[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.

```
1  /* Seconda verifica di Lab Sistemi Operativi (a.a. 2011-2012
2     Ricordarsi di commentare il codice e di spiegare, brevemente
3   */
4
5  // mettere qui la dichiarazione di semafori e eventuali vari
6
7  void init_sem() {}
8  void destroy_sem() {}
9
10 void ini_leggi() {}
11 void end_leggi() {}
12 void ini_scrivi() {}
13 void end_scrivi() {}
14
15 void raccogli_sx(int b) {}
16 void raccogli_dx(int b) {}
17 void deposita_sx(int b) {}
18 void deposita_dx(int b) {}
```

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

```
1  #include<stdio.h>
2  #include<pthread.h>
3  #include<stdlib.h>
4  #include<semaphore.h>
5  #include<unistd.h>
```

```
 6  #include<stdarg.h>
 7
 8  #define N_PASTI 15
 9  #define DIM_BUFFER 3
10  #define N_FILOSOFI 5
11
12  #include "filosofi.c"   // funzioni di sincronizzazione DA
13
14  /********** le funzioni qui sotto sono 'ACCESSORIE' al test
15
16  // funzioni di terminazione
17  void die(char * s, int i) {
18      printf("[ERROR] %s: %i\n",s,i);
19      exit(1);
20  }
21  /*void die2(char * s) {
22      printf("[SYNC ERROR] %s\n",s);
23      exit(1);
24  }*/
25
26  void die2(char *s, ...) {
27      va_list ap;
28
29      //printf("[SYNC ERROR] ",s);
30
31      va_start(ap, s);
32      vprintf(s,ap);
33      va_end(ap);
34
35      exit(1);
36
37  }
38  // buffer circolare
39  struct {
40      int buf[DIM_BUFFER];
41      int inserisci;
42      int preleva;
43  } buffer;
44
45  // scrive i nel buffer
46  void scrivi_buffer(int i) {
47      buffer.buf[buffer.inserisci]=i;
48      buffer.inserisci=(buffer.inserisci+1)%DIM_BUFFER;
49  }
50
51  // legge un intero dal buffer id
52  int leggi_buffer() {
53      int j=buffer.buf[buffer.preleva];
54      #ifdef CHECK_MUTEX
55      sleep(1);
56      #endif
57      buffer.preleva=(buffer.preleva+1)%DIM_BUFFER;
58      return j;
```

```
59  }
60
61  int bacchette_test[N_FILOSOFI]; // le bacchette, utilizzate
62  int pasti_test[N_PASTI]; // conteggia i pasti per il test
63  int pasti_consumati=0; // tutti i pasti sono stati consumat
64
65  // consuma il pasto e controlla che le bacchette siano util
66  void consuma_pasto(int id, int i) {
67      int j;
68      int id_dx = (id+1)%N_FILOSOFI;
69
70      if (bacchette_test[id]) die2("[Filosofo %i] Bacchetta %
71      if (bacchette_test[id_dx]) die2("[Filosofo %i] Bacchett
72
73      bacchette_test[id] = bacchette_test[id_dx] = 1;
74
75      printf("[Filosofo %i] Consumo il pasto %i\n",id,i);
76      sleep(2);
77
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' c
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",ic
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_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 l
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\
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     }
```

Leave a Reply

Name *

Mail * (will not be published)

Website

Comment

Submit Comment