

Mutua esclusione

- Una variabile condizione non fornisce la mutua esclusione.
- C'è bisogno di un mutex per poter sincronizzare l'accesso ai dati.

Sincronizzazione

- Una variabile condition è **sempre** associata ad un mutex
 - un thread ottiene il mutex e testa il predicato
 - se il predicato è verificato allora il thread esegue le sue operazioni e rilascia il mutex
 - se il predicato non è verificato, in modo atomico
 - ⇒ il mutex viene rilasciato (implicitamente)
 - ⇒ il thread si blocca sulla variabile condition
 - un thread bloccato riacquisisce il mutex nel momento in cui viene svegliato da un altro thread

Creazione condition

- Oggetti di sincronizzazione su cui un processo si può bloccare in attesa
 - associate ad una condizione logica arbitraria
 - Usati insieme ai mutex per implementare i *Monitor*
 - nuovo tipo `pthread_cond_t`
 - attributi variabili condizione di tipo `pthread_condattr_t`

Inizializzazione statica

```
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

- Per il tipo di dato `pthread_cond_t`, è definita la macro di inizializzazione `PTHREAD_COND_INITIALIZER`

Inizializzazione dinamica

```
int pthread_cond_init( pthread_cond_t *cond,  
                      pthread_condattr_t *cond_attr )
```

- `pthread_cond_t *cond`
 - puntatore ad un'istanza di condition che rappresenta la condizione di sincronizzazione
- `pthread_condattr_t *cond_attr`
 - punta a una struttura che contiene gli attributi della condizione
 - se NULL usa valori di default

Distruzione variabili condition

```
int pthread_cond_destroy( pthread_cond_t *cond )
```

- Dealloca tutte le risorse allocate per gestire la variabile condizione specificata
- Non devono esistere thread in attesa della condizione
- `pthread_cond_t *cond`
 - puntatore ad un'istanza di condition da distruggere
- Valore di ritorno
 - 0 in caso di successo oppure un codice d'errore $\neq 0$

Interfacce

- Operazioni fondamentali:
 - `wait` (*sospensione*)
 - `signal` (*risveglio*)
 - `broadcast` (*risveglia tutti*)

Interfaccia `wait`

- La `wait` serve per sincronizzarsi con una certa condizione all'interno di un blocco di dati condivisi e protetti da un `mutex`
- La presenza del `mutex` fra i parametri garantisce che, al momento del bloccaggio, esso venga liberato, eliminando a monte possibili errori di programmazione che potrebbero condurre a condizioni di deadlock.
- Se la `wait` ritorna in modo regolare, è garantito che la mutua esclusione, sul semaforo `mutex` passato, sia stata nuovamente acquisita.

wait

```
int pthread_cond_wait( pthread_cond_t *cond,  
                      pthread_mutex_t *mutex )
```

- `pthread_cond_t *cond`
 - puntatore ad un'istanza di condition che rappresenta la condizione di sincronizzazione
 - puntatore all'oggetto condizione su cui bloccarsi
- `pthread_mutex_t *mutex`
 - l'indirizzo di un semaforo di mutua esclusione necessario alla corretta consistenza dei dati
- Valore di ritorno
 - 0 in caso di successo

Interfaccia `signal`

- **La `signal` non si preoccupa di liberare la mutua esclusione, infatti, fra i suoi parametri non c'è il mutex**
- **Il mutex deve essere rilasciato esplicitamente, altrimenti si potrebbe produrre una condizione di deadlock.**
- **Due varianti**
 - **Standard: sblocca un solo thread bloccato**
 - **Broadcast: sblocca tutti i thread bloccati**

signal

```
int pthread_cond_signal ( pthread_cond_t *cond)
```

- Se esistono thread sospesi nella coda associata a `cond`, ne viene risvegliato uno (quale? Dipende dalla politica di scheduling).
- Se non vi sono thread sospesi sulla condizione, la `signal` **non ha effetto**.
- `pthread_cond_t *cond`
 - puntatore all'oggetto condizione
- Valore di ritorno
 - 0 in caso di successo

broadcast

```
int pthread_cond_broadcast ( pthread_cond_t *cond )
```

- Se esistono thread sospesi nella coda associata a `cond`, vengono svegliati tutti
 - altrimenti nessun effetto
- `pthread_cond_t *cond`
 - puntatore all'oggetto condizione
- Valore di ritorno
 - 0 in caso di successo

Valutazione condizione

- Il thread svegliato deve rivalutare la condizione
 - l'altro thread potrebbe non aver testato la condizione
 - la condizione potrebbe essere cambiata nel frattempo
 - possono verificarsi wakeup “spuri”

```
pthread_mutex_lock(&mutex);  
while(!condition_to_hold)  
    pthread_cond_wait(&cond, &mutex);  
computation();  
pthread_mutex_unlock(&mutex);
```

while VS if

```
pthread_mutex_lock();  
while(predicate==0) {  
    wait();  
}  
predicate--;  
pthread_cond_signal();  
pthread_mutex_unlock();
```

```
pthread_mutex_lock();  
if(predicate==0) {  
    wait();  
}  
predicate--;  
pthread_cond_signal();  
pthread_mutex_unlock();
```

Stato della coda

- Non è prevista una funzione per verificare lo stato della coda associata a una condizione.

Esempio di utilizzo (1 di 2)

- Risorsa che può essere usata contemporaneamente da MAX thread.
 - condition PIENO per la sospensione dei thread
 - M mutex associato a pieno
 - N_int numero di thread che stanno utilizzando la risorsa

```
#define MAX 100
/*variabili globali*/
int N_in=0 /*numero thread che stanno utilizzando la
risorsa*/
pthread_cond_t PIENO;
pthread_mutex_t M; /*mutex associato alla cond. PIENO*/
```

Esempio di utilizzo (2 di 2)

```
void codice_thread() {
    /*fase di entrata*/
    pthread_mutex_lock(&M);
    /* controlla la condizione di ingresso*/
    while(N_in == MAX)
        pthread_cond_wait(&PIENO, &M);
    /*aggiorna lo stato della risorsa */
    N_in ++;
    pthread_mutex_unlock(&M);
    <uso della risorsa>
    /*fase di uscita*/
    pthread_mutex_lock(&M);
    /*aggiorna lo stato della risorsa */
    N_in --;
    pthread_cond_signal(&PIENO);
    pthread_mutex_unlock(&M);
}
```

Esempio 4 (1 di 4)

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

/* mutex */
pthread_mutex_t condition_mutex = PTHREAD_MUTEX_INITIALIZER;

/* condition variable*/
pthread_cond_t condition_cond = PTHREAD_COND_INITIALIZER;
```

Esempio 4 (2 di 4)

```
void thread1_func(void *ptr) {
    printf("Avvio dell'esecuzione del %s.\n", (char *)ptr);
    sleep(2); /* pausa di 2 secondi */

    printf("Thread 1 in procinto di entrare nella sezione critica.\n");
    pthread_mutex_lock(&condition_mutex);
    printf("Thread 1 nella sezione critica.\n");

    printf("Thread 1 si sospende sulla condition variable.\n");
    pthread_cond_wait(&condition_cond, &condition_mutex);

    printf("Thread 1 riprende l'esecuzione.\n");

    printf("Thread 1 in procinto di uscire dalla sezione critica.\n");
    pthread_mutex_unlock(&condition_mutex);

    printf("Thread 1 in procinto di terminare.\n");
    pthread_exit(&ret2);
}
```

Esempio 4 (3 di 4)

```
void thread2_func(void *ptr) {
    printf("Avvio dell'esecuzione del %s.\n", (char *)ptr);
    sleep(5); /* pausa di 5 secondi */

    printf("Thread 2 in procinto di entrare nella sezione critica.\n");
    pthread_mutex_lock(&condition_mutex);
    printf("Thread 2 nella sezione critica.\n");

    printf("Thread 2 segnala l'evento della condition variable.\n");
    pthread_cond_signal(&condition_cond);

    printf("Thread 2 in procinto di uscire dalla sezione critica.\n");
    pthread_mutex_unlock(&condition_mutex);

    printf("Thread 2 in procinto di terminare.\n");
    pthread_exit(&ret2);
}
```

Esempio 4 (4 di 4)

```
main() {
    pthread_t thread1, thread2;
    char *msg1="Thread 1";
    char *msg2="Thread 2";
    if(pthread_create(&thread1, NULL, (void *)&thread1_func, (void *)msg1) != 0) {
        perror("Errore nella creazione del primo thread.\n");
        exit(1);
    }
    if(pthread_create(&thread2, NULL, (void *)&thread2_func, (void *)msg2) != 0) {
        perror("Errore nella creazione del secondo thread.\n");
        exit(1);
    }
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    exit(0);
}
```

Esempio 5: incremento contatore (1 di 3)

```
void *inc_count(void *idp) {
    int j,i; double result=0.0; int *my_id = idp;
    for (i=0; i<TCOUNT; i++) {
        pthread_mutex_lock(&count_mutex);
        count++;
        if (count == COUNT_LIMIT) { /* Check the value of count and signal */
            pthread_cond_signal(&count_threshold_cv);
            printf("inc_count(): thread %d, count = %d Threshold reached.\n", *my_id, count);
        }
        printf("inc_count(): thread %d,count = %d, unlocking mutex\n", *my_id, count);
        pthread_mutex_unlock(&count_mutex);

        /* Do some work so threads can alternate on mutex lock */
        for (j=0; j<1000; j++)
            result = result + (double)random();
    }
    pthread_exit(NULL);
}
```

Esempio 5: incremento contatore (2 di 3)

```
void *watch_count(void *idp) {

    int *my_id = idp;
    printf("Starting watch_count(): thread %d\n", *my_id);
    pthread_mutex_lock(&count_mutex);    /*Lock mutex and wait for signal.
*/

    while (count<COUNT_LIMIT) {
        pthread_cond_wait(&count_threshold_cv, &count_mutex);
        printf("watch_count():thread %d Condition signal
received.\n", *my_id);
    }

    pthread_mutex_unlock(&count_mutex);
    pthread_exit(NULL);
}
```

Esempio 5: incremento contatore (3 di 3)

```
int main (int argc, char *argv[]) {
    int i, res;
    pthread_t threads[3];
    pthread_attr_t attr;
    /* Initialize mutex and condition variable objects */
    pthread_mutex_init(&count_mutex, NULL);
    pthread_cond_init (&count_threshold_cv, NULL);

    /* Create threads and wait for all threads to complete */
    ...
    /* Clean up and exit */
    pthread_attr_destroy(&attr);
    pthread_mutex_destroy(&count_mutex);
    pthread_cond_destroy(&count_threshold_cv);
    pthread_exit(NULL);
}
```

Esempio 6 (1 di 6)

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>

/* Numero di cicli di lettura/scrittura che vengono fatti dai thread */
#define CICLI 1
/* Lunghezza del buffer */
#define LUN 20
/* Numero di cicli di attesa a vuoto di uno scrittore */
#define DELAY_WRITER 200000
/* Numero di cicli di attesa a vuoto di uno scrittore */
#define DELAY_READER 2000000
```

Esempio 6 (2 di 6)

```
/* Memoria Condivisa fra i thread ... */
struct {
    /* Semaforo di mutua esclusione */
    pthread_mutex_t mutex;
    /* Variabile condition per il lettore */
    pthread_cond_t lettore;
    /* Variabile condition per gli scrittori */
    pthread_cond_t scrittori;
    /* Buffer */
    char scritta[LUN+1];
    /* Variabili per la gestione del buffer */
    int primo, ultimo, elementi;
    /* Numero di lettori e scrittori bloccati */
    int blockscri, blocklet;
} shared = {PTHREAD_MUTEX_INITIALIZER,
            PTHREAD_COND_INITIALIZER, PTHREAD_COND_INITIALIZER};
```

Esempio 6 (3 di 6)

```
int main(void){
    pthread_t s1TID, s2TID, lTID;
    int res, i;
    /* Inizializzo la stringa scritta */
    for(i=0; i<LUN; i++) {
        shared.scritta[i] = 'x';
    }
    /* Ogni stringa C deve terminare con 0!!!! */
    shared.scritta[LUN] = 0;
    shared.primo = shared.ultimo = shared.elementi = 0;
    shared.blocklet = shared.blockscri = 0;

    /* Setto il concurrency level a 3 cioè si comunica al sistema il numero di
    thread che si intende creare*/
    pthread_setconcurrency(3);

    /* A questo punto posso creare i thread .... */
```

Esempio 6 (4 di 6)

```
res = pthread_create(&lTID, NULL, lettore, NULL);
if (res != 0) printf("Errore nella creazione del primo thread\n");
res = pthread_create(&s1TID, NULL, scrittore1, NULL);
if (res != 0) {
    printf("Errore nella creazione del secondo thread\n");
    pthread_kill(s1TID, SIGKILL);exit(-1);
}
res = pthread_create(&s2TID, NULL, scrittore2, NULL);
if (res != 0) {
    printf("Errore nella creazione del terzo thread\n");
    pthread_kill(lTID, SIGKILL);pthread_kill(s1TID, SIGKILL);
    return -1;
}
/* Aspetto che i tre thread finiscano ... */
pthread_join(s1TID, NULL);pthread_join(s2TID, NULL);
pthread_join(lTID, NULL);
printf("E' finito l'esperimento ....\n");
return (0);
}
```

Esempio 6 (5 di 6)

```
void *scrittore1(void *in){
    int i, j, k;
    for (i=0; i<CICLI; i++) {
        for(k=0; k<LUN; k++) {
            pthread_mutex_lock(&shared.mutex); /* acquisisco la mutua escl. */
            while (shared.elementi == LUN) {
                shared.blockscri++; /* Segnalo che mi sto bloccando */
                pthread_cond_wait(&shared.scrittori, &shared.mutex);
                shared.blockscri--; /* Segnalo che mi sto sbloccando */
            }
            /* Aggiungo un carattere e aggiorni i vari campi */
            shared.scritta[shared.ultimo] = '-';
            shared.ultimo = (shared.ultimo+1)%LUN);
            shared.elementi++;
            if (shared.blocklet != 0)/*Controllo se devo sbloccare il lettore */
                pthread_cond_signal(&shared.lettore);
            pthread_mutex_unlock(&shared.mutex); /* Rilascio la mutua escl.*/
            for(j=0; j<DELAY_WRITER; j++); /* perdo tempo*/
        }
    }
    return NULL;
}
```

Esempio 6 (6 di 6)

```
void *lettore(void *in){
    int i, k, j; char local[LUN+1]; local[LUN] = 0;
    for (i=0; i<2*CICLI; i++) {
        for (k=0; k<LUN; k++) {
            pthread_mutex_lock(&shared.mutex); /*acquisisco la mutua esclusione */
            while (shared.elementi == 0) {
                shared.blocklet++; /* Segnalo che mi sto bloccando */
                pthread_cond_wait(&shared.lettore, &shared.mutex);
                shared.blocklet--; /* Segnalo che mi sono sbloccato */
            }
            /* Leggo un carattere e aggiorno i vari campi */
            local[k] = shared.scritta[shared.primo];
            shared.primo = (shared.primo+1)%LUN;
            shared.elementi--;
            if (shared.blockscri != 0) /* Controllo se devo sbloccare uno scrittore */
                pthread_cond_signal(&shared.scrittori);
            pthread_mutex_unlock(&shared.mutex); /* Rilascio la mutua esclusione */
            for(j=0; j<DELAY_READER; j++);
        }
        printf("Stringa = %s \n", local);
    }
    return NULL;
}
```

Esercizi



Esercizio 1

- `threads-3a-mutex.c`
 - analizzare l'output
 - modificare in modo da ottenere un funzionamento corretto
- `threads-3b-mutex.c`
 - soluzione dell'esercizio precedente (prossima volta)

pthread-3a-mutex.c

```
#include <pthread.h>
#include <stdio.h>

#define NUM_THREADS 40

int shared = 0;

void *thread_main(void* arg){
    int i,k;
    for (i=0; i<1000000; i++) {
        k = shared;
        k = k+1;
        shared = k;
    }
    printf("Hello world from thread %d (shared=%d)\n", (int)arg, shared);
    pthread_exit(arg);
}

int main (void) {
    int t, status;
    pthread_t children[NUM_THREADS];
    for (t=0; t<NUM_THREADS; t++){
        pthread_create(&children[t], NULL, thread_main, (void*)t);
    }
    for (t=0; t<NUM_THREADS; t++){
        pthread_join(children[t], (void**)&status);
    }
    return 0;
}
```

Esercizio 2

- `pthread-4a-barrier.c`
 - modificare in modo da ottenere la sincronizzazione desiderata
 - ⇒ tutti i thread si bloccano alla barriera aspettando l'arrivo di tutti gli altri
 - ⇒ tutti i thread proseguono l'esecuzione quando l'ultimo ha raggiunto la barriera
 - suggerimento: usare una variabile condition
- `pthread-4b-barrier.c`
 - soluzione dell'esercizio precedente (prossima volta)

pthread-4a-barrier.c (1 di 2)

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

#define NUM_THREADS 4

/* Some variables are needed here... */

int n_threads; /* number of worker threads */

/* Complete the function body */
void barrier()
{
}

void *parallel_elaboration(void * arg)
{
    int delay = rand()%6;
    int i = *(int*)arg;
    printf("[Thread %d] Waiting for %d secs...\n",i,delay);
    sleep(delay);
    printf("[Thread %d] ...elaboration finished, waiting for the other
threads...\n",i);
    barrier();
    printf("[Thread %d] ...ok!\n",i);
    pthread_exit(NULL);
}
```

pthread-4a-barrier.c (2 di 2)

```
int main(void)
{
    pthread_t tids[NUM_THREADS];
    int params[NUM_THREADS];
    int i, rc;
    n_threads = NUM_THREADS;
    /* Some initialization goes here... */
    srand ( time(NULL) );
    printf("[Main] Starting...\n");
    for (i=0; i<NUM_THREADS; i++) {
        printf("[Main] Creating thread %d..\n", i+1);
        params[i] = i+1;
        rc = pthread_create(&tids[i], NULL,
parallel_elaboration, (void*)&params[i]);
        if (rc){
pthread_create()\n");          perror("[Main] ERROR from
                                exit(-1);
                                }
        }
    printf("[Main] ...done!\n");
    pthread_exit(NULL);
}
```