Ingegneria del Software

Esercitazione 6

I/O Buffer

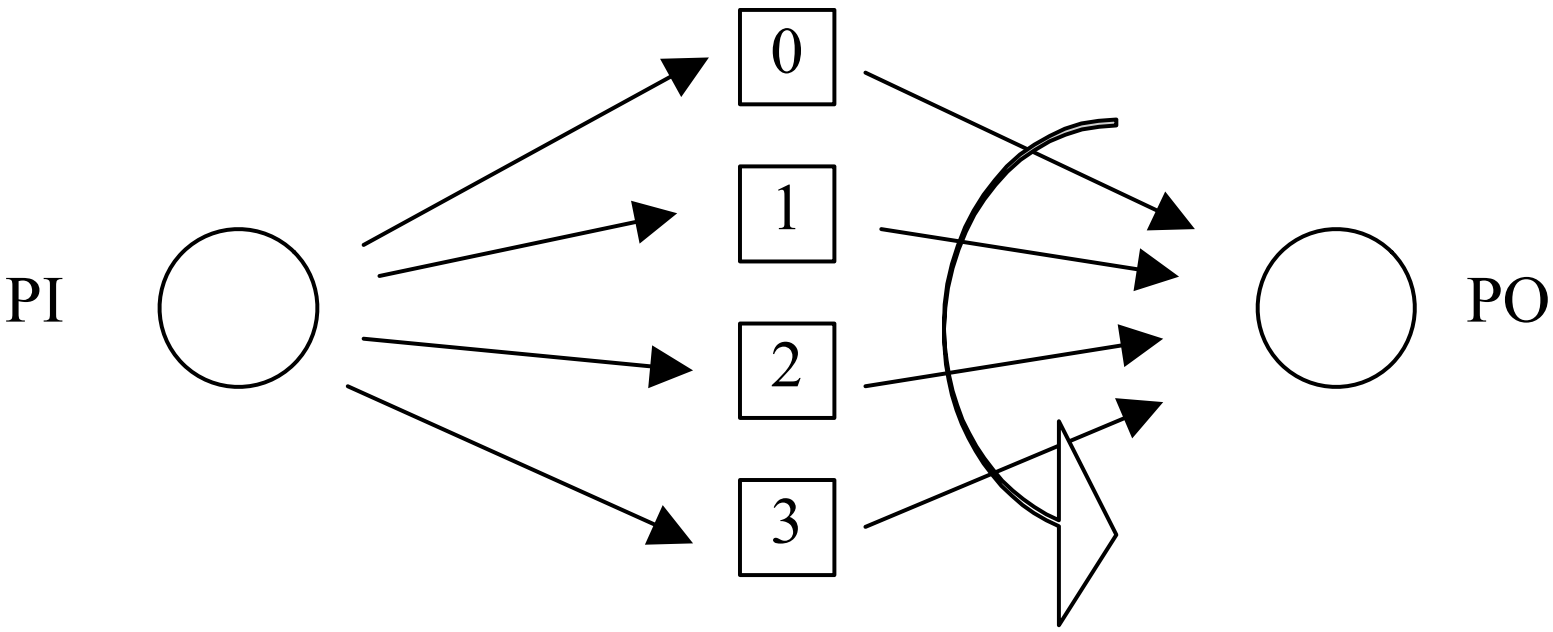
Dato un sistema con quattro buffer di caratteri.

Il modulo PI esegue ripetutamente le seguenti operazioni: legge da tastiera una coppia di valori <i, ch>, dove *i* è un numero tra 0 e 3, *ch* un carattere, e inserisce il carattere *ch* nel buffer *i* (ognuno dei quattro buffer contiene al più un carattere).

Il modulo PO considera a turno in modo circolare i quattro buffer e preleva il carattere in esso contenuto, scrivendo in uscita la coppia di valori <i, ch> se ha appena prelevato il carattere *ch* dal buffer *i*.

L'accesso a ognuno dei buffer è in mutua esclusione; PI rimane bloccato se il buffer a cui accede è pieno, PO se è vuoto.

I/O Buffer



Parte 1

- Data la seguente sequenza di valori letta da PI, scrivere la sequenza scritta in corrispondenza da PO.

  <1, c> <0, b> <2, m> <0, f> <1, h> <3, n>

Soluzione 1

<0, b> <1, c> <2, m> <3, n> <0, f> <1, h>

Parte 2

- Descrivere brevemente in quali casi si può verificare una situazione di *deadlock* tra PI e PO. Illustrare con un semplice esempio.

Soluzione 2

- Deadlock: <1,a> <1,b>

Parte 3

- Implementare il sistema in Java

Un impianto con valvola

Un impianto può portarsi dallo stato di funzionamento normale in uno stato di gestione di malfunzionamento (probabilità 20%). Entrato in tale stato, deve essere aperta una valvola di scarico (in un altro thread). La valvola si può aprire in massimo 8 secondi. Se non si apre entro 5 secondi, l'impianto passa ad uno stato di fermo. Altrimenti, se la valvola viene aperta nel tempo stabilito, essa rimane in tale stato per un tempo non inferiore a 5s e non superiore a 10s e solo dopo aver atteso lo sfogo della valvola l’impianto ritorna nello stato normale.

MinOf

Si implementi in Java il metodo statico **minOf** che riceve in input una lista di numeri interi **l** ed un numero intero **s** e che produce come output il numero minimo contenuto in **l**. Il metodo scompone logicamente la lista **l** in **s** sotto-sequenze e crea **s** thread ciascuno dedicato ad una delle sotto-sequenze. I thread hanno il compito di trovare il minimo (locale) della sotto-sequenza assegnata. Il metodo, infine, confronta i minimi locali trovati da ciascun thread per computare il minimo globale e ritornarlo come valore al chiamante.