LabSO 2021

Laboratorio Sistemi Operativi - A.A. 2020-2021

dr. Andrea Naimoli	Informatica LT andrea.naimoli@unitn.it
dr. Michele Grisafi	Ingegneria informatica, delle comunicazioni ed elettronica (LT) michele.grisafi@unitn.it

Nota sugli “snippet” di codice

Alcuni esempi di codice possono essere semplificati, ad esempio omettendo il blocco principale con la funzione `main` *(che andrebbe aggiunto) oppure elencando alcune o tutte le librerie da includere tutte su una riga o insieme (per cui invece occorre trascrivere correttamente le direttive* `#include` *secondo la sintassi corretta) o altre semplificazioni analoghe. In questi casi occorre sistemare il codice perché possa essere correttamente compilato e poi eseguito.*

Mutex

Il problema della sincronizzazione

Quando eseguiamo un programma con più thread essi condividono alcune risorse, tra le quali le variabili globali. Se entrambi i thread accedono ad una sezione di codice condivisa ed hanno la necessità di accedervi in maniera esclusiva allora dobbiamo instaurare una sincronizzazione. I risultati, altrimenti, potrebbero essere inaspettati.

Per compilare correttamente un codice che utilizza i "thread" occorre aggiungere l'opzione "`-pthread`".

I thread sono sostanzialmente dei flussi di esecuzione concorrente entro uno stesso processo, quindi con meno esigenze in termini di risorse e maggior semplicità di interazione.

Esempio

```
#include <pthread.h> <stdlib.h> <unistd.h><stdio.h>     //syncProblem.c
pthread_t tid[2];
int counter = 0;
void *thr1(void *arg){
    counter = 1;
    printf("Thread 1 has started with counter %d\n",counter);
    for (long unsigned i = 0; i < (0x00FF0000); i++);
    counter += 1;
    printf("Thread 1 expects 2 and has: %d\n", counter);
}
void *thr2(void *arg){
    counter = 10;
    printf("Thread 2 has started with counter %d\n",counter);
    for (long unsigned i = 0; i < (0xFFF0000); i++);
    counter += 1;
    printf("Thread 2 expects 11 and has: %d\n", counter);
}
...
```

Esempio

```
...
void main(void){
    pthread_create(&(tid[0]), NULL, thr1,NULL);
    pthread_create(&(tid[1]), NULL, thr2,NULL);
    pthread_join(tid[0], NULL);
    pthread_join(tid[1], NULL);
}
```

Una possibile soluzione: mutex

I mutex sono dei semafori imposti ai thread. Essi possono proteggere una determinata sezione di codice, consentendo ad un thread di accedervi in maniera esclusiva fino allo sblocco del semaforo. Ogni thread the vorrà accedere alla stessa sezione di codice dovrà aspettare che il semaforo sia sbloccato, andando in sleep fino alla sua prossima schedulazione.

I mutex vanno inizializzati e poi assegnati ad una determinata sezione di codice.

Creare e distruggere un mutex

```
int pthread_mutex_init(pthread_mutex_t *restrict mutex, const
                       pthread_mutexattr_t *restrict attr)

int pthread_mutex_destroy(pthread_mutex_t *mutex)
```

```c
#include <pthread.h>                                    //createMutex.c
pthread_mutex_t lock;
int main(){
    pthread_mutex_init(&lock, NULL); // Create mutex with default attrs
    pthread_mutex_destroy(&lock); // Destroy mutex (it can be re-init)
}
```

Bloccare e sbloccare un mutex

```
int pthread_mutex_lock(pthread_mutex_t *mutex)

int pthread_mutex_unlock(pthread_mutex_t *mutex)
```

```
#include <pthread.h>
pthread_mutex_t lock;
void * thread(void *){
    pthread_mutex_lock(&lock); /* Waits for mutex to be unlocked, then
                                  locks it becoming the owner */
    pthread_mutex_unlock(&lock); //Unlock mutex allowing others to lock it
}...
```

Esempio

```
#include <pthread.h> <stdlib.h> <unistd.h><stdio.h>      //mutex.c

pthread_mutex_t lock;
pthread_t tid[2];
int counter = 0;

void* thr1(void* arg){
    pthread_mutex_lock(&lock);
    counter = 1;
    printf("Thread 1 has started with counter %d\n",counter);
    for (long unsigned i = 0; i < (0x00FF0000); i++);
    counter += 1;
    pthread_mutex_unlock(&lock);
    printf("Thread 1 expects 2 and has: %d\n", counter);
}...
```

Esempio

```
...
void* thr2(void* arg){
    pthread_mutex_lock(&lock);
    counter = 10;
    printf("Thread 2 has started with counter %d\n",counter);
    for (long unsigned i = 0; i < (0xFFF0000); i++);
    counter += 1;
    pthread_mutex_unlock(&lock);
    printf("Thread 2 expects 11 and has: %d\n", counter);
}
void main(){
    pthread_mutex_init(&lock, NULL);
    pthread_create(&(tid[0]), NULL, thr1,NULL);
    pthread_create(&(tid[1]), NULL, thr2,NULL);
    pthread_join(tid[0], NULL);
    pthread_join(tid[1], NULL);
    pthread_mutex_destroy(&lock);
}
```

Tipi di mutex

- `PTHREAD_MUTEX_NORMAL`: deadlock detection
  - Ribloccare quando bloccato → deadlock
  - Sbloccare quando bloccato da altri → undefined
  - Sbloccare quando sbloccato → undefined
- `PTHREAD_MUTEX_ERRORCHECK`: error checking
  - Ribloccare quando bloccato → error
  - Sbloccare quando bloccato da altri → error
  - Sbloccare quando sbloccato → error
- `PTHREAD_MUTEX_RECURSIVE`: multiple locks
  - Ribloccare quando bloccato → increase lock count → richiede equal numero di sbloccaggi
  - Sbloccare quando bloccato da altri → error
  - Sbloccare quando sbloccato → error
- `PTHREAD_MUTEX_DEFAULT`:
  - Ribloccare quando bloccato → undefined
  - Sbloccare quando bloccato da altri → undefined
  - Sbloccare quando sbloccato → undefined

Esempio

```
#include <pthread.h> <stdlib.h> <unistd.h><stdio.h>     //recursive.c

pthread_mutex_t lock;
pthread_t tid[2];
int counter = 0;

void* thr1(void* arg){
    pthread_mutex_lock(&lock);
    pthread_mutex_lock(&lock);
    counter = 1;
    printf("Thread 1 has started with counter %d\n",counter);
    for (long unsigned i = 0; i < (0x00FF0000); i++);
    counter += 1;
    pthread_mutex_unlock(&lock);
    printf("Thread 1 expects 2 and has: %d\n", counter);
    pthread_mutex_unlock(&lock);
}...
```

Esempio

```
...
void* thr2(void* arg){
    pthread_mutex_lock(&lock); pthread_mutex_lock(&lock);
    counter = 10;
    printf("Thread 2 has started with counter %d\n",counter);
    for (long unsigned i = 0; i < (0xFFF0000); i++);
    counter += 1;
    pthread_mutex_unlock(&lock);  pthread_mutex_unlock(&lock);
    printf("Thread 2 expects 11 and has: %d\n", counter);
}
void main(){
    pthread_mutexattr_t attr;
    pthread_mutexattr_settype(&attr,PTHREAD_MUTEX_RECURSIVE);
    pthread_mutex_init(&lock, &attr);
    pthread_create(&(tid[0]), NULL, thr1,NULL);
    pthread_create(&(tid[1]), NULL, thr2,NULL);
    pthread_join(tid[0], NULL); pthread_join(tid[1], NULL);
    pthread_mutex_destroy(&lock);
}
```

CONCLUSIONI

I “MUTEX” sono un metodo semplice ma efficace per eseguire sezioni critiche in processi multithread.

È importante limitare al massimo la sezione critica utilizzando lock/unlock per la porzione di codice più piccola possibile e solo se assolutamente necessario quando possano capitare accessi concorrenti.