Sincronització de fils: semàfors

Lluís Garrido – lluis.garrido@ub.edu

Octubre 2014

Algorisme 1

Mostra com es pot implementar l'exclusió mútua amb semàfors. En aquest cas observar que el semàfor s només pot prendre els valors 0 o 1. El valor 1 indica que no hi ha cap fil a la secció crítica, mentre que el valor 0 indica que la secció crítica està ocupada per un fil. Aquests tipus de semàfors s'acostumen a anomenar semàfors binaris (ja que prenen només dos valors, 0 o 1).

El semàfor s'inicialitza a 1. El primer fil que entra a *sem_wait* posa el semàfor a 0. Tota la resta de fils que volen entrar a la secció crítica s'hauran d'esperar perquè el semàfor es torni a posar a 1 (generalment, els fils que esperen s'adormiran). Mentrestant, el primer fil executa la secció crítica. En sortir el fil executa un *sem_post*. Així el semàfor es torna a posar a 1 i es despertaran els fils que estan esperant per entrar a la secció crítica. Només un ho aconseguirà.

A la segona versió de l'algorisme es mostra el cas en que el semàfor s'inicialitza a M. En aquest cas un màxim de M fils poden entrar a la secció crítica.

Algorisme 2

Es mostra aquí com múltiples productors i consumidors es poden intercomunicar entre sí mitjançant un *buffer* de comunicació de mida 1. Tenim dos semàfors que són complementaris entre sí: una variable *buit* que indica si el *buffer* és buit (1 és buit, 0 no és buit) i una variable *ocupat* que indica si al *buffer* hi ha dades.

Un productor només podrà passar per la instrucció *sem_wait*(&*buit*) si el buffer és buit. Així que el productor hi ha posat la dada ho comunica al consumidor fent *sem_post*(&*ocupat*). Això fa que el semàfor d'*ocupat* sigui 1, mentre que el de *buit* valdrà 0. Aleshores pot venir un consumidor que agafarà la dada. Ho farà amb *sem_wait*(&*ocupat*). Un cop ha agafat la dada notifica amb *sem_post*(&*buit*) que el buffer torna a estar buit i que un altre productor hi pot dipositar dades.

Aquest algorisme permet tenir múltiples productors i consumidors. En cas que el buffer sigui buit els consumidors es quedaran esperant al *sem_wait(&ocupat)* que algun productor faci sem_*post(&*ocupat*)*. De la mateixa forma, els productors es queden esperant a *sem_wait*(&*buit*) fins que algun consumidor agafa la dada del buffer i fa *sem_post*(&*buit*).

Algorisme 3

L'algorisme anterior és adequat si les dades es produeixen aproximadament en la mateixa velocitat en què es consumeixen. Això generalment no és cert. És per això que utilitzar un *buffer* de capacitat més gran que 1 pot incrementar notablement l'eficiència de la solució.

Suposarem de moment que només tenim un consumidor i un productor. Igual que abans, el productor produeix dades que deixa en un buffer, mentre que el consumidor agafa les dades que hi ha al buffer.

En aquest cas utilitzarem un buffer circular. Tenim dues variables, r i w, que s'utilitzaran com a índexos al buffer circular per saber on es llegeixen i on s'escriuen les dades. En escriure, ho farem a la posició w i incrementem en 1 la posició de w. En llegir, ho fem de la posició r i incrementem en 1 la posició de r. Hem d'assegurar aquestes restriccions sobre r i w: la w mai no pot sobrepassar la r i a l'inrevés, la r no pot sobrepassar mai la w.

A l'algorisme es mostra "buf[w] = data; w = (w + 1) % N" i "data = buf[r]; r = (r + 1) % N". Aquesta és la forma en que productor i consumidor escriuen i llegeixen les dades. Com assegurem les restriccions sobre r i w ? Amb els semàfors. En aquest cas farem servir el que s'anomena un semàfor general, és a dir, un semàfor que pot agafar qualsevol valor positiu (inclòs el zero).

El semàfor *buit* s'inicialitza al nombre d'elements buits que té el buffer. El semàfor *ocupat* s'inicialitza al nombre d'elements ocupats del buffer. Quan el productor vol dipositar un element en el buffer, disminueix en 1 el nombre d'elements buits del buffer amb *sem_ wait*(&*buit*). Aleshores amb *sem_ post*(&*ocupat*) notifica que el nombre d'elements ocupats s'incrementa en 1. Quan el buffer és ple el semàfor *buit* valdrà 0 i el productor no hi podrà dipositar un element: el productor es quedarà adormit per entrar a la secció crítica.

De forma similar, el consumidor crida a la funció *sem_ wait(&ocupat)* abans d'agafar un element del *buffer*. Si hi ha algun element la funció *sem_ wait* retornarà de seguida i haurà disminuït en 1 el valor d'ocupat. El consumidor agafa la dada del *buffer* i no notifica al productor cridant a *sem_ post(&buit)*, que incrementa en 1 el valor de buit.

Observar que el semàfor ofereix una forma senzilla de limitar el nombre màxim de fils que poden accedir a un recurs.

Algorisme 4

Anem a ampliar aquest algorisme per a múltiples productors i consumidors. En aquest cas l'únic que hem de fer és assegurar que les instruccions "buf[w] = data; w = (w + 1) % N" i "data = buf[r]; r = (r + 1) % N" són executades cada cop per un únic fil. Això ho aconseguim mitjançant semàfors que ens proveeixen de l'exclusió mútua.

Algorisme 5

Presentem aquí un exemple de lectors-escriptors. Presentem una proposta basada en exclusió mútua. Aquesta solució no és adequada ja que només hi pot haver un lector llegint dades. Recordem que hi pot haver múltiples lectors a la secció crítica ja que aquests no modifiquen el seu contingut.

Per modificar l'algorisme i aconseguir que hi puguin entrar múltiples lectors, observar que només el primer lector que entra a la secció crítica ha d'agafar el semàfor. Tota la resta no fa falta que ho facin.

Algorisme 6

Es presenta aquí una modificació de l'algorisme anterior perquè múltiples lectors puguin ser a l'interior de la secció crítica. Observeu que hi ha una secció crítica abans i després de llegir les dades (al lector) i abans i després d'escriure les dades (a l'escriptor). Què pot passar ?

1. Suposem que no hi ha cap escriptor escrivint dades. Quan el primer fil lector entri a la secció protegida per *clauLec* incrementarà de 0 a 1 la variable *nr*. A continuació fa un $sem\_wait(\&rw)$, de forma que a partir d'ara un escriptor s'haurà d'esperar per entrar a la seva secció crítica. Imaginem que mentre el lector està llegint dades arriba un segon lector. Aquest simplement incrementarà en 1 la variable *nr*. Podrà accedir també a la base de dades per lectura. A continuació un dels lectors finalitza de llegir. Per això accedeix a la zona d'exclusió mútua de sortida: disminueix en 1 el nombre de lectors. Quan el segon lector surti farà un $sem\_post(\&rw)$. En aquest moment es despertaran els escriptors que estiguin esperant al $sem\_wait(\&rw)$.

2. Suposem que hi ha un escriptor escrivint dades. Ara arriba el primer lector. Farà un $sem\_wait(\&clauLec)$ i a continuació es quedarà esperant a $sem\_wait(\&rw)$. Observar que en quedar-se esperant al $sem\_wait(\&rw)$ no s'allibera automàticament la *clauLec* (els monitors funcionen de forma diferent). Tots els nous lectors que arribin a continuació esperaran a $sem\_wait(\&clauLec)$. Així que l'escriptor faci $sem\_post(\&rw)$ el lector (o escriptor) que estigui esperant al $sem\_wait(\&rw)$ competirà per adquirir el semàfor.

Quin problema té aquesta solució ? Aquesta solució dóna preferència als lectors sobre els escriptors. Hi ha formes de solucionar-ho amb els semàfors, però són complexes i hi ha altres mecanismes que ho permeten fer de forma més senzilla: els monitors.