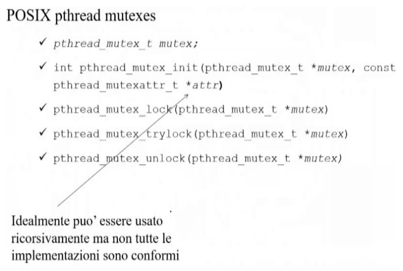
Sincronizzazione 7 Mutex UNIX

martedì 25 novembre 2025 16:06

Sono delle strutture basate sulla mutua esclusione : binario. In dettaglio:

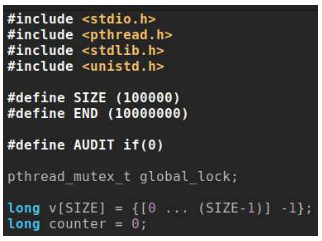


POSIX pthread mutexes

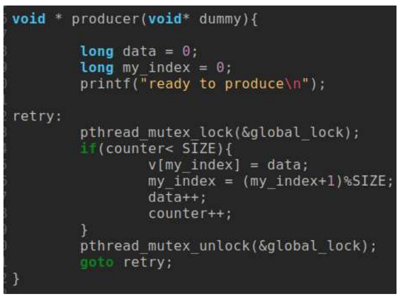
- ✓ *pthread_mutex_t mutex;*
- ✓ `int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr)`
- ✓ `pthread_mutex_lock(pthread_mutex_t *mutex)`
- ✓ `pthread_mutex_trylock(pthread_mutex_t *mutex)`
- ✓ `pthread_mutex_unlock(pthread_mutex_t *mutex)`

Idealmente puo' essere usato ricorsivamente ma non tutte le implementazioni sono conformi

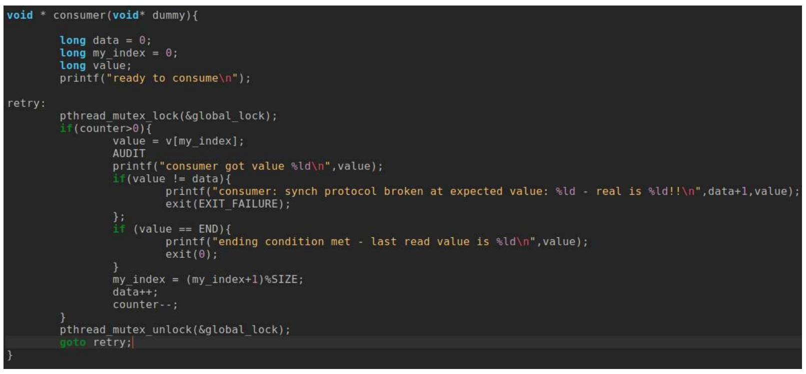
Vediamo un esempio:



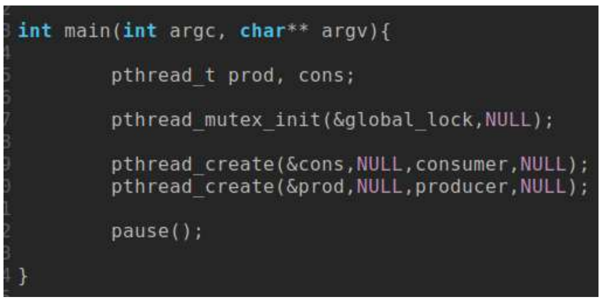
```c
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>

#define SIZE (100000)
#define END (10000000)

#define AUDIT if(0)

pthread_mutex_t global_lock;

long v[SIZE] = {[0 ... (SIZE-1)] -1};
long counter = 0;
```



```c
void * producer(void* dummy){

        long data = 0;
        long my_index = 0;
        printf("ready to produce\n");

retry:
        pthread_mutex_lock(&global_lock);
        if(counter< SIZE){
                v[my_index] = data;
                my_index = (my_index+1)%SIZE;
                data++;
                counter++;
        }
        pthread_mutex_unlock(&global_lock);
        goto retry;
}
```

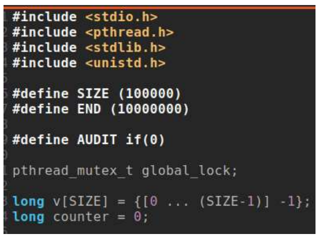


```c
void * consumer(void* dummy){

        long data = 0;
        long my_index = 0;
        long value;
        printf("ready to consume\n");

retry:
        pthread_mutex_lock(&global_lock);
        if(counter>0){
                value = v[my_index];
                AUDIT
                printf("consumer got value %ld\n",value);
                if(value != data){
                        printf("consumer: synch protocol broken at expected value: %ld - real is %ld!!\n",data+1,value);
                        exit(EXIT_FAILURE);
                };
                if (value == END){
                        printf("ending condition met - last read value is %ld\n",value);
                        exit(0);
                }
                my_index = (my_index+1)%SIZE;
                data++;
                counter--;
        }
        pthread_mutex_unlock(&global_lock);
        goto retry;
}
```

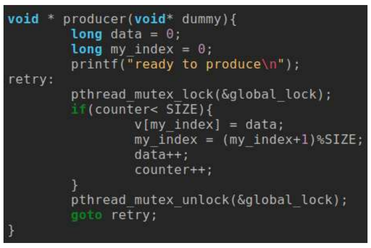


```c
int main(int argc, char** argv){

        pthread_t prod, cons;

        pthread_mutex_init(&global_lock,NULL);

        pthread_create(&cons,NULL,consumer,NULL);
        pthread_create(&prod,NULL,producer,NULL);

        pause();

}
```

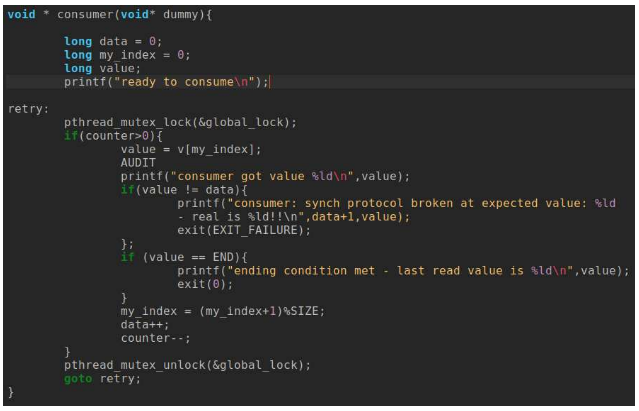
Unico mutex globale. Nella versione 1 abbiamo due serie di mutex :



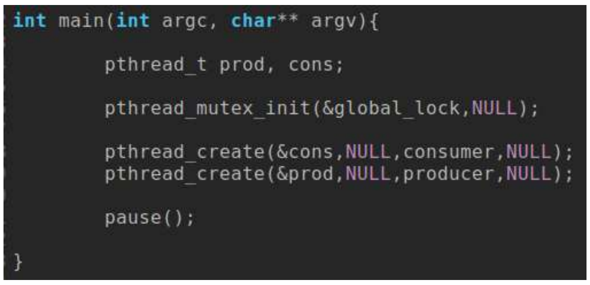
```c
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>

#define SIZE (100000)
#define END (10000000)

#define AUDIT if(0)

pthread_mutex_t global_lock;

long v[SIZE] = {[0 ... (SIZE-1)] -1};
long counter = 0;
```



```c
void * producer(void* dummy){
        long data = 0;
        long my_index = 0;
        printf("ready to produce\n");
retry:
        pthread_mutex_lock(&global_lock);
        if(counter< SIZE){
                v[my_index] = data;
                my_index = (my_index+1)%SIZE;
                data++;
                counter++;
        }
        pthread_mutex_unlock(&global_lock);
        goto retry;
}
```



```c
void * consumer(void* dummy){

        long data = 0;
        long my_index = 0;
        long value;
        printf("ready to consume\n");

retry:
        pthread_mutex_lock(&global_lock);
        if(counter>0){
                value = v[my_index];
                AUDIT
                printf("consumer got value %ld\n",value);
                if(value != data){
                        printf("consumer: synch protocol broken at expected value: %ld
                        - real is %ld!!\n",data+1,value);
                        exit(EXIT_FAILURE);
                };
                if (value == END){
                        printf("ending condition met - last read value is %ld\n",value);
                        exit(0);
                }
                my_index = (my_index+1)%SIZE;
                data++;
                counter--;
        }
        pthread_mutex_unlock(&global_lock);
        goto retry;
}
```



```c
int main(int argc, char** argv){

        pthread_t prod, cons;

        pthread_mutex_init(&global_lock,NULL);

        pthread_create(&cons,NULL,consumer,NULL);
        pthread_create(&prod,NULL,producer,NULL);

        pause();

}
```