Sistemi operativi Mod. 2 [CT0125] - Prof. R. Focardi

Verifica semafori: Imbuto 2.0

Si devono sincronizzare un certo numero di thread `pallina` in modo che entrino in un imbuto solo a gruppi di una dimensione prefissata `N` e, dopo un certo tempo, escono dall'imbuto. Solo nel momento in cui sono usciti tutti, il gruppo successivo di `N` thread `pallina` può accedere.

ATTENZIONE: In questa versione dell'esercizio non è il thread principale a sbloccare i gruppi di thread pallina, ma sono le palline stesse che si sincronizzano autonomamente.

Il thread principale si limita a inizializzare i semafori, creare i thread `pallina` attendere la loro terminazione e eliminare i semafori, secondo il seguente schema. La dimensione dei gruppi `N` viene passata a `inizializza_sem` in modo da poter essere utilizzata, successivamente, per la sincronizzazione.

```
inizializza_sem(N); // inizializza i semafori e salva N

// Crea i thread pallina e attende la loro terminazione

distruggi_sem(); // distruggi i semafori
```

Lo schema del thread "pallina" è il seguente:

```
entra_imbuto(); // attende di entrare nell'imbuto

// entra nell'imbuto e percorri tutta la strada verso il fondo

esci_imbuto(); // esce dall'imbuto
```

L'obiettivo della verifica è di implementare le 2 funzioni di sincronizzazione `entra_imbuto` e `esci_imbuto` tramite semafori (più le 2 funzioni `inizializza_sem` e `distruggi_sem` per inizializzare e eliminare i semafori) in modo da realizzare il comportamento richiesto.

IMPORTANTE: la funzione `esci_imbuto` deve necessariamente contare il numero di palline che sono uscite dall'imbuto per poi sbloccare il gruppo successivo: questa sincronizzazione non è realizzabile utilizzando solamente `wait` e `post` sui semafori. Utilizzare, a tale scopo, una **variabile globale** proteggendo opportunamente la sezione critica. Non ci sono problemi a fare delle `post` su altri semafori dentro una sezione critica perché le `post` non sono mai bloccanti.

Le funzioni da implementare sono nel file `soluzione.c` che viene incluso da `imbuto2.c` (scarica lo zip allegato). **Consegnare** solo il file `soluzione.c` . Prima di consegnare compilare con l'opzione `-DSTRESSTEST` che testa il programma con un numero elevato di thread e senza sleep (in modo da aumentare la probabilità di *race condition*). Per ripetere il test in automatico potete usare questo semplice script bash: `while true; do ./imbuto2; done`

Nota Bene:

- Programmi che **non compilano** o **non superano il test** non verranno valutati (non consegnateli).
- Solo i programmi funzionanti **verranno valutati in base ai commenti.** Commentate in maniera appropriata il vostro programma e **inserite un commento iniziale** in cui spiegate l'idea risolutiva (vedete lo schema suggerito in soluzione.c). Soluzioni non commentate **non** saranno valutate.
- **NON COPIATE E NON FATE COPIARE!** (Nel caso di soluzioni copiate verrà annullata la verifica di chi copia e chi ha fatto copiare indistintamente)

imbuto2.zip

Stato consegna

Stato consegna	Nessun tentativo
Stato valutazione	Non valutata
Termine consegne	giovedì, 2 aprile 2020, 15:30
Tempo rimasto	Consegna in ritardo da: 51 giorni 1 ora
Ultima modifica	-

Commenti alle consegne

Commenti (0)

◄ Verifica pipe: crypto

Vai a...

Verifica monitor: coda prioritaria ►

Università Ca' Foscari
Dorsoduro 3246, 30123 Venezia
PEC protocollo@pec.unive.it
P.IVA 00816350276 - CF 80007720271