[2011-12] Semafori

Ci sono `N_FILOSOFI` filosofi a pranzo serviti da un singolo cameriere. Il cameriere porta i pasti ai filosofi e li appoggia sulla tavola. C'è posto solo per `DIM_BUFFER` pasti e se non c'è posto il cameriere attende, secondo il seguente schema:

```
ini_scrivi();

   scrivi_buffer(i); // deposita il pasto i sulla tavola

end_scrivi();
```

Ogni filosofo prende il primo pasto disponibile dalla tavola, raccoglie la bacchetta sinistra poi quella destra, mangia e deposita le bacchette. Lo schema del filosofo `id` è il seguente.

```
ini_leggi();

    i=leggi_buffer(); // prende il pasto dalla tavola

end_leggi();

raccogli_sx(id);
raccogli_dx(id);

    consuma_pasto(id,i); // consuma il pasto

deposita_sx(id);
deposita_dx(id);
```

Si devono realizzare le funzioni di sincronizzazione (file filosofi.c) facendo attenzione a eventuali stalli.

```c
/* Seconda verifica di Lab Sistemi Operativi (a.a. 2011-2012)
   Ricordarsi di commentare il codice e di spiegare, brevemente, la soluzione proposta
 */

// mettere qui la dichiarazione di semafori e eventuali variabili globali

void init_sem() {}
void destroy_sem() {}

void ini_leggi() {}
void end_leggi() {}
void ini_scrivi() {}
void end_scrivi() {}

void raccogli_sx(int b) {}
void raccogli_dx(int b) {}
void deposita_sx(int b) {}
void deposita_dx(int b) {}
```

Chiamare il file `filosofi.c` e testarlo con il seguente programma:

```c
#include<stdio.h>
#include<pthread.h>
#include<stdlib.h>
#include<semaphore.h>
#include<unistd.h>
#include<stdarg.h>

#define N_PASTI 15
#define DIM_BUFFER 3
#define N_FILOSOFI 5

#include "filosofi.c"   // funzioni di sincronizzazione DA REALIZZARE PER LA VERIFICA

/********** le funzioni qui sotto sono 'ACCESSORIE' al test, non serve guardarle in dettaglio ********

// funzioni di terminazione
void die(char * s, int i) {
    printf("[ERROR] %s: %i\n",s,i);
    exit(1);
}
/*void die2(char * s) {
    printf("[SYNC ERROR] %s\n",s);
    exit(1);
}*/

void die2(char *s, ...) {
    va_list ap;

    //printf("[SYNC ERROR] ",s);

    va_start(ap, s);
    vprintf(s,ap);
    va_end(ap);

    exit(1);
}

// buffer circolare
struct {
     int buf[DIM_BUFFER];
     int inserisci;
     int preleva;
} buffer;

// scrive i nel buffer
void scrivi_buffer(int i) {
     buffer.buf[buffer.inserisci]=i;
     buffer.inserisci=(buffer.inserisci+1)%DIM_BUFFER;
}

// legge un intero dal buffer id
int leggi_buffer() {
     int j=buffer.buf[buffer.preleva];
     #ifdef CHECK_MUTEX
     sleep(1);
     #endif
     buffer.preleva=(buffer.preleva+1)%DIM_BUFFER;
     return j;
}

int bacchette_test[N_FILOSOFI]; // le bacchette, utilizzate per il test
int pasti_test[N_PASTI]; // conteggia i pasti per il test
int pasti_consumati=0; // tutti i pasti sono stati consumati

// consuma il pasto e controlla che le bacchette siano utilizzate correttamente
void consuma_pasto(int id, int i) {
     int j;
     int id_dx = (id+1)%N_FILOSOFI;

     if (bacchette_test[id]) die2("[Filosofo %i] Bacchetta %d gia' in uso\n",id,i);
     if (bacchette_test[id_dx]) die2("[Filosofo %i] Bacchetta %d gia' in uso\n",id_dx,i);

     bacchette_test[id] = bacchette_test[id_dx] = 1;

     printf("[Filosofo %i] Consumo il pasto %i\n",id,i);
     sleep(2);
```

```
78      bacchette_test[id] = bacchette_test[id_dx] = 0;
79
80      if (pasti_test[i]) {
81          die2("[ERRORE] sto per consumare il pasto %i gia' consumato in precedenza\n",i);
82      }
83      pasti_test[i]=1; // pasto consumato
84      for (j=0;j<N_PASTI && pasti_test[j];j++);
85      if (j==N_PASTI)
86          pasti_consumati=1; // e' ora di uscire
87  }
88
89  void * cameriere(void * n) {
90      int i;
91
92      for (i=0;i<N_PASTI;i++) {
93          printf("[Cameriere] Consegno il pasto %i\n",i);
94
95          ini_scrivi();
96
97            scrivi_buffer(i); // scrive i nel buffer
98
99          end_scrivi();
100     }
101 }
102
103 void * filosofo(void * n) {
104     int id = * (int *) n;
105     int i;
106
107     while(1) {
108         ini_leggi();
109
110             i=leggi_buffer(); // prende il pasto dal buffer
111
112
113         end_leggi();
114
115         printf("[Filosofo %d] Ho ricevuto il pasto %d\n",id, i);
116
117         raccogli_sx(id);
118             sleep(1); // forza il deadlock
119         raccogli_dx(id);
120
121             consuma_pasto(id,i); // consuma il pasto
122
123         deposita_sx(id);
124         deposita_dx(id);
125     }
126 }
127
128 int main() {
129     pthread_t th1[N_FILOSOFI], th2;
130     int th1_id[N_FILOSOFI];
131     int i,ret;
132
133     // inizializza i semafori
134     init_sem();
135
136     for (i=0;i<N_PASTI;i++)
137         pasti_test[i]=0; // per il test
138
139     // crea i filosofi
140     for (i=0;i<N_FILOSOFI;i++) {
141         th1_id[i]=i;
142         if((ret=pthread_create(&th1[i],NULL,filosofo,&th1_id[i])))
143             die("errore create",ret);
144         printf("Creato il filosofo %i\n", th1_id[i]);
145     }
146
147     // fa partire il cameriere un po' dopo per verificare la sincronizzazione
148     sleep(2);
149     // crea il cameriere
150     if((ret=pthread_create(&th2,NULL,cameriere,NULL )))
151         die("errore create",ret);
152     printf("Creato il cameriere\n");
153
154     /* attende la terminazione
155     for (i=0;i<N_FILOSOFI;i++)
156         if((ret=pthread_join(th1[i], NULL)))
157             die("errore join",ret);
158             */
159     if((ret=pthread_join(th2, NULL)))
```

```
160             die("errore join",ret);
161
162     for (i=0;i<5 && !pasti_consumati;i++) {
163         printf("[MAIN] Attendo che i pasti siano consumati\n");
164         sleep(10);
165     }
166
167     // elimina i semafori
168     destroy_sem();
169     if (i==5)
170         die2("I pasti non sono stati tutti consumati\n");
171     else {
172     printf("Terminato correttamente\n");
173         exit(0);
174     }
175 }
```

SOLUZIONE :

```c
#define N_PASTI 15
#define DIM_BUFFER 3
#define N_FILOSOFI 5

// semaforo del produttore (cameriere)
sem_t sem_produce;
// semaforo dei consumatori (filosofi)
sem_t sem_riceve;
// creo un semaforo x bacchetta, array di semafori usati dai filosofi per prendere le bacchette
sem_t sem_bacchette[N_FILOSOFI];
// semaforo usato dai filosofi per consumare il pasto in mutua esclusione
sem_t sem_mutex;
// semaforo per poter pranzare, possono pranzare n - 1 filosofi, per evitare il deadlock
sem_t sem_sedie;

void init_sem() {
    int i;
    // il cameriere produce (consegna) al massimo i piatti per i quali c'è posto in tavola (DIM_BUFFE
sem_init(&sem_produce, 0, DIM_BUFFER);
    // i filosofi devono ancora ricevere un piatto, inizializzo a 0 il semaforo
    sem_init(&sem_riceve, 0, 0);
// inizializzo i semafori delle bacchette a 1 così tutti possono prenderle
for(i = 0; i < N_FILOSOFI; i++) {
sem_init(&sem_bacchette[i], 0, 1);
}
    // il mutex è un semaforo binario, lo inizializzo a 1
sem_init(&sem_mutex, 0, 1);
// il semaforo delle sedie prevede che consumino al max n - 1 filosofi
sem_init(&sem_sedie, 0, N_FILOSOFI - 1);
}

void destroy_sem() {
    int i;
    sem_destroy(&sem_produce);
    sem_destroy(&sem_riceve);
    sem_destroy(&sem_mutex);
    sem_destroy(&sem_sedie);
for(i = 0; i < N_FILOSOFI; i++) {
sem_destroy(&sem_bacchette[i]);
}
}

// sincronizzo l'operazione di scrittura - produzione (il cameriere serve un piatto)
// con quella di ricezione (il primo filosofo libero riceve il piatto)
// Lettori - filosofi
void ini_leggi() {
    sem_wait(&sem_riceve); // P(riceve)
        // essendoci più filosofi devo gestire la lettura con una sezione critica
        // altrimenti uno stesso pasto potrebbe essere ricevuto, e quindi consumato, due volte
        sem_wait(&sem_mutex);
}
// il consumo del buffer avviene nel file di test
void end_leggi() {
        sem_post(&sem_mutex);
    sem_post(&sem_produce); // V(produce)
}

// Scrittori - cameriere
void ini_scrivi() {
    sem_wait(&sem_produce); // P(produce)
        // la sezione critica non serve, non ci sono interferenze tra produttori perché il cameriere
        // sem_wait(&sem_mutex);
}

// l'inserimento nel buffer avviene nel file di test
void end_scrivi() {
        //sem_post(&sem_mutex);
    sem_post(&sem_riceve); // V(riceve)
}

// raccogli = faccio due P
// b è l'id del filosofo (da 0 a 5) = all'indice della bacchetta di sx
void raccogli_sx(int b) {
    // al massimo 4 dei 5 filosofi possono concorrere a consumare il pasto
    // altrimenti si crea il deadlock per prendere le bachette
    sem_wait(&sem_sedie);
        sem_wait(&sem_bacchette[b]);
}
void raccogli_dx(int b) {
        //printf("Se tutti si fermano qui c'è dead lock\n");
        sem_wait(&sem_bacchette[(b + 1) % N_FILOSOFI]);
    sem_post(&sem_sedie);
}

// deposita = faccio due V
void deposita_sx(int b) {
        sem_post(&sem_bacchette[b]);
}
void deposita_dx(int b) {
        sem_post(&sem_bacchette[(b + 1) % N_FILOSOFI]);
}
```