aún en s.c. no puede haberle dado el turno al otro
- **Contradicción**

Prueba (Cont.)

Cadencia correcta:

- Proceso$_i$ "está de paseo"
- Entonces Entra$_i$ = False
- Por tanto el otro proceso puede entrar

```
Entra_Dcho: Boolean := False;

loop
   Entra_Dcho:= True;
   Turno:= Izq;
   while Entra_Izq and
         (Turno = Izq) loop
      null;
   end loop;
   -- Sección crítica
   Entra_Dcho := False;
end loop;
```

Ausencia de interbloqueo:

- Proceso_Derecho y Proceso_Izquierdo bloqueados en bucle entrada
- Imposible: o bien Turno = Derecha o bien Turno = Izquierda
- Uno de los dos puede continuar

Prueba (Cont.)

Ausencia de inanición:

- Proceso$_i$ entra s.c. repetidamente:
  - Al salir da oportunidad al otro Procesa (con Entra$_i$)
- Proceso$_i$ puede intentar reentrar antes que el otro:
  - Pero al intentar reentrar se quita Turno a sí mismo
  - Y el otro proceso ha solicitado paso (con Entra$_j$)

```
Entra_Dcho: Boolean := False;

loop
  Entra_Dcho:= True;
  Turno:= Izq;
  while Entra_Izq and
       (Turno = Izq) loop
    null;
  end loop;
  -- Sección crítica
  Entra_Dcho := False;
end loop;
```


Propiedades vistas

Seguridad:

- Ausencia de interbloqueo (**normalmente necesaria**)
- Exclusión mutua

Vivacidad:

- Cadencia correcta
  (acceso recurso si está libre y proceso intenta)
- Ausencia de inanición
  (asegurar acceso recurso a todos los procesos)

Espera activa

- Algoritmo Peterson (y otros): *espera activa*
- Tareas esperando entrar realizan trabajo
- Válidos como idea inicial
  - Sólo algunas veces usados en casos muy particulares (multiprocesadores, R.C. muy pequeñas, sistemas con *hardware* dedicado)
- No útiles en la mayor parte de los casos
- Mecanismos de más bajo nivel (S.O. suspende tareas)
- **¡Huid de la espera activa!**

Ejercicio

- Ben-Ari, *Principles of Concurrent Programming*, ejercicio 3.4
- Dos procesos *P1* y *P2*
- Demostrar corrección o encontrar contraejemplo

```
c1, c2: Integer := 1;
loop -- Proceso P1
  loop
    c1 := 1 - c2;
  exit when c2 /= 0;
  end loop;
  .... -- Sección crítica
  c1 := 1;
end loop;
loop -- Proceso P2
  loop
    c2 := 1 - c1;
  exit when c1 /= 0;
  end loop;
  .... -- Sección crítica
  c2 := 1;
end loop;
```