

Esercitazione [05]

Produttore/Consumatore

Leonardo Aniello – aniello@dis.uniroma1.it

Daniele Cono D'Elia – delia@dis.uniroma1.it

Giuseppe Laurenza – laurenza@dis.uniroma1.it

Federico Lombardi – lombardi@dis.uniroma1.it

Sistemi di Calcolo - Secondo modulo (SC2)

Programmazione dei Sistemi di Calcolo Multi-Nodo

Corso di Laurea in Ingegneria Informatica e Automatica

A.A. 2016-2017

Sommario

- Produttore/Consumatore: breve riepilogo
- Obiettivi dell'esercitazione
- Singolo Produttore/Singolo Consumatore in C
- Esercizio: Più Produttori/Più Consumatori

Produttore/Consumatore

- Punti chiave
 - Un buffer (array di entry) di dimensione finita/infinita
 - *Nella pratica, la dimensione è sempre finita!*
 - **Produttori**: producono entry da inserire poi nel buffer
 - **Consumatori**: consumano entry dopo averle tolte dal buffer
- Proprietà
 - Un consumatore non può togliere entry da un buffer vuoto
 - *Se il buffer è vuoto, il consumatore deve aspettare*
 - Un produttore non può inserire entry in un buffer pieno
 - Se il buffer è pieno, il produttore deve aspettare
 - Un consumatore (produttore) può togliere (inserire) una entry alla volta
 - Una entry può essere tolta e consumata (prodotta e inserita) da un solo consumatore (produttore)

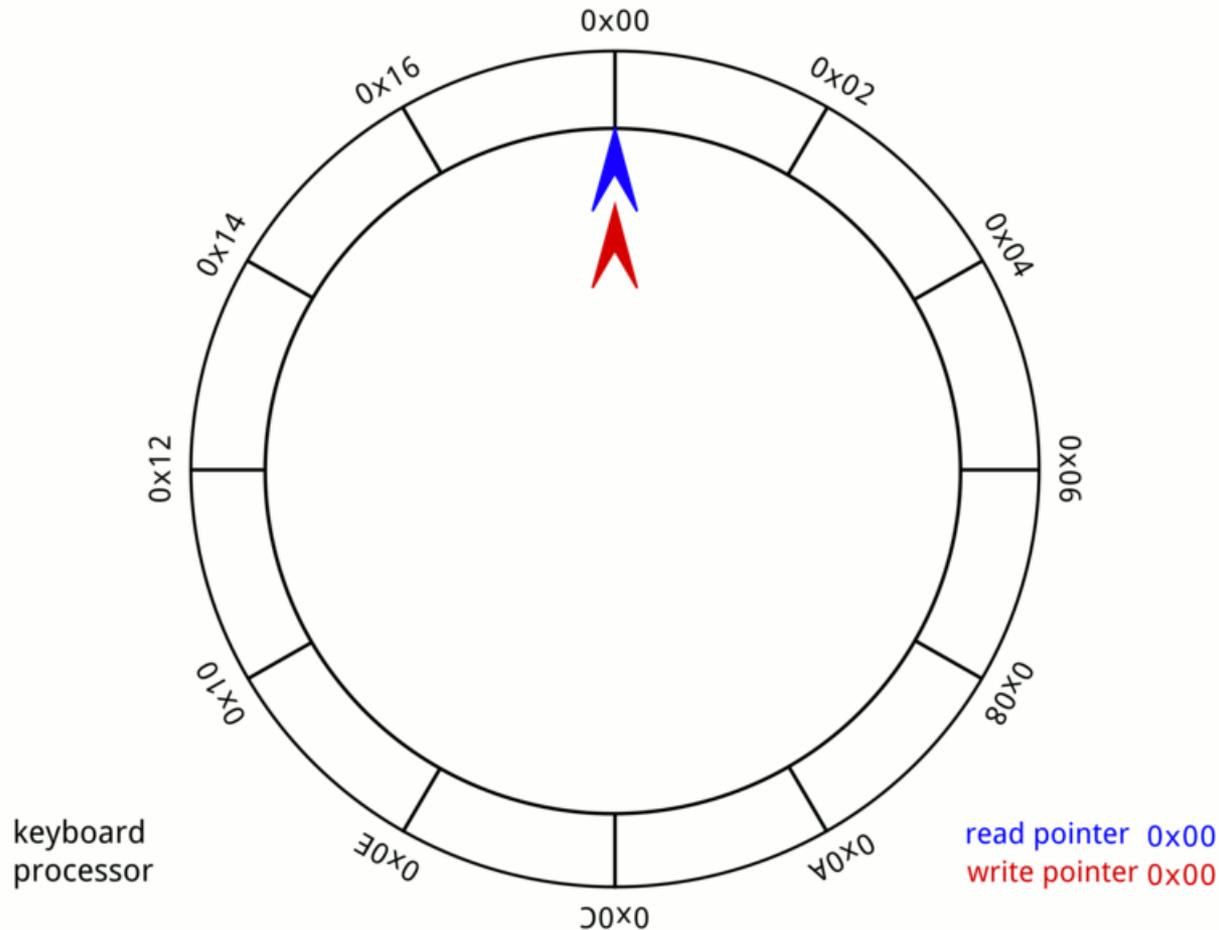
Obiettivi Esercitazione [5]

- Usare i semafori per implementare soluzioni al problema del produttore/consumatore nelle seguenti configurazioni
 - un produttore / un consumatore
 - più produttori / un consumatore
 - un produttore / più consumatori
 - più produttori / più consumatori

Buffer Circolare

- Buffer circolare di dimensione massima N
 - $write_index \in [0, N-1]$ è la posizione nel buffer dove verrà inserita la prossima entry
 - Un produttore inserisce l'entry prodotta in posizione $write_index$, poi incrementa $write_index$ di 1 (modulo N)
 - $read_index \in [0, N-1]$ è la posizione nel buffer dalla quale verrà tolta la prossima entry
 - Un consumatore toglie l'entry in posizione $read_index$, poi incrementa $read_index$ di 1 (modulo N); infine la consuma
 - Se i consumatori non tolgono entry dal buffer quando è vuoto, il $read_index$ non «scavalcherà» mai il $write_index$
 - Non vengono tolte (e quindi consumate) entry che in realtà non sono mai state inserite
 - Se i produttori non inseriscono entry nel buffer quando è pieno, il $write_index$ non «doppierà» mai il $read_index$
 - Non vengono sovrascritte entry non ancora tolte (e quindi non ancora consumate)

Buffer Circolare



By MuhannadAjjan - Own work, CC BY-SA 4.0,
<https://commons.wikimedia.org/w/index.php?curid=45368479>

Un Produttore / Un Consumatore

- Un semaforo per contare le posizioni occupate nel buffer
 - Inizializzato a 0
 - Il consumatore si blocca su questo sem. se il buffer è vuoto
 - Il produttore incrementa questo sem. quando inserisce entry
- Un semaforo per contare le posizioni libere nel buffer
 - Inizializzato a N
 - Il produttore si blocca su questo sem. se il buffer è pieno
 - Il consumatore incrementa questo sem. quando toglie entry
- **Codice:** `one_producer_one_consumer.c`
- **Compilazione:** richiede la libreria `pthread`
- **Esecuzione:** nessun parametro richiesto

Un Produttore / Un Consumatore

Approfondimento (1/2)

- Nelle slide #36 sulla concorrenza presentata a lezione viene mostrata una soluzione al problema «un consumatore/un produttore» con buffer finito
- In quel caso viene usato un ulteriore semaforo s per imporre mutua esclusione nell'accesso alla struttura dati tramite le funzioni `append()` e `take()`
- Perché nella nostra soluzione non serve?

Figure 5.13

**A Solution to
the Bounded-
Buffer
Producer/Consumer
Problem
Using
Semaphores**

```
/* program boundedbuffer */
const int sizeofbuffer = /* buffer size */;
semaphore s = 1, n= 0, e= sizeofbuffer;
void producer()
{
    while (true) {
        produce();
        semWait(e);
        semWait(s);
        append();
        semSignal(s);
        semSignal(n);
    }
}
void consumer()
{
    while (true) {
        semWait(n);
        semWait(s);
        take();
        semSignal(s);
        semSignal(e);
        consume();
    }
}
void main()
{
    parbegin (producer, consumer);
}
```

Un Produttore / Un Consumatore

Approfondimento (2/2)

- In base all'implementazione, una struttura dati condivisa può essere acceduta in modi diversi
 - Nella soluzione presentata a lezione, tramite le funzioni `append()` e `take()`
 - Nella soluzione presentata qui, tramite l'accesso esplicito ad opportune posizioni di un array
- Nella soluzione presentata a lezione, non sappiamo come quelle funzioni siano state implementate
 - In particolare, non sappiamo cosa succede se `append()` e `take()` vengono eseguite in parallelo da due thread
 - In mancanza di altre informazioni, possiamo adottare un approccio conservativo: evitiamo che quelle funzioni possano essere accedute in parallelo usando un ulteriore semaforo
- Nella soluzione presentata qui, sappiamo in che modo la struttura dati condivisa viene acceduta
 - Siccome produttore e consumatore non accedono mai in parallelo alla stessa posizione del buffer (esercizio di ragionamento: perché?), non abbiamo bisogno di ulteriori semafori

Più Produttori / Più Consumatori

- Con più produttori, è possibile che più entry vengano scritte nella stessa posizione
 - Sovrascrittura di entry = perdita di entry
 - Necessità di mettere in mutua esclusione la porzione di codice che effettua l'inserimento di entry nel buffer
- Con più consumatori, è possibile che una stessa entry venga tolta (letta e poi consumata) più volte
 - Duplicazione di entry
 - Possibile perdita di entry come conseguenza della duplicazione! (dipende dall'implementazione)
 - Necessità di mettere in mutua esclusione la porzione di codice che effettua la lettura di entry dal buffer

Esercizio

- Estendere il sorgente `one_producer_one_consumer.c` per implementare le soluzioni alle seguenti configurazioni del problema produttore/consumatore
 - più produttori/un consumatore
 - un produttore/più consumatori
 - più produttori/più consumatori

Nota: la semantica del metodo `processTransactions()` fornito nel codice per la configurazione <1 produttore, 1 consumatore> necessita di modifiche tali che, in presenza di più consumatori, anche il deposito sia aggiornato in modo race-free!