Sincronització de fils: monitors

Lluís Garrido – lluis.garrido@ub.edu

Octubre 2014

Algorisme 1

Es mostra un exemple de com un semàfor pot ser implementat mitjançant un monitor. Això no significa que un semàfor s'implementi d'aquesta forma en un sistema operatiu.

Suposem que $s == 0$ i que un fil entra a *sem_wait*. Veurà que $s == 0$ i per tant es quedarà bloquejat a *cond*. El fil allibera, en bloquejar-se, la clau sobre la variable *mutex* perquè un altre fil pugui adquirir-lo. Observar que la clau que s'allibera és el segon argument de la funció *wait*. Suposem ara que un altre fil executa *sem_post*. El fil farà $s = s + 1$ i desbloqueja amb *signal* el primer fil que hi hagi a *cond*. Així que el fil que executa *sem_post* surt de la secció crítica allibera la clau sobre la variable *mutex*. En aquest moment el fil que hem desbloquejat pot adquirir la clau *mutex* i executar la seva part de codi.

Observar que al *sem_wait* es fa servir un "*while*" en comptes d'un "*if*". Això és degut a que pot passar el següent: imaginem que tenim un fil bloquejat a *cond*. Suposem que a continuació un segon fil crida a *sem_post* de forma que es crida a *signal* i es desbloqueja el fil de *cond*. Recordar que després de cridar a *signal* el fil que ha fet la crida a *signal* manté adquirida la clau sobre *mutex*. Per tant, el primer fil no comença a executar fins que el segon allibera la clau. En aquest moment arriba un tercer fil i crida a *sem_wait*. Atès que el segon fil té la clau sobre la variable *mutex*, el tercer fil també haurà d'esperar-se a que aquest alliberi la clau. A continuació el segon fil allibera la clau: el primer fil (que hem desbloquejat) i el tercer fil (que acaba d'arribar) competiran per adquirir la clau sobre *mutex*. Només un d'ells ho aconseguirà ja que només un fil pot tenir la clau sobre *mutex*. Suposem que el tercer fil aconsegueix entrar: aquest fil disminueix en ú el valor d's i allibera la clau sobre *mutex*. A continuació el primer fil adquireix la clau sobre *mutex*. Abans de disminuir el valor d's comprova si s és positiu. Ja que no és positiu és tornarà a bloquejar. Si haguéssim fet simplement un "*if*" en comptes d'un "*while*" el primer fil no hauria comprovat si s és positiu després de desbloquejar-se i per tant hauríem acabat amb un valor d's negatiu, que és no vàlid.

Algorisme 2

Es mostra un exemple d'implementació d'una barrera. Aquest algorisme funciona amb qualsevol nombre de fils tot i que no serà tan eficient com ho pot ser una barrera activa. La raó és que aquí els fils es bloquegen i desbloquegen amb les funcions *wait* i *signal* respectivament.

L'algorisme disposa d'una variable *comptador* que està inicialitzada a N, el nombre de fils que executen en paral·lel. Cada cop que un fil arriba a la barrera disminueix en ú la variable *comptador*. En cas que el fil no sigui el darrer fil en entrar a la barrera, aquest es bloqueja amb la instrucció *wait*. Quan el darrer fil entra a la barrera, es posa la variable *comptador* a N i fa una crida a *broadcast*. La funció *broadcast* desbloqueja tots els fils que estiguin esperant a la cua. Recordar que el fil que fa la crida a *broadcast* manté la clau adquirida sobre la variable *mutex*. Així que el aquest fil alliberi la clau sobre la variable *mutex*, s'aniran despertant un a un els fils que hi havia a la cua (adquirint i alliberant un a un la clau sobre la variable *mutex*).

Algorisme 3

Es presenta aquí una solució de múltiples consumidors i productors amb un *buffer* de mida M. Es recomana comparar aquesta solució amb la dels semàfors. La variable comptador indica el nombre d'elements ocupats del *buffer*.

Si el *buffer* és ple ($comptador == N$), el productor es queda bloquejat a la instrucció $wait(condP, mutex)$. Quan arriba un consumidor agafa un element del *buffer* i ho notifica als productors amb un $signal(condP)$. Això desbloqueja el primer fil que hi hagi esperant a *condP*. En cas que no hi hagi cal fil esperant a *condP* aquesta instrucció no té cap efecte.

De forma similar, suposem que el *buffer* és buit ($comptador == 0$). En arribar un consumidor aquest és quedarà bloquejat a $wait(condC, mutex)$. Així que un productor dipositi una dada al *buffer* ho notifica amb $signal(condC)$. Això desbloqueja el primer fil que hi hagi a la cua (en cas que n'hi hagi algun).

Observar que les instruccions *wait* del productor i consumidor estan "protegides" mitjançant un *while* que comprova la condició perquè un fil es bloquegi. Quan un fil desbloquejat comença a executar, comprovarà primer de tot la condició del *while*. En cas que la condició del *while* es compleixi, el fil tornarà a bloquejar-se. Recordar el que pot passar: quan el fil que fa el *signal* crida a *unlock* tots els fils desbloquejats de la cua així com els fils que són a l'entrada de *lock* competiran per adquirir la clau sobre la variable *mutex*. Qualsevol d'ells ho pot aconseguir i hem d'assegurar que l'algorisme funcioni correctament independentment de qui agafa primer la clau sobre *mutex*.

Algorisme 4

Es presenta una solució pels lectors i escriptors on els lectors tenen preferència sobre els escriptors. És a dir, perquè un escriptor pugui començar a escriure no

hi pot haver cap lector llegint. Cal fer notar que en aquesta solució no protegim l'accés a les dades com ho fem amb els productors i consumidors, sinó que hem definit un protocol d'entrada i un de sortida per accedir a les dades.

La variable w indica si hi ha algun escriptor escrivint, mentre que la variable nr indica el nombre de lectors que estan llegint. El lector és senzill: en cas que hi hagi un escriptor escrivint el lector es bloquejarà. En cas contrari s'incrementa la variable nr, que correspon al nombre de lectors llegint, i es podrà accedir a les dades. Per la seva banda l'escriptor comprova si hi ha algun lector llegint o un altre escriptor escrivint. Si és així es bloqueja.

Suposem que no hi ha cap escriptor escrivint i que tenim una sèrie de lectors que van arribant. Aquests podran cridar a *read_lock* sense problemes (incrementaran la variable nr). Si ara arriben escriptors aquests s'hauran de bloquejar (ja que $nr > 0$). Així que els lectors acabin de llegir cridaran a *read_unlock*. Quan el darrer lector cridi a *read_unlock*, aquest desbloquejarà tots els escriptors que estiguin a la cua. Només un d'ells aconseguirà la clau sobre la variable *mutex* (un cop el fil que ha fet el *broadcast* alliberi la clau) i començarà a escriure. Suposem que arriben lectors mentre l'escriptor escriu. Aquests s'hauran de bloquejar i tindrem per tant a la variable *cond* una sèrie de lectors així com d'escriptors. Quan l'escriptor crida a *read_unlock* es desbloquejaran tots els fils de *cond*. Aquests competiran per adquirir la clau sobre la variable *mutex*. Qualsevol d'ells ho pot aconseguir.

Algorisme 5

Es presenta aquí una solució de lectors i escriptors que és justa. La idea és la següent: així que arribi un escriptor cap lector que arribi després d'aquest podrà accedir a les dades encara que hi hagi altres lectors accedint-hi en aquell moment. Per això la variable w està associada a que un escriptor ha fet una petició per escriure (a l'algorisme anterior la variable w indicava que l'escriptor està escrivint). És a dir, quan un fil vol escriure posarà la w a *true* per indicar que vol escriure (encara que no hagi aconseguit encara l'accés exclusiu a les dades). Observar que el codi dels lectors no ha canviat. Només ha canviat el codi de la funció *write_lock* així com el significat (semàntic) de la variable w.

Suposem que tenim una sèrie de lectors accedint a les dades. És a dir, $nr\ != 0$. Suposem també que en aquest moment arriba un escriptor. Veurà que $w == false$ (indica que cap altre escriptor ha fet una petició per escriure abans que ell). Aleshores posarà w a *true* (per indicar a la resta de fils que vol escriure) i es bloquejarà ja que veurà que hi ha lectors accedint a les dades. Suposem ara que arriben nous lectors: aquests veuran que hi ha un escriptor que ha fet una petició per escriure i per tant es bloquejaran. Cap nou lector podrà accedir a les dades. Així que el darrer lector que estava accedint a les dades cridi a *read_unlock* es desbloquejaran tots els fils (lectors i escriptors) que estiguin dormint a la cua *cond*. Però atès que $w == true$ cap lector podrà continuar executant. Un sol escriptor podrà adquirir el *mutex* i per tant començarà a escriure. Quan finalitzi d'escriure, farà el corresponent *broadcast*.

Observar que el *while* (w) que hi ha a l'escriptor fa que un o més escriptors es bloquegin en cas que hi hagi algun altre escriptor que hagi demanat per accedir a les dades.