Il problema dei filosofi a cena

Atella Luca
A.A 2019/2020
Sistemi operativi

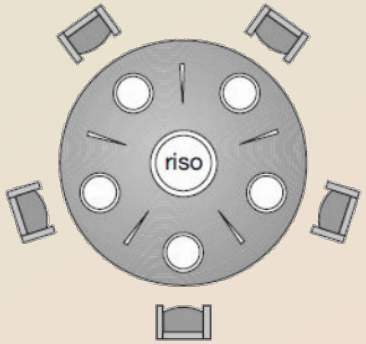
21/01/2020

prof.Domenico Bloisi

Illustrazione del problema

Il problema dei filosofi a cena è tra gli esempi di **controllo della concorrenza** più conosciuti.

- Cinque filosofi
- Cinque piatti di spaghetti
- Cinque forchette



Ogni filosofo deve avere una forchetta alla sua destra e un’altra alla sua sinistra.

Supponiamo che la cena consista in fasi alterne di mangiare/pensare e ciascun filosofo per nutrirsi abbia bisogno di due forchette e che quest'ultime vengano prese una per volta.

Dopo essere riuscito a prendere due forchette il filosofo mangia per un po', lascia le forchette e ricomincia a pensare.

Il problema della sincronizzazione

Se non si presta la dovuta attenzione alla gestione della sincronizzazione è facile imbattersi in una situazione di stallo(deadlock) o in starvation.

Deadlock

Se le risorse richieste da un thread(in stato di attesa) sono trattenute da altri thread a loro volta in stato di attesa.

Starvation

Se un thread non ottiene mai le risorse hardware/software per eseguire la sua attività.

In riferimento ai filosofi a cena

• il deadlock si verifica quando ciascun filosofo prende in mano una forchetta, e di conseguenza nessun filosofo riuscirà a prenderne l'altra.

• la starvation può verificarsi indipendentemente dal deadlock se un filosofo non riesce mai a prendere entrambe le forchette e quindi a mangiare.

Condizioni necessarie allo stallo

E’ importante prestare attenzione alle condizioni necessarie affinchè possa verificarsi una situazione di stallo.

Per evitare lo stallo non devono verificarsi i seguenti eventi:

- Mutua esclusione
- Possesso e attesa
- Assenza di prelazione
- Attesa circolare

Il codice

```
66  void creaThread(pthread_t filosofi[NUM_FILOSOFI],int* numeroForchette){
67          for(pos = 0; pos < NUM_FILOSOFI; pos++){
68                  usleep(1000000/2);
69                  pthread_create(&filosofi[pos],NULL,prendiForchetta,(void*)numeroForchette);
70                  pthread_join(filosofi[pos],0);
71          }
72  }
```

Il codice

```
38  void *prendiForchetta(void* numeroForchette){
39          int n;
40          n = NUM_FILOSOFI/2;
41          pthread_join((long unsigned int)filosofi[pos],0);
42          //cout <<"Numero forchette: " << *(int*)numeroForchette << endl;
43          if(*(int*)numeroForchette >= n && haPresoLaForchetta[pos] == false){
44                  wait(*(int*)numeroForchette);
45                  haPresoLaForchetta[pos] = true;
46                  stampaStato(pos, "HA PRESO LA FORCHETTA");
47                  cout <<"Numero forchette: " << *(int*)numeroForchette << endl;
48                  pthread_join(filosofi[pos],0);
49                  pthread_exit(0);
50                  //cout << "POS:" << pos << endl;
51          } else if(haPresoLaForchetta[pos] == true && *(int*)numeroForchette < n){
52                  stampaStato(pos,"STA MANGIANDO");
53              haPresoLaForchetta[pos] = false;
54               cout <<"Numero forchette: " << *(int*)numeroForchette << endl;
55               signal(*(int*)numeroForchette);
56               pthread_exit(0);
57          } else if(haPresoLaForchetta[pos] == false && *(int*)numeroForchette < n){
58                  stampaStato(pos,"STA PENSANDO");
59                  pthread_exit(0);
60          }
61          //cout <<"Numero forchette: " << *(int*)numeroForchette << endl;
62          pthread_exit(0);
63  }
```

La libreria pthread

Per la scrittura del codice ci siamo serviti della libreria pthread POSIX.

Procediamo con la descrizione delle funzioni principali della libreria.

LA CREAZIONE DEL THREAD

- int **pthread_create**(pthread_t * thread,const pthread_attr_t * attr,void * (*start_routine)(void *),void *arg);

➔ La funzione crea il thread e ritorna 0 in caso di successo, accetta in imput i seguenti parametri:

1) Il puntatore alla variabile contente l'id del thread

2) Gli eventuali attributi del thread (tipicamente NULL)

3) La procedura che il thread deve eseguire(con cast a void*)

4) Gli eventuali parametri della procedura da eseguire(con cast a void*)

La libreria pthread

TERMINARE UN THREAD

- void **pthread_exit**(void *retval);

➔ Questa procedura serve a terminare l'esecuzione di un thread, accetta come parametro il puntatore alla variabile che contiene il valore di ritorno(il quale potrà essere accolto da altri thread)

ATTENDERE UN THREAD

- int **pthread_join**(pthread_t th, void **thread_return);

➔ Un thread può sospendersi in attesa della terminazione dell'esecuzione di un altro thread, accetta come parametri th(il TID del thread da attendere) e il puntatore (con cast a void*) della variabile nella quale verrà memorizzato l'eventuale valore di ritorno del thread.

Semafori

Oltre alla libreria ci siamo serviti di particolari procedure che prendono il nome di “Semafori”, i quali sono tornati utili per limitare l’accesso alla nostra risorsa principale(il numero di forchette)

```
74  void wait(int& s){
75          while(s < 0);
76
77          s--;
78  }
79
80  void signal(int& s){
81          s++;
82  }
```

Esecuzione

```
luca@PC-LUCA-Linux:~/Dropbox/Università/Sistemi Operativi/Progetto$
+-------------------------BENVENUTO-------------------------+
|Quest'applicazione gestisce il problema dei cinque filosofi.|
+-----------------------------------------------------------+
FILOSOFO: 1 HA PRESO LA FORCHETTA
Numero forchette: 4
FILOSOFO: 2 HA PRESO LA FORCHETTA
Numero forchette: 3
FILOSOFO: 3 HA PRESO LA FORCHETTA
Numero forchette: 2
FILOSOFO: 4 HA PRESO LA FORCHETTA
Numero forchette: 1
FILOSOFO: 5 STA PENSANDO
FILOSOFO: 1 STA MANGIANDO
Numero forchette: 2
FILOSOFO: 5 HA PRESO LA FORCHETTA
Numero forchette: 1
FILOSOFO: 1 STA PENSANDO
FILOSOFO: 2 STA MANGIANDO
Numero forchette: 2
FILOSOFO: 1 HA PRESO LA FORCHETTA
Numero forchette: 1
FILOSOFO: 2 STA PENSANDO
FILOSOFO: 3 STA MANGIANDO
Numero forchette: 2
FILOSOFO: 2 HA PRESO LA FORCHETTA
Numero forchette: 1
FILOSOFO: 3 STA PENSANDO
FILOSOFO: 4 STA MANGIANDO
Numero forchette: 2
FILOSOFO: 3 HA PRESO LA FORCHETTA
Numero forchette: 1
```

Le situazioni da evitare sono:

- **Deadlock**
- **Starvation**(>>)

Dall'output si può facilmente evincere che il problema relativo al deadlock è stato praticamente risolto pianificando gli accessi alla risorsa nelle situazioni critiche.

(Un esempio di situazione critica è indicato con la freccia)

Esecuzione

```
FILOSOFO: 4 HA PRESO LA FORCHETTA
Numero forchette: 1
FILOSOFO: 5 STA PENSANDO
FILOSOFO: 1 STA MANGIANDO
Numero forchette: 2
FILOSOFO: 5 HA PRESO LA FORCHETTA
Numero forchette: 1
FILOSOFO: 1 STA PENSANDO
FILOSOFO: 2 STA MANGIANDO
Numero forchette: 2
FILOSOFO: 1 HA PRESO LA FORCHETTA
Numero forchette: 1
FILOSOFO: 2 STA PENSANDO
FILOSOFO: 3 STA MANGIANDO
Numero forchette: 2
FILOSOFO: 2 HA PRESO LA FORCHETTA
Numero forchette: 1
FILOSOFO: 3 STA PENSANDO
FILOSOFO: 4 STA MANGIANDO
Numero forchette: 2
FILOSOFO: 3 HA PRESO LA FORCHETTA
Numero forchette: 1
FILOSOFO: 4 STA PENSANDO
FILOSOFO: 5 STA MANGIANDO
Numero forchette: 2
FILOSOFO: 4 HA PRESO LA FORCHETTA
Numero forchette: 1
FILOSOFO: 5 STA PENSANDO
FILOSOFO: 1 STA MANGIANDO
Numero forchette: 2
^C
luca@PC-LUCA-Linux:~/Dropbox/Università/Sistemi Operativi/Progetto$
```

La situazione di starvation è evitata: ogni thread accede alla risorsa.

(la certezza si ha nel punto definito dalla freccia, tenendo in considerazione l'immagine della slide precendente, dove ogni filosofo è riuscito a mangiare)

Non ci resta che interrompere l'esecuzione premendo **Ctrl + C**

La nostra soluzione del problema dei filosofi a cena è disponibile online.

Ottenibile eseguendo il seguente comando(dopo aver installato Git):

```
➔git clone https://github.com/atellaluca/Sistemi-operativi
```