Esercizio 1

(10 min.)

Scrivere due programmi in C (***produttore*** & ***consumatore***) che una volta agganciati alla stessa memoria condivisa, operano su di essa in maniera esclusiva tramite l'utilizzo di *semafori* (System V).

Più precisamente il ***consumatore*** dovrà attendere il completamento della scrittura sulla memoria da parte del ***produttore***, che a sua volta attenderà il ***consumatore*** affinchè abbia effettivamente completato la lettura.

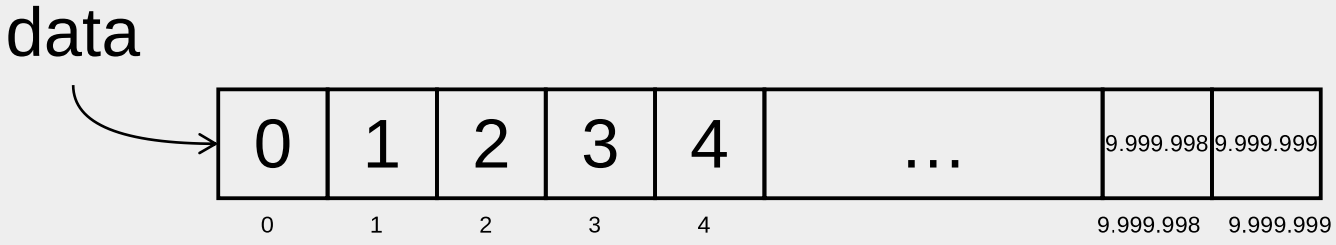
Il ***produttore*** prende il dato da ***stdin***, mentre il ***consumatore*** stampa il dato a ***stdout***.

Esercizio 2

(10 min.)

Scrivere un programmi in C a cui passate un valore ***N*** come argomento che usa per generare ***N*** nuovi threads per poi mettersi in attesa della loro terminazione.

I nuovi threads si dovranno sincronizzare per mezzo di ***pthread_spinlock/pthread_mutex*** per scrivere in maniera esclusiva all'interno di ogni singolo slot di un vettore ***data*** di **10.000.000** numeri interi, ed in maniera tale che il vettore contenga il valore ***i*** alla posizione ***i***. Ogni thread, una volta acquisito il *lock/mutex*, dovrà scrivere nello slot ***i*** non ancora scritto per poi rilaciare il *lock/mutex* e permettere quindi ad un altro thread la scrittura nello slot ***i+1***.



Esercizio 3

(10 min.)

Scrivere un programma in C che, una volta generati **64** threads, permetta solo a **4** di questi (*4 per volta, ma non obbligatoriamente sempre gli stessi 4*) di poter spinnare su di uno spinlock per accedere alla sezione critica.

Un thread che entra in sezione critica lascia un messaggio di al più 50 bytes all'interno di un buffer di caratteri globale e pre-allocato.

Devono essere scritti nel buffer esattamente 1.000.000 messaggi.

Esercizio 4

(10 min.)

Scrivere un programma in C in cui il processo padre (**Scrittore**) genera **N** processi figli (**Lettori**).

Per un numero **W** di iterazioni, il processo **Scrittore** scrive un messaggio su di una memoria condivisa che dovrà essere letto una ed una sola volta da ogni processo **Lettore** che poi tornerà in attesa del prossimo mesaggio passato dallo **Scrittore**.

Usare i semafori System V per ottenere il comportamento descritto sopra.