

1 Problema dei filosofi a cena

Il problema della cena dei filosofi è un classico problema di condivisione delle risorse nella programmazione concorrente. Questo problema è riassunto come segue. Ci sono n filosofi seduti attorno a un tavolo. Ogni filosofo ha davanti a sé un piatto di riso e per mangiare ha bisogno di due bacchette. Ci sono un totale di n bacchette situate a sinistra di ciascun piatto. Un filosofo si comporta in questo modo: pensa, poi se vuole mangiare, prende la bacchetta sinistra, poi prende la bacchetta destra e quando finisce di mangiare restituisce le due bacchette e ricomincia a pensare, e così via. In questo esercizio desideriamo modellare il problema della cena dei filosofi con un programma C in cui ogni bacchetta sarà rappresentata da un lock e ogni filosofo sarà implementato da un thread.

1. Proponete un implementazione della cena dei filosofi con 5 filosofi. In questa implementazione, ciascun filosofo dovrà dire (stampando sul terminal) il suo stato, ad esempio 'Filosofo 1: sta pensando', 'Filosofo 1: ha la sua bacchetta sinistra', 'Filosofo 1: ha la sua bacchetta destra', 'Filosofo 1: sta mangiando', 'Filosofo 1: ha rilasciato le sue due bacchette'. Per essere sicuro che il vostro programma finisce, potete supporre che ogni filosofo vorrà mangiare 5 volte.

Consegna: Scriverete questa implementazione in un file `philos.c`.

2. Se avete rispettato le indicazione, la vostra implementazione dovrebbe produrre un deadlock dei filosofi. Provate ad esibirlo (usando per esempio dei `sleep` nel vostro codice).
3. Proponete una modificazione della vostra implementazione nel modo di evitare il deadlock.

Consegna: Scriverete questa nuova implementazione in un file `philos2.c`.

2 Barriera

Lo scopo di questo esercizio è di farvi implementare una barriera in C. Scrivete in un header file `my_barrier.h` la struttura di dati e le prototipi come indicato sotto:

```
#include <pthread.h>

typedef struct my_barrier{
    volatile unsigned int vinit;
    volatile unsigned int val;
    pthread_mutex_t lock;
    pthread_cond_t varcond;
} my_barrier;

unsigned int pthread_my_barrier_init(my_barrier *mb, unsigned int v);

unsigned int pthread_my_barrier_wait(my_barrier *mb);
```

Vi chiediamo di implementare in un file `barrier.c` le due funzione `pthread_my_barrier_init` e `pthread_my_barrier_wait` e di testarli in un `main`.

Vi ricordiamo che la barriera blocca i thread che chiamano `wait`, finché `vinit` threads hanno fatto questa chiamata, a questo punto la barriera sveglia tutti gli thread in attesa e torna nel suo stato iniziale ad aspettare `vinit` threads. La funzione `pthread_my_barrier_init` serve a inizializzare la barriera mettendo il valore `v` dentro `vinit` (il valore `val` della barriera serve a contare il numero di threads che ha fatto `wait` fino ad ora). Per questa ultima funzione, se `v` è uguale a 0, la funzione deve ritornare -1 per segnalare un problema. Ciascuna di queste due funzione ritorna 0, se tutto è andato a buon fine,

3 Consegna

Per la consegna, creare uno zip con i file `philos.c`, `philos2.c`, `my_barrier.h` e `barrier.c`. Lo zip dovrà anche contenere un file `partecipanti.txt` dove gli nomi di chi ha partecipato alla consegna (questo anche se siete da solo a farla).