[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

```
/* Seconda verifica di Lab Sistemi Operativi
   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
#include
#include
#include
#include
#include

#define N_PASTI 20
#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];
    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);

    bacchette_test[id] = bacchette_test[id_dx] = 0;

    pasti_test[i]=1; // pasto consumato
    for (j=0;j
```