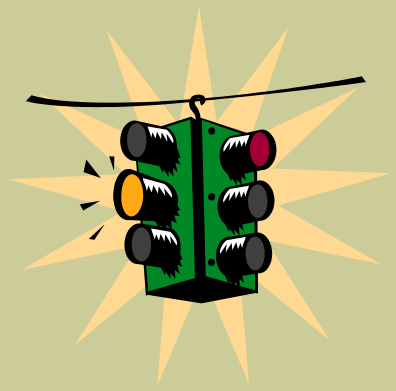
Programación Concurrente ATIC
Redictado de Programación Concurrente

Clase 3

Facultad de Informática

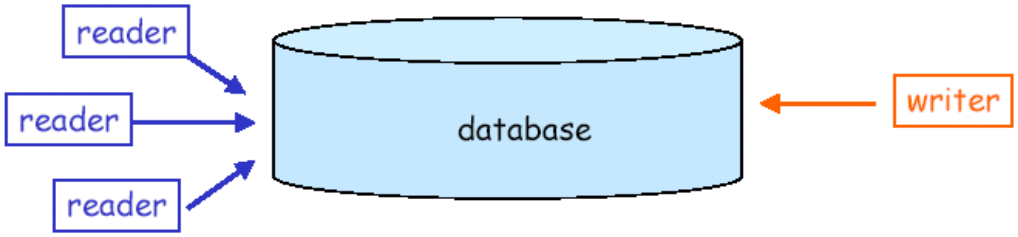
UNLP

Semáforos (continuación)



Problemas básicos y técnicas
Lectores y escritores

- ***Problema:*** dos clases de procesos (*lectores* y *escritores*) comparten una Base de Datos. El acceso de los *escritores* debe ser exclusivo para evitar interferencia entre transacciones. Los *lectores* pueden ejecutar concurrentemente entre ellos si no hay escritores actualizando.



- Procesos asimétricos y, según el scheduler, con diferente prioridad.
- Es también un problema de ***exclusión mutua selectiva***: clases de procesos compiten por el acceso a la BD.
- Diferentes soluciones:
  - Como problema de exclusión mutua.
  - Como problema de sincronización por condición.

Problemas básicos y técnicas

Lectores y escritores: *como problema de exclusión mutua*

- Los escritores necesitan acceso mutuamente exclusivo.
- Los lectores (como grupo) necesitan acceso exclusivo con respecto a cualquier escritor.

```
sem rw = 1;

process Lector [i = 1 to M]
{  while(true)
       { …
         P(rw);
         lee la BD;
         V(rw);
       }
}

process Escritor [j = 1 to N]
{  while(true)
       { …
         P(rw);
         escribe la BD;
         V(rw);
        }
}
```

No hay concurrencia entre lectores

- Los lectores (como grupo) necesitan bloquear a los escritores, pero sólo el primero necesita tomar el *lock* ejecutando *P(rw)*.
- Análogamente, sólo el último lector debe hacer *V(rw)*.

Problemas básicos y técnicas

Lectores y escritores: *como problema de exclusión mutua*

```
int nr = 0;          # número de lectores activos
sem rw = 1;          # bloquea el acceso a la BD
```

```
process Escritor [j = 1 to N]
{  while(true)
        { …
          P(rw);
          escribe la BD;
          V(rw);
        }
}
```

```
process Lector [i = 1 to M]
{ while(true)
      { …
        <  nr = nr + 1;  if (nr == 1) P(rw); >
        lee la BD;
        <  nr = nr - 1; if (nr == 0) V(rw);  >
      }
}
```

Problemas básicos y técnicas

Lectores y escritores: *como problema de exclusión mutua*

```
int nr = 0;          # número de lectores activos
sem rw = 1;          # bloquea el acceso a la BD
sem mutexR= 1;       # bloquea el acceso de los lectores a nr
```

```
process Escritor [j = 1 to N]
{  while(true)
        { …
          P(rw);
          escribe la BD;
          V(rw);
        }
}
```

```
process Lector [i = 1 to M]
{ while(true)
      { …
        P(mutexR);
        nr = nr + 1;
        if (nr == 1) P(rw);
        V(mutexR);
        lee la BD;
        P(mutexR);
        nr = nr - 1;
        if (nr == 0) V(rw);
        V(mutexR);
      }
}
```

Problemas básicos y técnicas

Lectores y escritores: *sincronización por condición*

- Solución anterior ⇒ preferencia a los lectores ⇒ no es ***fair***.
- Otro enfoque ⇒ introduce la técnica ***passing the baton***: emplea SBS para brindar exclusión y despertar procesos demorados.
- Puede usarse para implementar ***await*** arbitrarios, controlando de forma precisa el orden en que los procesos son despertados
- En este caso, pueden contarse (por medio de *nr* y *nw*) los procesos de cada clase intentando acceder a la BD, y luego restringir el valor de los contadores. ¿Cuáles son los estados buenos y malos de ***nr*** y ***nw***?

```
int nr = 0, nw = 0;

process Lector [i = 1 to M]
{ while(true)
    { …
      ⟨ await (nw == 0) nr = nr + 1; ⟩
      lee la BD;
      ⟨ nr = nr − 1; ⟩
    }
}
```

```
process Escritor [j = 1 to N]
{ while(true)
    { …
      ⟨ await (nr==0 and nw==0) nw=nw+1; ⟩
      escribe la BD;
      ⟨ nw = nw – 1; ⟩
    }
}
```

Problemas básicos y técnicas

Técnica Passing de Baton

- En algunos casos, await puede ser implementada directamente usando semáforos u otras operaciones primitivas. *Pero no siempre…*
- En el caso de las guardas de los await en la solución anterior, se superponen en que el protocolo de entrada para escritores necesita que tanto **nw** como **nr** sean 0, mientras para lectores sólo que **nw** sea 0.
- Ningún semáforo podría discriminar entre estas condiciones → ***Passing the baton.***

Passing the baton: técnica general para implementar sentencias ***await.***

Cuando un proceso está dentro de una SC mantiene el *baton* (*testimonio*, *token*) que significa permiso para ejecutar.

Cuando el proceso llega a un ***SIGNAL*** (sale de la SC), pasa el *baton* (control) a otro proceso. Si ningún proceso está esperando por el *baton* (es decir esperando entrar a la SC) el *baton* se libera para que lo tome el próximo proceso que trata de entrar.

Problemas básicos y técnicas

Técnica Passing de Baton

La sincronización se expresa con sentencias atómicas de la forma:

$$\mathbf{F_1 : \langle\ S_i\ \rangle} \quad \mathbf{o} \quad \mathbf{F_2 : \langle\ await\ (B_j)\ S_j\ \rangle}$$

Puede hacerse con semáforos binarios divididos (SBS).

e semáforo binario inicialmente ***1*** (controla la entrada a sentencias atómicas).

Utilizamos un semáforo b_j y un contador d_j cada uno con guarda diferente B_j; todos inicialmente *0*.

b_j se usa para demorar procesos esperando que B_j sea *true*.
d_j es un contador del número de procesos demorados sobre b_j.

e y los b_j se usan para formar un SBS: a lo sumo uno a la vez es ***1***, y cada camino de ejecución empieza con un ***P*** y termina con un único ***V***.

Problemas básicos y técnicas

Técnica Passing de Baton

$\langle S_i \rangle$

F $_1$ **:** **P(e);**
S_i;
SIGNAL;

$\langle$ **await (B_j) S_j** $\rangle$

F $_2$ **:** **P(e);**
if (not B_j) {$d_j = d_j + 1$; V(e); P(b_j); }
S_j;
SIGNAL

SIGNAL: **if (B_1 and $d_1 > 0$) {$d_1 = d_1 - 1$; V(b_1)}**
□ ...
□ (B_n and $d_n > 0$) {$d_n = d_n - 1$; V(b_n)}
□ else V(e);
fi

Problemas básicos y técnicas

Lectores y escritores: *Técnica Passing de Baton*

```
int nr = 0, nw = 0;

process Lector [i = 1 to M]
{ while(true)
    { …
      ⟨ await (nw == 0) nr = nr + 1; ⟩
      lee la BD;
      ⟨ nr = nr − 1; ⟩
    }
}


process Escritor [j = 1 to N]
{ while(true)
    { …
      ⟨ await (nr==0 and nw==0) nw=nw+1; ⟩
      escribe la BD;
      ⟨ nw = nw – 1; ⟩
     }
}
```

```
int nr = 0, nw = 0, dr = 0, dw = 0;
sem e = 1, r = 0, w = 0;

process Lector [i = 1 to M]
{ while(true) {
        P(e);
        if (nw > 0){dr = dr+1; V(e); P(r); }
        nr = nr + 1;
        SIGNAL1 ;
        lee la BD;
        P(e); nr = nr – 1; SIGNAL2 ;
    }
}

process Escritor [j = 1 to N]
{ while(true) {
        P(e);
        if (nr > 0 or nw > 0) {dw = dw+1; V(e); P(w); }
        nw = nw + 1;
        SIGNAL3 ;
        escribe la BD;
        P(e); nw = nw – 1; SIGNAL4 ;
    }
}
```

Problemas básicos y técnicas

Lectores y escritores: *Técnica Passing de Baton*

```
int nr = 0, nw = 0, dr = 0, dw = 0;
sem e = 1, r = 0, w = 0;

process Lector [i = 1 to M]
{ while(true)
    {  P(e);
       if (nw > 0){dr = dr+1; V(e); P(r); }
       nr = nr + 1;
       SIGNAL1 ;
       lee la BD;
       P(e); nr = nr – 1; SIGNAL2 ;
    }
}

process Escritor [j = 1 to N]
{ while(true)
    {  P(e);
       if (nr > 0 or nw > 0) {dw = dw+1; V(e); P(w); }
       nw = nw + 1;
       SIGNAL3 ;
       escribe la BD;
       P(e); nw = nw – 1; SIGNAL4 ;
    }
}
```

El rol de los **SIGNAL$_i$** es el de señalizar ***exactamente*** a uno de los semáforos $\Rightarrow$ los procesos se van pasando el ***baton***.

SIGNAL$_i$ es una abreviación de:

```
if (nw == 0 and dr > 0)
      {dr = dr - 1; V(r);}
elsif (nr == 0 and nw == 0 and dw > 0)
      {dw = dw - 1; V(w);}
else  V(e);
```

Algunos de los SIGNAL se pueden simplificar.

Problemas básicos y técnicas

Lectores y escritores: *Técnica Passing de Baton*

```
int nr = 0, nw = 0, dr = 0, dw = 0;          sem e = 1, r = 0, w = 0:
```

```
process Lector [i = 1 to M]
{  while(true)
      {  P(e);
        if (nw > 0) {dr = dr+1; V(e); P(r); }
        nr = nr + 1;
        if (dr > 0) {dr = dr – 1; V(r); }
        else V(e);
        lee la BD;
        P(e);
        nr = nr – 1;
        if (nr == 0 and dw > 0)
              {dw = dw – 1; V(w); }
        else V(e);
       }
}
```

```
process Escritor [j = 1 to N]
{ while(true)
   {  P(e);
      if (nr > 0 or  nw > 0)
              {dw=dw+1; V(e); P(w);}
      nw = nw + 1;
      V(e);
      escribe la BD;
      P(e);
      nw = nw – 1;
      if (dr > 0) {dr = dr - 1; V(r); }
      elseif (dw > 0) {dw = dw - 1; V(w); }
      else V(e);
   }
}
```

Da preferencia a los lectores ⇒ ¿Cómo puede modificarse?

Alocación de Recursos y Scheduling

Problema: decidir cuándo se le puede dar a un proceso determinado acceso a un recurso.

Recurso: cualquier objeto, elemento, componente, dato, SC, por la que un proceso puede ser demorado esperando adquirirlo.

Definición del problema: procesos que compiten por el uso de unidades de un recurso compartido (cada unidad está ***libre*** o ***en uso***).

request (parámetros): ⟨await (request puede ser satisfecho) tomar unidades;⟩

release (parámetros): ⟨retornar unidades;⟩

- Puede usarse Passing the Baton:

request (parámetros): P(e);
if (request no puede ser satisfecho) DELAY;
tomar las unidades;
SIGNAL;

release (parámetros): P(e);
retornar unidades;
SIGNAL;

Alocación de Recursos y Scheduling

Alocación Shortest-Job-Next (SJN)

- Varios procesos que compiten por el uso de un recurso compartido *de* ***una sola unidad***.
- **request** (tiempo,id). Si el recurso está libre, es alocado inmediatamente al proceso ***id***; sino, el proceso ***id*** se demora.
- **release** ( ). Cuando el recurso es liberado, es alocado al proceso demorado (si lo hay) con el mínimo valor de ***tiempo***. Si dos o más procesos tienen el mismo valor de ***tiempo***, el recurso es alocado al que esperó más.
- SJN minimiza el tiempo promedio de ejecución, aunque *es* ***unfair*** (¿por qué?). Puede mejorarse con la técnica de ***aging*** (dando preferencia a un proceso que esperó mucho tiempo).
- Para el caso general de alocación de recursos (NO SJN):

```
bool libre = true;
request (tiempo,id):  ⟨await (libre) libre = false;⟩
release ():  ⟨libre = true;⟩
```

Alocación de Recursos y Scheduling

Alocación Shortest-Job-Next (SJN)

- En SJN, un proceso que invoca a *request* debe demorarse hasta que el recurso esté libre y su pedido sea el próximo en ser atendido de acuerdo a la política. El parámetro tiempo entra en juego sólo si un pedido debe ser demorado.

```
request (tiempo, id):
            P(e);
            if (not libre) DELAY;
            libre = false;
            SIGNAL;

release ( ):
            P(e);
            libre = true;
            SIGNAL;
```

- En ***DELAY*** un proceso:
  - Inserta sus parámetros en un conjunto, cola o lista de espera (*pares*).
  - Libera la SC ejecutando V(e).
  - Se demora en un semáforo hasta que *request* puede ser satisfecho.
- En ***SIGNAL*** un proceso:
  - Cuando el recurso es liberado, si pares no está vacío, el recurso es asignado a un proceso de acuerdo a SJN.

- Cada proceso tiene una condición de demora distinta, dependiendo de su posición en ***pares***. El proceso ***id*** se demora sobre el semáforo ***b[id]***.

Alocación de Recursos y Scheduling

Alocación Shortest-Job-Next (SJN)

```
bool libre = true;  Pares = set of (int, int) =  ∅;  sem e = 1, b[n] = ([n] 0);

request(tiempo,id):   P(e);
                      if (! libre){ insertar (tiempo, id) en Pares; V(e); P(b[id]); }
                      libre = false;
                      V(e);

release( ):           P(e);
                      libre = true;
                      if (Pares ≠ ∅ ) { remover el primer par (tiempo,id) de Pares; V(b[id]); }
                      else V(e);
```

s es un ***semáforo privado*** si exactamente un proceso ejecuta operaciones ***P*** sobre ***s***. Resultan útiles para señalar procesos individuales. Los semáforos ***b[id]*** son de este tipo.

Alocación de Recursos y Scheduling

Alocación Shortest-Job-Next (SJN)

```
bool libre = true;  Pares = set of (int, int) = ∅;  sem e = 1, b[n] = ([n] 0);

Process Cliente [id: 1..n]
{  int sig;
   //Trabaja
   tiempo = //determina el tiempo de uso del recurso//
   P(e);
   if (! libre) { insertar (tiempo, id) en Pares;
                  V(e);
                  P(b[id]);
                }
   libre = false;
   V(e);
   //USA EL RECURSO
   P(e);
   libre = true;
   if (Pares ≠ ∅ ) { remover el primer par (tiempo, sig) de Pares;
                     V(b[sig]);
                   }
   else V(e);
}
```

¿Que modificaciones deberían realizarse para respetar el orden de llegada?

¿Que modificaciones deberían realizarse para generalizar la solución a recursos de múltiple unidad?