Programació de serveis i processos

TEMA 01 – PART IV

Programació de processos i fils

1. Programació multiprocés i paral·lela
   1.4. Sincronització i comunicació entre processos
      1.4.3. Solucions a problemes de sincronització i comunicació
         - Semàfors

Unió Europea
Fons social europeu
L'FSE inverteix en el teu futur

IES DR. LLUÍS SIMARRO

1. Programació multiprocés i paral·lela

1.4 Sincronització i comunicació entre processos

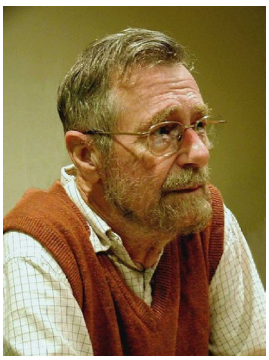
1.4.3. Solucions a problemes de sincronització i comunicació

Al llarg de la història s'han proposat solucions específiques pels problemes anteriors que val la pena tindre en compte, malgrat que **resulta difícil generalitzar**-ne una solució ja que depenen de la **complexitat**, la quantitat de **seccions crítiques**, del nombre de **processos** que requerisquen **exclusió mútua** i de la **interdependència** existent entre processos. Ací veurem la sincronització per mitjà de **semàfors**, de **monitors** i de **pas de missatges**.

Semàfors

Imaginem una carretera d'un sol carril que ha de passar per un túnel. Hi ha un semàfor a cada extrem del túnel que ens indica quan podem passar i quan no. Si el semàfor està en verd el cotxe passarà de forma immediata i el semàfor passarà a roig fins que isca. Aquest símil ens introdueix a la definició real d'un semàfor.

Els semàfors són una tècnica de **sincronització** de **memòria compartida** que **impedeix l'entrada** del **procés** a la **secció crítica bloquejant-lo**. El concepte va ser introduït per l'informàtic holandès Dijkstra per resoldre el problema l'**exclusió mútua** i permetre resoldre gran part dels problemes de **sincronització entre processos**.

	**Edsger Wybe Dijkstra**, informàtic holandès. La seua idea d'exclusió mútua ideada durant la dècada dels 60 és utilitzada per molts processadors i programadors moderns

Els **semàfors**, no només **controlen** l'**accés** a la **secció crítica** sinó que a més disposen d'informació complementària per poder **decidir** si **cal o no bloquejar l'accés** d'aquells **processos** que ho **sol·liciten**. Així per exemple, serviria per solucionar problemes senzills (amb poca interdependència) del tipus **escriptors-lectors** o **productors-consumidors**.

La solució per exemple en el cas dels **escriptors-lectors** passaria per fer que els **lectors abans de consultar la secció crítica demanaren permís** d'**accés al semàfor**, el qual en funció de si es troba **bloquejat** (roig) o **alliberat** (verd) **pararà** l'**execució** del procés sol·licitant o bé el **deixarà continuar**.

Els **escriptors** per altra banda, **abans** d'**entrar a la secció critica**, manipularan el semàfor **posant-lo** en **roig** i no el tornaran a **posar** en **verd** fins que **hagen acabat** d'escriure i **abandonen** la **secció crítica**.

De forma genèrica distingirem 3 tipus d'operacions en un semàfor. El semàfor admet 3 operacions:

- **Inicialitza(semàfor)** (**initial(s)**): es tracta de l'operació que permet **posar** en **marxa** el **semàfor**. La operació pot **rebre** un **valor** per paràmetre que indicarà si aquest **començarà boquejat** (roig) o **alliberat** (verd).

- **Allibera(semàfor)** (**sendSignal(s)**): canvia el valor intern del semàfor **posant-lo en verd** (l'**allibera**). Si existeixen **processos en espera**, els **activa** per tal que **finalitzen** la seua **execució**.

- **Bloqueja(semàfor)** (**sendWait(s)**): serveix per indicar que el **procés actual vol executar** la **secció crítica**. En cas que el **semàfor** es trobe **bloquejat**, es **para l'execució** del **procés**. També permet **indicar** que cal **posar** el **semàfor** en **roig**.

En cas que es necessite **exclusió mútua**, el semàfor disposarà també d'un **sistema d'espera** (per mitjà de cues de processos) que garantisca l'**accés** a la secció crítica d'un **únic procés a la vegada**.

En realitat la implementació d'un semàfor dependrà molt del problema a resoldre, encara que la dinàmica del funcionament siga sempre molt similar. Així per exemple els semàfors que donen suport al problema de tipus **productor-consumidor**, necessiten implementar-se utilitzant **exclusió mútua** tant per **alliberar** com per **bloquejar**. A més, l'execució d'**alliberament incrementarà** en una unitat un **comptador intern**, mentre que l'execució de **bloqueig** a banda de parar el procés quan el semàfor es trobe bloquejat, **disminuirà** en una unitat el **comptador intern**.

Tenint en compte que el processos **productors** executaran sempre l'operació d'**alliberament** (incrementant el comptador intern) i els processos **consumidors** demanaran accés executant l'operació de **bloqueig** (que disminuirà el comptador intern), és fàcil veure que el valor del **comptador** serà sempre igual a la **quantitat** de **dades** que els **productors** han **generat sense** que els **consumidors** els hagen encara **consumit**. Així podem deduir que si en algun moment el valor arriba al valor **zero**, significarà que **no hi han dades a consumir** i per tant farem que el **semàfor** es trobe **sempre bloquejat** en aquest cas, però es **desbloquege** tan prompte com el **comptador incremente** el seu **valor**.

A més de resoldre problemes de tipus productor-consumidor o lector-escriptor, podem utilitzar **semàfors** per gestionar problemes de **sincronització** on un **procés** haja d'**activar** l'**execució d'un altre** o d'**exclusió mútua** assegurant que **només un procés aconseguirà accedir** a la **secció crítica** perquè el semàfor es quedarà bloquejat fins a l'eixida.

Si volem **sincronitzar dos processos** fent que un d'ells (**p1**) **execute** una **acció** sempre **abans que** l'altra (**p2**), usant un **semàfor**, l'inicialitzarem a 0 per assegurar que es troba bloquejat. La codificació del procés p2 assegura que abans de qualsevol crida a l'acció que volem controlar, sol·licitarà accés al semàfor amb un `sendWait`. Per contra, a la codificació del procés p1 caldrà situar sempre una crida a `sendSignal` just després d'executar l'acció a controlar (com en la taula que hi ha a continuació).

D'aquesta forma ens assegurarem que el procés p1 executarà l'acció sempre abans que p2. Com a **exemple**, imaginem **dos processos**. Un ha d'**escriure Hola**, (procés **p1**) i el procés **p2** ha d'**escriure món**. L'ordre correcte d'execució és primer p1 i després p2, per aconseguir escriure Hola món. En cas que el **procés P1 s'execute abans que p2**, cap problema: escriu ***Hola*** i fa un `sendSignal` al semàfor (semàfor=1). Quan el procés p2 s'executa trobarà semàfor=1, per tant no quedarà bloquejat, podrà fer un `sendWait` al semàfor (semàfor=0) i escriurà ***món***.

Procés 1 (p1)	Procés 2 (p2)
initial(0)	
…	
System.out.print("Hola")	sendWait()
sendSignal()	System.out.print("món")

Però què passa si s'**executa primer el procés p2**? Com que trobarà semàfor a 0, es quedarà bloquejat en fer la petició cridant `sendWait` fins que el procés p1 escriga ***Hola*** i faça el `sendSignal`, desbloquejant el semàfor. Aleshores p2, que estava bloquejat, s'activarà, posarà el semàfor de nou a roig (semàfor=0) i escriurà ***món*** a la pantalla.

Un altre **exemple** que pot il·lustrar l'ús de semàfors, seria el d'una **oficina bancària** que **gestiona** el nostre **compte corrent** al qual es pot accedir des de diferents oficines per ingressar diners o treure diners. Aquestes dues operacions modifiquen el nostre saldo. Serien dues funcions com les següents:

```
        public void ingressar(float diners) {
             float aux;
             aux=llegirSaldo();
             aux=aux+diners;
             guardarSaldo(aux);
         }

        public void traure(float diners) {
```

```
            float aux;
            aux=llegirSaldo();
            aux=aux-diners;
            guardarSaldo(aux);
        }
```

El **problema** ve quan de forma **simultània** es vol fer un **ingrés** i es vol **traure** diners. Si per una banda estem traient diners del compte corrent i per una altra banda algú està fent un ingrés, es podria crear una **situació anòmala**. Hi haurà dos processos concurrents, un traurà diners i l'altre n'ingressarà. Si accedeixen al mateix temps a `llegirSaldo()` els dos agafen el mateix valor, imaginem 100€. El procés que vol ingressar diners, ho vol fer amb la quantitat de 300€. I el que vol treure'n, en vol 30€.

Si continuem l'execució dels dos processos, **depenent** de quin siga l'**ordre d'execució** de les **instruccions**, ens podem trobar amb un **saldo diferent**. Si després de llegir el saldo, el procés d'ingrés acaba l'execució i guarda el saldo, guardaria **400€** (100€ + 300€). Però posteriorment acabaria el procés de treure diners i guardaria a saldo el valor de **70€** (100 € - 30 €). **Hem perdut l'ingrés**. Hi ha dos **processos** que s'estan executant a una **secció crítica** que hauríem de protegir en **exclusió mútua**. **Únicament un procés** ha de poder **accedir** a aquesta **secció crítica** i poder **modificar** la variable compartida **saldo**.

Per evitar el problema podem utilitzar un **semàfor**. L'**iniciarem a 1**, indicant el número de **processos que podran entrar a la secció crítica**. I tant en el procés de **traure** diners com en el d'**ingressar**-ne afegirem un `sendWait()` **a l'inici** de les seccions crítiques i un `sendSignal()` al final.

```
    public void ingressar(float diners) {
          sendWait();
```

```
            float aux;
            aux=llegirSaldo();
            aux=aux+diners;
            guardarSaldo(aux);
            sendSignal();
      }
```

```
        public void traure(float diners) {
            sendWait();
            float aux;
            aux=llegirSaldo();
            aux=aux-diners;
            guardarSaldo(aux);
            sendSignal();
      }
```

D'aquesta forma quan un **procés entra** a la **secció crítica** d'un mètode, agafa el semàfor. **Si és 1**, podrà **fer** el `sendWait`, per tant el **semàfor** es posarà **a 0**, **tancat**. I **cap** altre **procés** no **podrà entrar** a cap dels dos **mètodes**. Si un procés intenta entrar, trobarà el **semàfor a 0** i **quedarà bloquejat fins** que el procés que té el **semàfor** faça un `sendSignal`, pose el **semàfor a 1** i **allibere** el **semàfor**.

Utilitzar semàfors és una forma eficient de sincronitzar processos concurrents. **Resol** de forma **simple** l'**exclusió mútua**. Però des del punt de vista de la programació, els **algoritmes són complicats de dissenyar i d'entendre**, ja que les **operacions** de **sincronització** poden estar **disperses** pel codi. Per tant es poden **cometre errors amb facilitat**.