Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative
Istituto sistemi informativi e networking

SUPSI

Lettori e scrittori, Barbiere sonnolento

Amos Brocco, Ricercatore, DTI / ISIN

14 maggio 2012

Lettori e scrittori

- Problema che riproduce una situazione in cui ci sono più processi concorrenti che cercano di leggere e scrivere in una memoria (es. database)
  - Più processi di lettura (lettori) possono tranquillamente accedere in maniera concorrente
  - Se un processo di scrittura (scrittore) sta modificando i dati, nessun altro processo deve poter accedere, nemmeno i lettori (che potrebbero leggere dei dati inconsistenti)

Lettori e scrittori: soluzione favorevole ai lettori

- Più processi di lettura (lettori) possono tranquillamente accedere in maniera concorrente
  - *il primo lettore blocca l'accesso agli scrittori*
  - *l'ultimo lettore sblocca l'accesso agli scrittori*
- Se un processo di scrittura (scrittore) sta modificando i dati, nessun altro processo deve poter accedere, nemmeno i lettori (che potrebbero leggere dei dati inconsistenti)

Lettori e scrittori: soluzione favorevole ai lettori

```
semaphore nlm = 1; /* Protegge numlet */
semaphore db = 1; /* Protegge dati */
int numlet = 0;

thread lettore(void)
{
    while (TRUE) {
        down(&nlm);
        numlet = numlet + 1;
        if (numlet == 1) down(&db);
        up(&nlm);
        leggi();
        down(&nlm);
        numlet = numlet - 1;
        if (numlet == 0) up(&db);
        up(&nlm);
        processa_dati();
    }
}
```

```
thread scrittore(void)
{
    while (TRUE) {
        genera_dati();
        down(&db);
        scrivi();
        up(&db);
    }
}
```

Lettori e scrittori: soluzione favorevole ai lettori

```
semaphore nlm = 1;
semaphore db = 1;
int numlet = 0;

thread lettore(void)
{
    while (TRUE) {
        down(&nlm);
        numlet = numlet + 1;
        if (numlet == 1) down(&db);
        up(&nlm);
        leggi();
        down(&nlm);
        numlet = numlet - 1;
        if (numlet == 0) up(&db);
        up(&nlm);
        processa_dati();
    }
}
```

Il primo lettore blocca i dati in modo da non lasciar entrare uno scrittore

L'ultimo lettore sblocca i dati in modo da lasciar entrare uno scrittore (se era in attesa)

Lettori e scrittori: soluzione favorevole ai lettori

Lo scrittore deve aspettare che tutti i lettori abbiano finito! Poi blocca l'accesso a altri thread (lettori / scrittori)

```
thread scrittore(void)
{
    while (TRUE) {
        genera_dati();
        down(&db);
        scrivi();
        up(&db);
    }
}
```

Favorire i lettori

- Se ci sono sempre lettori, lo scrittore potrebbe non aver mai l'occasione di scrivere:
  - **starvation**!
- Per evitare questa situazione potremmo modificare il programma come segue:
  - quando un lettore arriva e c'è uno scrittore in attesa, il lettore deve aspettare dietro lo scrittore invece di permettergli di entrare subito a leggere
    - “Concurrent Control with "Readers" and "Writers" P.J. Courtois,* F. H, 1971”

Lettori e scrittori: costruzione di una soluzione favorevole agli scrittori

- Più processi di lettura (lettori) possono tranquillamente accedere in maniera concorrente
- Se un processo di scrittura (scrittore) sta modificando i dati, nessun altro processo deve poter accedere, nemmeno i lettori (che potrebbero leggere dei dati inconsistenti)
  - *il primo scrittore blocca l'accesso ai lettori*
  - *l'ultimo scrittore sblocca l'accesso ai lettori*

Lettori e scrittori: costruzione di una soluzione favorevole agli scrittori

```
semaphore fs = 1; /* Flag scrittori: down quando entra primo scrittore */
semaphore nsm = 1;
```

```
down(&nsm);
numscr = numscr + 1;
if (numscr == 1) {
   down(&fs);
}
up(&nsm);
```

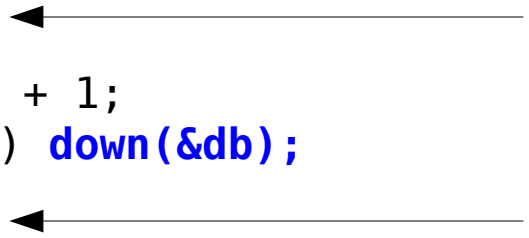
```
down(&nsm);
numscr = numscr - 1;
if (numscr == 0) {
   up(&fs);
}
up(&nsm);
```

```
thread scrittore(void)
{
    while (TRUE) {
        genera_dati();
        down(&db);
        scrivi();
        up(&db);
    }
}
```

"quando un lettore arriva e c'è uno scrittore in attesa, il lettore deve aspettare"

Lettori e scrittori: costruzione di una soluzione favorevole agli scrittori

```
semaphore nlm = 1;
semaphore db = 1;
int numlet = 0;

thread lettore(void)
{
    while (TRUE) {
        down(&nlm);
        numlet = numlet + 1;
        if (numlet == 1) down(&db);
        up(&nlm);
        leggi();
        down(&nlm);
        numlet = numlet - 1;
        if (numlet == 0) up(&db);
        up(&nlm);
        processa_dati();
    }
}
```



Nota: sblocco subito il semaforo fs per permettere la lettura concorrente di più lettori

"quando un lettore arriva e c'è uno scrittore in attesa, il lettore deve aspettare"

Lettori e scrittori: costruzione di una soluzione favorevole agli scrittori

```
semaphore nlm, nsm = 1;
semaphore fs = 1;
semaphore db = 1;
int numlet = 0;
int numscr = 0;

thread lettore(void)
{
    while (TRUE) {
        down(&fs);
        down(&nlm);
        numlet = numlet + 1;
        if (numlet == 1) down(&db);
        up(&nlm);
        up(&fs);

        leggi();

        down(&nlm);
        numlet = numlet - 1;
        if (numlet == 0) up(&db);
        up(&nlm);
        processa_dati();
    }
}
```

```
thread scrittore(void)
{
    while (TRUE) {
        genera_dati();
        down(&nsm);
        numscr = numscr + 1;
        if (numscr == 1) down(&fs);
        up(&nsm);
        down(&db);
        scrivi();
        up(&db);

        down(&nsm);
        numscr = numscr - 1;
        if (numscr == 0) up(&fs);
        up(&nsm);
    }
}
```

... manca ancora qualcosa: se sia dei lettori che degli scrittori sono in attesa su down(&fs), non viene data la precedenza agli scrittori!

Lettori e scrittori: soluzione favorevole agli scrittori

```
semaphore mutex, nlm, msm = 1;
semaphore fs = 1; /* Flag scrittori */
semaphore db = 1;
int numlet = 0;
int numscr = 0;

thread lettore(void)
{
    while (TRUE) {
        down(&mutex);
        down(&fs);
        down(&nlm);
        numlet = numlet + 1;
        if (numlet == 1) down(&db);
        up(&nlm);
        up(&fs);
        up(&mutex);

        leggi();

        down(&nlm);
        numlet = numlet - 1;
        if (numlet == 0) up(&db);
        up(&nlm);
        processa_dati();
    }
}
```

```
thread scrittore(void)
{
    while (TRUE) {
        genera_dati();
        down(&nsm);
        numscr = numscr + 1;
        if (numscr == 1) down(&fs);
        up(&nsm);
        down(&db);
        scrivi();
        up(&db);

        down(&nsm);
        numscr = numscr - 1;
        if (numscr == 0) up(&fs);
        up(&nsm);
    }
}
```

Con mutex non posso avere più lettori in attesa su down(&fs), quindi quando faccio up(&fs) se sblocco qualcuno in attesa può trattarsi solo di uno scrittore

Lettori e scrittori: soluzione favorevole agli scrittori

```
semaphore mutex, nlm, msm = 1;
semaphore fs = 1; /* Flag scrittori */
semaphore db = 1;
int numlet = 0;
int numscr = 0;

thread lettore(void)
{
    while (TRUE) {
        down(&mutex);
        down(&fs);
        down(&nlm);
        numlet = numlet + 1;
        if (numlet == 1) down(&db);
        up(&nlm);
        up(&fs);
        up(&mutex);

        leggi();

        down(&nlm);
        numlet = numlet - 1;
        if (numlet == 0) up(&db);
        up(&nlm);
        processa_dati();
    }
}
```

Vogliamo garantire che ci sia al massimo un lettore in attesa su '&l'

Se non ci sono scrittori qui non devo aspettare, altrimenti loro hanno la precedenza

Mi metto in coda, se sono il primo impedisco agli scrittori di continuare

Lascio passare il prossimo

Se sono l'ultimo lettore, gli scrittori possono passare

Lettori e scrittori: soluzione favorevole agli scrittori

Mi metto in coda, se sono il primo impedisco ai lettori di mettersi in coda (aspetteranno prima)

Aspetto la possibilità di scrivere (i.e. che esca il lettore corrente), poi posso scrivere

Ho finito, sveglio un lettore che era in attesa

```
thread scrittore(void)
{
    while (TRUE) {
        genera_dati();
        down(&nsm);
        numscr = numscr + 1;
        if (numscr == 1) down(&fs);
        up(&nsm);
        down(&db);
           scrivi();
        up(&db);

        down(&nsm);
        numscr = numscr – 1;
        if (numscr == 0) up(&fs);
        up(&nsm);
    }
}
```

Favorire gli scrittori

- Se ci sono sempre scrittori, un lettore potrebbe non aver mai l'occasione di leggere:
  - **starvation**!

Lettori e scrittori: soluzione equa

- Più processi di lettura (lettori) possono tranquillamente accedere in maniera concorrente
  - *il primo lettore blocca l'accesso agli scrittori*
  - *l'ultimo lettore sblocca l'accesso agli scrittori*
- Se un processo di scrittura (scrittore) sta modificando i dati, nessun altro processo deve poter accedere, nemmeno i lettori (che potrebbero leggere dei dati inconsistenti)

- Viene stabilito l'ordine di arrivo
  - se arriva uno scrittore, aspetta finché tutti i lettori correntemente in esecuzione (non in attesa!) finiscono, e poi ha la precedenza

Soluzione equa

```
semaphore nlm, msm = 1;
semaphore ordine = 1;
semaphore db = 1;
int numlet = 0;

thread lettore(void)
{
    while (TRUE) {
        down(&ordine);
        down(&nlm);
        numlet = numlet + 1;
        if (numlet == 1) down(&db);
        up(&ordine);
        up(&nlm);

        leggi();

        down(&nlm);
        numlet = numlet - 1;
        if (numlet == 0) up(&db);
        up(&nlm);
        processa_dati();
    }
}
```

```
thread scrittore(void)
{
    while (TRUE) {
        genera_dati();
        down(&ordine);
        down(&db);
        up(&ordine);
        scrivi();
        up(&db);
    }
}
```

Presupponiamo che i thread in attesa sul semaforo ordine vengano risvegliati nell'ordine corretto

Soluzione equa

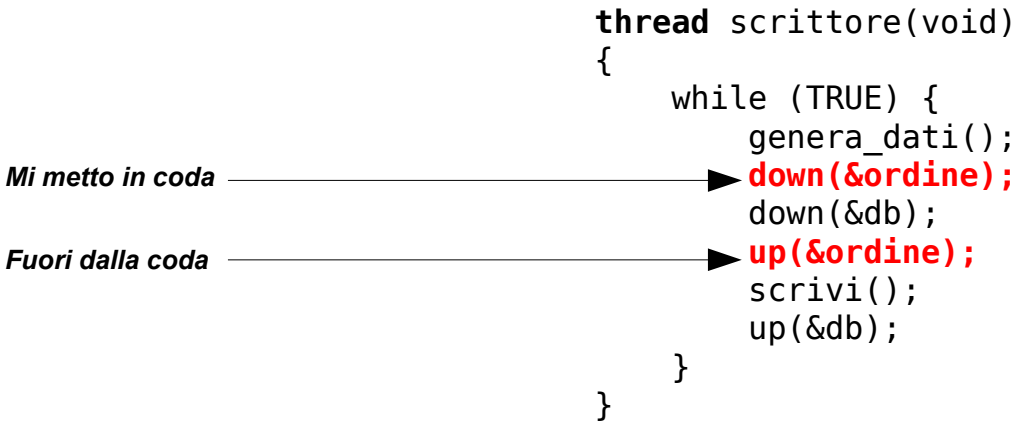
```
semaphore nlm, msm = 1;
semaphore ordine = 1;
semaphore db = 1;
int numlet = 0;

thread lettore(void)
{
    while (TRUE) {
        down(&ordine);
        down(&nlm);
        numlet = numlet + 1;
        if (numlet == 1) down(&db);
        up(&ordine);
        up(&nlm);

        leggi();

        down(&nlm);
        numlet = numlet - 1;
        if (numlet == 0) up(&db);
        up(&nlm);
        processa_dati();
    }
}
```

Mi metto in coda

Sono fuori dalla coda, pronto per leggere

Il semaforo 'db' blocca gli scrittori fintanto che ci sono lettori in esecuzione...

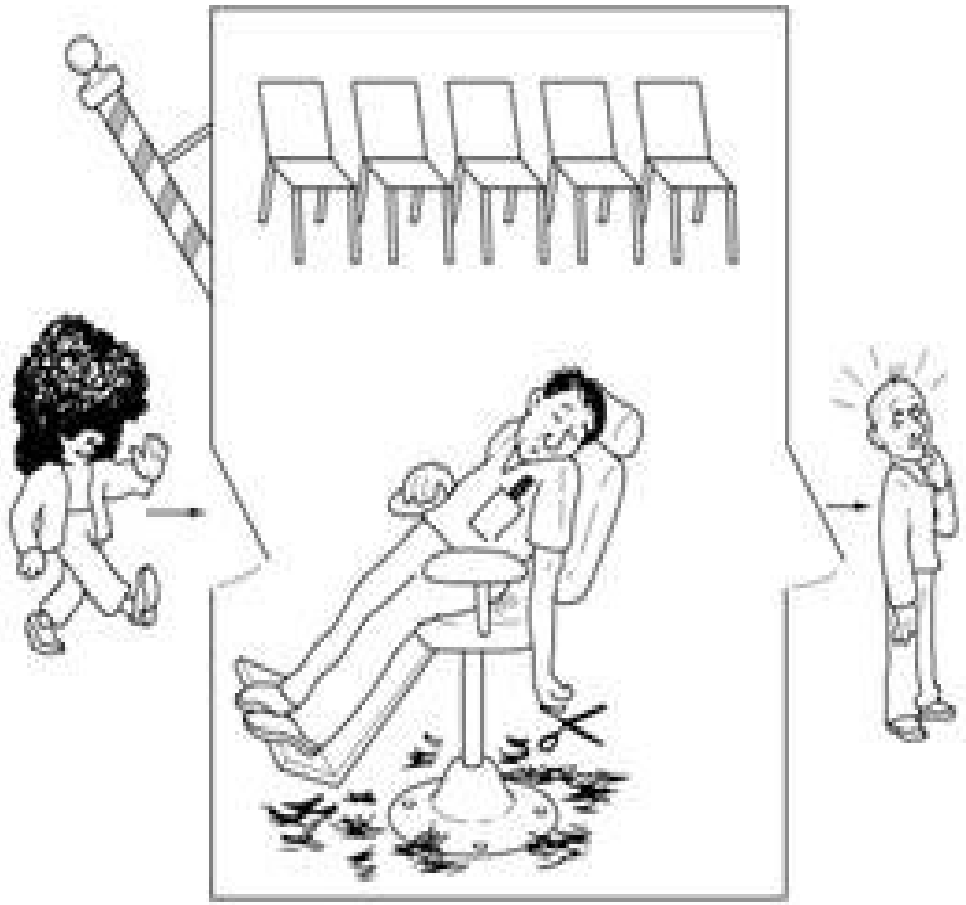
Soluzione equa




```
thread scrittore(void)
{
    while (TRUE) {
        genera_dati();
        down(&ordine);
        down(&db);
        up(&ordine);
        scrivi();
        up(&db);
    }
}
```

Appena uno scrittore viene risvegliato si mette in attesa su db... finché non ottiene l'accesso ogni altro thread dovrà aspettare su 'ordine': i lettori quindi non potranno più aumentare

Il problema del barbiere sonnolento (Sleeping Barber, E.Dijkstra – 1965)



da Modern Operating Systems, 2nd Ed., A. Tanenbaum

Il problema del barbiere sonnolento (Sleeping Barber, E.Dijkstra – 1965)

- Il barbiere ha una sedia per tagliare i capelli e per dormire e una sala d'aspetto con N sedie per i clienti.
- Il barbiere, quando ha terminato un servizio, guarda nella sala d'aspetto se ci sono altri clienti. Se sì, fa accomodare il prossimo sulla sua sedia, altrimenti si mette lui a dormire.
- Il cliente, quando arriva, guarda cosa sta facendo il barbiere. Se dorme, lo sveglia. Se sta già lavorando, va nella sala d'aspetto e, se la sala non è piena, attende il suo turno, altrimenti se ne va.

Il barbiere sonnolento: soluzione ingenua

```
semaforo_clienti = 0
semaforo_barbiere = 0
stato = dorme

thread barbiere {

  while (TRUE) {
    if (clienti > 0) {
       stato=lavora
     /* Fai accomodare cliente */
       up(semaforo_clienti)
       clienti=clienti – 1
       esegue servizio
    }
    else {
       stato=dorme
       down(semaforo_barbiere)
    }
  }
}

thread cliente {

  if (stato==dorme)/* Sveglia! */
     up(semaforo_barbiere)
  if (clienti < N) {
     /* Si mette in coda */
     clienti = clienti+1
     down(semaforo_clienti)
     riceve servizio
   }
   else
      se ne va
}
```

Il barbiere sonnolento: soluzione ingenua

- Attenzione ai **deadlock** !
  - Il barbiere guarda nella sala d'aspetto. Non c'è nessuno (clienti=0), va sulla sua sedia e dorme (stato=dorme).
  - Dopo aver visto "clienti=0" e prima di porre stato=dorme, arriva un cliente che vede il barbiere ancora stato=lavora. Va nella sala sala d'aspetto e attende l'incremento del semaforo_clienti, che non avverrà, a meno che non arrivi un altro cliente.

Il barbiere sonnolento: soluzione corretta

```
thread barbiere {

  while (TRUE) {
    blocca mutex
    if (clienti > 0) {
       stato=lavora
       up(semaforo_clienti)
       clienti=clienti – 1
       sblocca mutex
       esegue servizio
    }
    else {
       stato=dorme
       sblocca mutex
       down(semaforo_barbiere)
    }
  }
}
```

```
semaforo_clienti = 0
semaforo_barbiere = 0
stato = dorme

thread cliente {
 blocca  mutex
 if (stato==dorme)
     up(semaforo_barbiere)
  if (clienti < N) {
     clienti = clienti+1
     sblocca  mutex
     down(semaforo_clienti)
     riceve servizio
   }
   else {
      sblocca mutex
      se ne va
   }
}
```