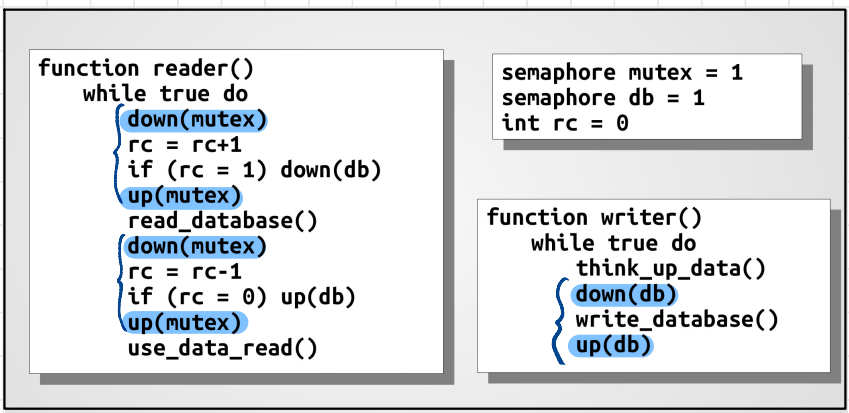
Problema dei lettori e scrittori

Questo è un problema reale che modella l'accesso ad un database, o in generale ad una struttura dove è possibile che più entità agiscano in lettura e scrittura. Le uniche operazioni compatibili tra di loro sono due accessi in lettura mentre vengono a crearsi problemi se si ha una scrittura-lettura o una scrittura-scrittura, in quanto non possono essere eseguite in modo concorrente. L'osservazione di base è che , utilizzando un mutex renderei poco efficiente l'utilizzo della struttura dati stessa, perché bloccare più accessi in lettura non crea nessun problema e quindi non ha senso bloccare a priori qualsiasi tipo di accesso concorrente alla struttura dati.

Soluzione basata sui semafori



rc è una variabile condivisa che tiene conto del numero di lettori (readers counter). Il semaforo db garantisce la mutua esclusione sull'accesso al database e il semaforo mutex garantisce l'accesso esclusivo alla variabile rc.

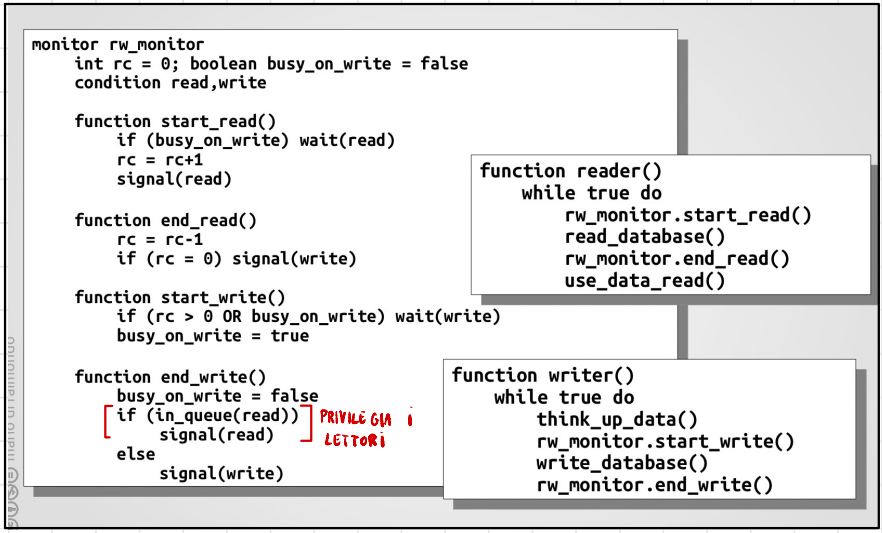
- La logica della prima sezione critica nel reader() è aggiungere un lettore ma se rc vale 1 allora uso la down(db) per bloccare l'accesso allo scrittore . Devo bloccare solo se sono il primo lettore , mentre tutti gli altri lettori non dovranno eseguire down (db). La chiamata bloccante down(db) blocca all'interno della sezione critica in modo voluto , così da bloccare tutti gli altri accessi in scrittura. Sul semaforo db si crea una coda di scrittori che verranno svegliati uno alla volta in ordine di arrivo, mentre se arrivano molti lettori appena si sblocca il primo allora esso sbloccherà in massa anche tutti gli altri che avranno accesso contemporaneamente. L'ultimo lettore si preoccupa sempre di rilasciare il blocco esclusivo sul database.

- La seconda sezione critica serve a decrementare rc dopo aver letto ed in particolare se dopo il decremento rc vale 0, allora sblocca l'accesso ad uno scrittore tramite la chiamata ad up(db).

- Se non ci sono lettori attivi allora lo scrittore prende il blocco esclusivo sul database attraverso la down(db) e up(db), in modo da bloccare l'accesso sia ad altri scrittori, sia ad altri lettori.

Questo modello blocca uno scrittore se vi sono lettori dentro e si considera che altri lettori possono in ogni caso continuare ad entrare, quindi il singolo scrittore potrebbe attendere per tempi molto lunghi e non si riesce a preventivare il tempo di attesa degli scrittori. Una prima soluzione potrebbe essere quella di bloccare i lettori che vogliono avere accesso al database se vi sono degli scrittori in attesa, in modo da garantire un tempo di attesa finito agli scrittori che stanno aspettando. Tuttavia questa soluzione sta bloccando operazioni di lettura che potevano essere eseguite in modo concorrente senza creare problemi.

Soluzione 1 (basata sui monitor) :

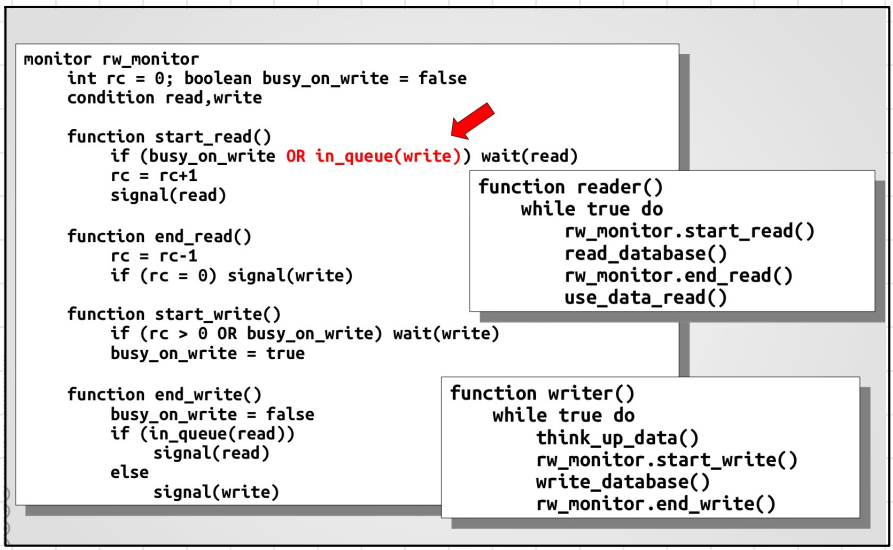


Il database non è dentro il monitor ed infatti devo implementare la mutua esclusione all'accesso sia tra più scrittori, sia tra scrittore e lettore.
Sfrutto una variabile booleana busy_on_write() che tiene conto di un'eventuale scrittura in atto sul database.
Posso iniziare a leggere tramite la procedura start_read() solo se , dopo il controllo di busy_on_write() essa sarà a false (nessuno sta scrivendo). Per quanto riguarda la signal(read) essa non ha alcun effetto al momento ma sarà necessaria successivamente per risvegliare tutti gli altri lettori in cascata non appena viene risvegliato il primo (aggancio con la end_write()).
Si ha accesso in scrittura tramite la start_write() solo se non ci sono altri lettori e se non ci sono altri scrittori in attesa , infatti c'è una wait(write) se rc > 0 || busy_on_write. A quel punto metto a true busy_on_write() segnalando che sto aspettando per scrivere. La end_write() controlla se ci sono lettori in coda con la funzione in queue(read) ed eventualmente sveglia prima i lettori accodati attraverso la signal(read), privilegiandoli rispetto agli scrittori.
Consideriamo che al risveglio del primo lettore verranno svegliati anche tutti gli altri in cascata con la signal(read) .

Questa soluzione funziona efficientemente ma **penalizza gli scrittori** che rimangono in attesta per tempi molto lunghi, in favore dei lettori che possono avere accesso alla risorsa con una priorità più alta rispetto agli scrittori.

Soluzione 2 (basata sui monitor) :

L'unica differenza sta nel fatto che ci si accoda su due variabili diverse e non su una singola variabile semaforo db come prima. Evidentemente ci sono gli stessi contro della soluzione precedente ma ci proietta verso un'altra soluzione più equa per lettori e scrittori : un lettore decide se entrare o no anche in relazione al fatto ci siano scrittori in coda e non solo scrittori già attivi nel database.



Soluzione 3 (basata sui monitor) :

Un'altra soluzione potrebbe decidere di privilegiare gli scrittori in coda attraverso l'inversione delle condizioni nella end_write() e privilegiamo gli scrittori, esattamente come la soluzione n1 privilegiava i lettori. Si può preferire questo approccio nel caso in cui l'evento di scrittura ha una priorità più alta rispetto alla lettura.