Problema dei lettori-scrittori

Index

Introduction

Nel problema dei lettori/scrittori si ha un'**area dati condivisa** tra molti processi di cui **alcuni la leggono, altri la scrivono**

Condizioni da soddisfare:

- più lettori possono leggere l'area contemporaneamente (nei produttori/consumatori non era permesso)
- solo uno scrittore può scrivere nell'area
- se uno scrittore è all'opera sull'area, nessun lettore può effettuare letture

La vera grande differenza con i produttori/consumatori sta nel fatto che l'area condivisa **si accede per intero** (niente problemi di buffer pieno o vuoto, ma è importante permettere ai lettori di accedere contemporaneamente)

Soluzione con precedenza ai lettori

```
/* program readersandwriters */
int readcount;
semaphore x = 1, wsem = 1;
void reader()
{
    while (true) {
      semWait (x);
      readcount++;
      if (readcount == 1) semWait (wsem);
      semSignal (x);
      READUNIT();
      semWait (x);
      readcount--;
      if (readcount == 0) semSignal (wsem);
      semSignal (x);
    }
 }
void writer()
{
    while (true) {
      semWait (wsem);
      WRITEUNIT();
      semSignal (wsem);
    }
}

void main()
{
    readcount = 0;
    parbegin (reader, writer);
}
```

Il `writer` ha come unico compito quello di scrivere e lo fa attraverso un semaforo di mutua esclusione (solo un `writer` per volta può scirvere)
Il `reader` invece (come per la soluzione di trastevere) incrementa il valore di `readcount` e se si tratta del primo reader, vengono bloccati eventuali scrittori (ma se uno scrittore si trova già nella sezione critica è il lettore a bloccarsi). Viene quindi eseguita l'operazione di lettura e infine viene decrementato il valore di `readcount` e se si tratta dell'ultimo lettore, vengono sbloccati eventuali `writer`

Potrebbe accadere in questa soluzione che si vada in starvation sui `writer`

Soluzione con precedenza agli scrittori

```
/*program readersandwriters*/
int  readcount, writecount;
semaphore x = 1, y = 1, z = 1, wsem = 1, rsem = 1;
void reader()
{
    while (true) {
     semWait (z);
          semWait (rsem);
               semWait (x);
                    readcount++;
                    if (readcount == 1) semWait (wsem);
               semSignal (x);
          semSignal (rsem);
     semSignal (z);
     READUNIT();
     semWait (x);
          readcount--;
          if (readcount == 0) semSignal (wsem);
     semSignal (x);
    }
}
void writer ()
{
    while (true) {
     semWait (y);
          writecount++;
          if (writecount == 1) semWait (rsem);
     semSignal (y);
     semWait (wsem);
     WRITEUNIT();
     semSignal (wsem);
     semWait (y);
          writecount--;
          if (writecount == 0) semSignal (rsem);
     semSignal (y);
    }
}
void main()
{
    readcount = writecount = 0;
    parbegin (reader, writer);
}
```

In questo caso ho fatto ciò che prima avevo fatto solo per i `reader`, per i `writer`
Per i `reader` invece è stato aggiunto, oltre al solito semaforo locale, il semaforo `rsem` che permette, essendo `writer` controllato da un semaforo locale, di evitare che il lettore possa prevalere sullo scrittore; infatti se ad esempio, dopo aver decrementato `readcount` arriva un altro lettore, e dovrebbe continuare a leggere. Immaginiamo però che nel mentre sia arrivato uno scrittore, visto che è arrivato uno scrittore viene fatto `wait(rsem)` che blocca agli ulteriori lettori di entrare nella coda. Viene quindi finita la coda di quelli erano già riusciti a leggere, che quindi sbloccheranno il `writer`

Soluzione con i messaggi

```
// mailbox = readrequest, writerequest, finished
// send non bloccante, receive bloccante
// empty verifica se ci sono messaggi da ricevere

void reader(int i) {
        while(true) {
                nbsend(readrequest, null);
                receive(controller_pid, null);
                READUNIT();
                nbsend(finished, null);
        }
}

void writer(int j) {
        while(true) {
                nbsend(writerequest, null);
                receive(controller_pid, null);
                WRITEUNIT();
                nbsend(finished, null);
        }
}

void controller() {
        int count = MAX_READERS;
        while(true) {
                // se è positivo ci potrebbero essere dei reader
                if (count > 0) {
                        if (!empty(finished)) {
                                receive(finished, msg); /* da reader!
*/
                                // se count==MAX_READERS vuol dire
che tutti i reader
                                // hanno letto
                                count++;
                        }
                        else if (!empty(writerequest)) {
                                receive(writerequest, msg);
                                writer_id = msg.sender;
                                // se ci sono lettori count < 0
                                // se non ci sono lettori count = 0
                                count = count - MAX_READERS;
                        }
                        else if (!empty(readrequest)) {
                                // per sapere a chi far leggere
```

```
utilizzo il campo sender
                                // del messaggio da cui ho ricevuto
la richiesta
                                receive(readrequest, msg);
                                count--;
                                nbsend(msg.sender, "OK");
                        }
                }
                // non ci sono lettori, lascio scrivere il writer
                if (count == 0) {
                        nbsend(writer_id, "OK");
                        receive(finished, msg); /* da writer! */
                        count = MAX_READERS;
                }
                // ci sono lettori, aspetto la fine di ogni lettura
incrementando
                // count fino a 0 per poi poter scirvere
                while (count < 0) {
                        receive(finished, msg); /* da reader! */
                        count++;
                }
        }
}
```

Ovviamente occorre un processo di inizializzazione che crea le 3 mailbox e lancia 1 controller più reader e writer a piacimento. Ma se ci sono più di `MAX_READER-1` richieste contemporaneamente da lettori, la soluzione non funziona