13-04-2023

Problema produttore-consumatore

In questo caso 2 processi **condividono uno STESSO BUFFER** di una certa dimensione limitata (*N*). In particolare:

- il **PRODUTTORE** inserisce dati nel buffer
- il **CONSUMATORE** legge il dato prelevato dal buffer

```
count = 0
function producer()
        while (true) do
                item = produce_item() //crea un oggetto
                if (count = N) // count indica il numero di elementi memorizzati nel buffer
                        sleep()

                insert_item(item)
                count = count + 1 //incrementa il contatore

                if (count = 1)
                        wakeup(consumer) //serve per evitare di lasciare qualche consumer in
sleep() perchè se il buffer era vuoto, c'è la possibilità che un consumatore sia in sleep
```

```
function consumer()
        while (true) do
        if (count = 0) //non ci sono item nel buffer
                sleep()
        item = remove_item()
        count = count - 1

        if (count = N - 1)
                wakeup(producer) //risveglia un potenziale producer dormiente

        consume_item(item)
```

- Il **consumatore** legge solo se ci sono informazioni e quindi il buffer **NON** è vuoto
- il **produttore** scrive sul buffer e può scrivere **SOLO se c'è spazio**, quindi se il buffer non è pieno, altrimenti va in **SLEEP** fino a quando un consumatore toglie un elemento del buffer e quindi *libera spazio*.

Limiti di questa soluzione:

- le istruzioni **non sono ATOMICHE**
- la count è slegata da sleep e wakeup e rappresenta la variabile condivisa
- Se il consumatore deve fare un'operazione con buffer vuoto, allora va in `sleep()` e viene **stoppato**
- Parte il produttore, legge count = 0, carica gli elementi e incrementa count -> vede count = 1 pensando che qualche consumatore stia dormendo e manda il segnale wakeup.

- Il consumatore riparte quando count = 0 e va in sleep dopo che il produttore ha fatto wakeup e il produttore continua a caricare
- Quindi il count vale 2,3... e non vengono eseguite altre `wakeup` di consumatore
- Ad una certa `count = N` e quindi il produttore va in sleep per buffer pieno
  Il produttore e il consumatore vanno in **sleep**
- Il problema viene sollevato perchè è stato inviato un `wakeup` inviato **A VUOTO**.
- Si risolve aggiungendo **un bit di attesa wakeup**:
  - se il consumer legge `count = 0` viene stoppato.
  - Il bit wakeup viene **settato a 1** quando viene **chiamata una wakeup dal produttore**.
  - Stavolta, il consumatore, *prima di andare in sleep*, **controlla il valore del bit wakeup** e se vale 1, allora vuol dire che è stato inviato un wakeup e non va in sleep
- Questa problematica viene risolta solo se c'è un produttore e uno/più consumatori. Se avessimo **più produttori** allora il **problema si risolleverebbe**.

Semafori

Con ***DJKISTRA*** si ha una soluzione dove viene mantenuta una variabile che viene usata per contare il numero di *wakeup* inviate, detta **SEMAFORO** ed è una variabile condivisa fra più processi.

> *Inizialmente vale 0, cioè non sono stati inviati wakeup.*

Vengono suggerite 2 operazioni: **down** e **up** (dette anche **wait** e **signal**)

> In un certo senso, quindi, l'`if` che controlla `count` e l'operazione sleep che contiene, sono unite, evitando quindi la possibilità che l'`if` possa essere sospeso prima di fare la `sleep`. Quindi viene sospeso **PRIMA oppure DOPO**, garantendo **ATOMICITA'**

- `down` prende in input un *semaforo* e ne **controlla il valore**: (**ENTRA nella sezione critica**)
  - se è `>0` allora lo **DECREMENTA** e **accede alla sezione critica** e lo fa finchè il semaforo è `>0`
  - se è `<0` allora significa che la **sezione critica è occupata** e quindi va in `sleep`
  - questa istruzione **risveglia eventuali processi dormienti**
- `up` controlla il valore del semaforo e ne **INCREMENTA** il valore (**ESCE dalla sezione critica**) ed è l'operazione che viene fatta ***subito dopo essere usciti dalla sezione critica***

> Questo sistema **evita** il problema della **BUSY WAITING** e le operazioni sono ATOMICHE (*eseguite senza interruzione*) e quindi *gestisce* la ***MUTUA ESCLUSIONE***

Problema produttore-consumatore con i semafori

Per gestire nel migliore dei modi questo problema, servono **3 semafori**:

1. uno in comune fra tutti che dice se l'operazione si può effettuare o no: nel buffer c'è il produttore oppure il consumatore quindi **garantisce la mutua esclusione**. (detto ***MUTEX***)
2. ***FULL*** = indica il **numero di posizioni piene** all'interno del buffer
3. ***EMPTY*** = indica il **numero di posizioni vuote** all'interno del buffer

Quindi, i vincoli sono:

- il produttore scrive **solo se c'è spazio**
- il consumatore legge **solo se c'è informazione**

```
int N=100
semaphore mutex = 1 //binario
semaphore empty = N
semaphore full = 0

function producer()
        while (true) do
                item = produce_item()
                down(empty) //se empty == 0 non ci sono posizioni vuote, quindi implica la sleep, altrimenti vuol dire che c'è spazio e quindi si possono inserire dati e decrementa empty
                down(mutex) // c'è qualcun'altro che sta usando il buffer. se >0 vuol dire che nessuno occupa il buffer. Se mutex = 0 vuiol dire che la sezione critica è occupata quindi va in sleep()
                insert_item(item)
                up(mutex) //segnala che lascia la condivisione del buffer
                up(full) // incremento il semaforo full di uno

function consumer() //preleva dati quando CI SONO dati
        while (true) do
                down(full) // full > 0 allora ci sono dati da prelevare e quindi continua, altrimenti sleep()
                down(mutex) // verifica se può accedere alla sezione critica
                item = remove_item()
                up(mutex) // indica che ha lasciato la sezione critica
                up(empty) // indica che c'è uno spazio vuoto in più perchè l'elemento viene rimosso

                consume_item(item) // viene usato l'item prelevato
```

Osservazioni

- Vengono usati 3 semafori con **scopi diversi**
  - **mutex** gestisce la **MUTUA ESCLUSIONE**, sul buffer si lavora singolarmente, a maggior ragione quando ci sono più produttori e consumatori
  - **full** ed **empty** vengono usati per **SINCRONIZZARE** le operazioni fra produttore e consumatore visto che serve per capire se il produttore può operare oppure deve attendere un consumatore che gli permatta di poter operare
    - il produttore fa `down(empty)` (*quindi risveglia eventuali **consumatori addormentati***) mentre il consumatore fa `up(empty)` (*quindi risveglia eventuali **produttori addormentati***)
    - Il consumatore può accedere se `full > 1`: il produttore lo incrementa e il consumatore lo decrementa. Vale il viceversa per la variabile `empty`
- l'**ordine delle operazioni** sui **semafori** è **FONDAMENTALE**:
  - Se le operazioni iniziali di produttore e consumatore vengono invertire si potrebbe creare **DEADLOCK**
  - **MAI bloccare la sezione critica se non si è sicuri di poter eseguire l'operazione**

Esempio di deadlock (inversione delle istruzioni)

```
int N=100
semaphore mutex = 1
semaphore empty = N
```

```
semaphore full = 0

function producer()
        while (true) do
                item = produce_item()
                down(mutex) //inversione
                down(empty) //inversione
                insert_item(item)
                up(mutex)
                up(full)

function consumer()
        while (true) do
                down(mutex) //inversione
                down(full) //inversione
                item = remove_item()
                up(mutex)
                up(empty)

                consume_item(item)
```

In questo caso:

- il produttore esegue down(mutex) e in quel momento è 1. Quindi viene portato a 0 e accede alla sezione critica.
- fa down(empty) e va in **sleep**
- il consumatore, quando accede alla sezione critica, trova `mutex = 0` quindi va in **sleep**.
- il produttore, con una *up(mutex)* dovrebbe *risvegliare il consumatore* ma è in sleep perchè l'ha causato down(empty) e il consumatore è in sleep per down(mutex).
- I due processi rimangono **bloccati per sempre** e si ha **DEADLOCK**