Thread POSIX

- Introduzione ai thread POSIX
  - operazioni elementari sui thread
- Sincronizzazione
  - Semafori
    - ⇨ semafori di mutua esclusione
    - ⇨ semafori generali
    - ⇨ utilizzo ed esempi
  - Variabili condition
    - ⇨ generalità
    - ⇨ utilizzo ed esempi

Thread POSIX: aspetti preliminari

Thread

- Thread
  - insieme di istruzioni che viene eseguito in modo indipendente rispetto al main
- Stato di un thread
  - stack, registri, proprietà di scheduling, stato dei segnali, dati privati
- Vantaggi
  - concorrenza
  - limitato uso di risorse

Thread POSIX

- Standard ANSI/IEEE POSIX 1003.1 (1990)
  - thread POSIX (pthread)
- Utilizzo
  - includere l'header della libreria[1]
    `#include <pthread.h>`
  - compilare specificando la libreria[2]
    `gcc <opzioni> -pthread`

[1] per poter interpretare correttamente i messaggi di errore è necessario anche includere l'header `<errno.h>`

[2] per ulteriori informazioni sulla compilazione fare riferimento alla documentazione della piattaforma utilizzata
`man pthread` (o `man pthreads`)

Risorse (1 di 2)

- POSIX Threads Programming Tutorial
  - http://www.llnl.gov/computing/tutorials/pthreads/
- Libri (consultazione)
  - B. Lewis, D. Berg, "*Threads Primer*", Prentice Hall
  - D. Butenhof, "*Programming With POSIX Threads*", Addison Wesley
  - B. Nichols et al, "*Pthreads Programming*", O'Reilly

Risorse (2 di 2)

- Manpages
  - pacchetto `manpages-posix-dev` (**Debian**)
  - `man pthread.h`
  - `man <nomefunzione>`
- Manuale GNU libc
  - http://www.gnu.org/software/libc/manual/html_node/POSIX-Threads.html

Gestione dei thread

Tipi definiti nella libreria pthread

- All'interno di un programma un thread è rappresentato da un identificatore
  - tipo opaco *pthread_t*

```
pthread_t pthread_self( void )
int pthread_equal( pthread_t t1, pthread_t t2 )
```

- Attributi di un thread
  - tipo opaco *pthread_attr_t*

Creazione di un thread (1 di 2)

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

- *pthread_t* *thread
  - identificatore del thread (se creato con successo)
- *const pthread_attr_t* *attr
  - attributi del processo da creare
  - se NULL usa valori default
- *void* *(*start_routine)(void *)
  - funzione da eseguire alla creazione del thread

Creazione di un thread (2 di 2)

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

- *void* *arg
  - argomento da passare al thread
- *Valore di ritorno*
  - 0 in assenza di errore
  - diverso da zero altrimenti
    - ⇨ attributi errati
    - ⇨ mancanza di risorse

Terminazione di un thread

```
void pthread_exit( void *value_ptr )
```

- *void* *value_ptr
  - valore di ritorno del thread
    (può essere ottenuto attraverso la funzione join)

- Altre possibilità
  - ritorna dalla funzione chiamante
  - cancellazione da parte di un altro thread
  - terminazione dell'intero processo

Esempio: creazione e terminazione (1 di 2)

```
/* Include */
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS     5

/* Corpo del thread */
void *PrintHello(void *num) {
   printf("\n%d: Hello World!\n", num);
   pthread_exit(NULL);
}
```

Continua ⇨

Esempio: creazione e terminazione (2 di 2)

```
/* Programma */
int main (int argc, char *argv[])
{
   pthread_t threads[NUM_THREADS];
   int rc, t;
   for(t=0; t<NUM_THREADS; t++){
      printf("Creating thread %d\n", t);
      rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
      if (rc){
         printf("ERROR; return code from pthread_create() is %d\n", rc);
         exit(-1);
      }
   }
   pthread_exit(NULL);
}
```


Passaggio parametri (1 di 2)

- Per riferimento con un cast a *void**
- Esempio (errato)
  - il ciclo modifica il contenuto dell'indirizzo passato come parametro

```
int rc, t;

for(t=0; t<NUM_THREADS; t++) {
   printf("Creating thread %d\n", t);
   rc = pthread_create(&threads[t], NULL, PrintHello,
        (void *) &t);
   ...
}
```


Passaggio parametri (2 di 2)

- Esempio (corretto)
  - struttura dati univoca per ogni thread

```
int *taskids[NUM_THREADS];
for(t=0; t<NUM_THREADS; t++){
   taskids[t] = (int *) malloc(sizeof(int));
   *taskids[t] = t;
   printf("Creating thread %d\n", t);
   rc = pthread_create(&threads[t], NULL, PrintHello,
        (void *) taskids[t]);
   ...
}
```


Sincronizzazione

Join tra thread

- Forma elementare di sincronizzazione
  - il thread che effettua il join si blocca finché uno specifico thread non termina
  - il thread che effettua il join può ottenere lo stato del thread che termina
- Attributo `detachstate` di un thread
  - specifica se si può invocare o no la funzione join su un certo thread
  - un thread è joinable per default

Operazione di join

```
int pthread_join( pthread_t *thread, void **value )
```

- *pthread_t* *thread
  - identificatore del thread di cui attendere la terminazione
- *void* **value
  - valore restituito dal thread che termina
- Valore di ritorno
  - 0 in caso di successo
  - EINVAL se il thread da attendere non è joinable
  - ESRCH se non è stato trovato nessun thread corrispondente all'identificatore specificato

Impostazione attributo di join

```
int pthread_attr_init( pthread_attr_t *attr )
int pthread_attr_setdetachstate( pthread_attr_t *attr,
                                 int detachstate )
int pthread_attr_destroy( pthread_attr_t *attr )
```

- Esempio

```
/* Attributo */
pthread_attr_t attr;
/* Inizializzazione esplicita dello stato joinable */
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
...
pthread_attr_destroy(&attr);
```


Esempio: thread join

```
int main (int argc, char *argv[]) {
   pthread_t thread[NUM_THREADS];
   ...
   pthread_attr_destroy(&attr);
   for(t=0; t<NUM_THREADS; t++)
   {
      rc = pthread_join(thread[t], (void **)&status);
      if (rc) {
         printf("ERROR; return code from pthread_join() is %d\n", rc);
         exit(-1);
      }
      printf("Completed join with thread %d status= %d\n",t, status);
   }
   pthread_exit(NULL);
}
```

Sincronizzazione tra thread

- Attraverso variabili globali
  - condivise tra thread
  - meccanismi di protezione
- Compito del programmatore
  - corretto utilizzo delle funzioni di sincronizzazione
- Meccanismi forniti dalla libreria
  - semafori di mutua esclusione
  - semafori generali
  - variabili condition

Semafori di mutua esclusione

Semafori di mutua esclusione (mutex)

- Protezione delle sezione critiche
  - variabili condivise modificate da più thread
  - solo un thread alla volta può accedere ad una risorsa protetta da un mutex
  - il mutex è un semaforo binario
- Interfaccia
  - lock per bloccare una risorsa
  - unlock per liberare una risorsa

Uso dei mutex

- Esempio
  - creazione e inizializzazione di una variabile mutex
  - più thread tentano di accedere alla risorsa invocando l'operazione di lock
  - un solo thread riesce ad acquisire il mutex mentre gli altri si bloccano
  - il thread che ha acquisito il mutex manipola la risorsa
  - lo stesso thread la rilascia invocando la unlock
  - un altro thread acquisisce il mutex e così via

Mutex: tipo e inizializzazione statica

- Nella libreria pthread un mutex è una variabile di tipo *pthread_mutex_t*
- Inizializzazione
  - statica contestuale alla dichiarazione
  - dinamica attraverso la funzione `pthread_mutex_attr_init()`

```
/* Variabili globali */
pthread_mutex_t amutex = PTHREAD_MUTEX_INITIALIZER;
```


Mutex: inizializzazione dinamica

```
int pthread_mutex_init( pthread_mutex_t *mutex, const
                        pthread_mutexattr_t *mattr )
```

- *pthread_mutex_t* *mutex
  - puntatore al mutex da inizializzare
- *pthread_mutexattr_t* *mattr
  - attributi del mutex da creare
  - estensioni per sistemi real time
  - se `NULL` usa valori default
- Valore di ritorno
  - sempre il valore 0

Mutex: operazioni lock e trylock

- Due varianti
  - bloccante (standard)
  - non bloccante (utile per evitare deadlock)

```
int pthread_mutex_lock( pthread_mutex_t *mutex )
int pthread_mutex_trylock( pthread_mutex_t *mutex )
```

- *pthread_mutex_t* *mutex
  - puntatore al mutex da bloccare
- Valore di ritorno
  - 0 in caso di successo, diverso da 0 altrimenti
  - `trylock` restituisce `EBUSY` se il mutex è occupato

Mutex: unlock e distruzione

```
int pthread_mutex_unlock( pthread_mutex_t *mutex )
int pthread_mutex_destroy( pthread_mutex_t *mutex )
```

- *pthread_mutex_t* *mutex
  - puntatore al mutex da sbloccare/distruggere
- Valore di ritorno
  - 0 in caso di successo
  - `destroy` restituisce `EBUSY` se il mutex è occupato

Esempio: uso dei mutex (1 di 2)

```
#include <pthread.h>
int a=1; b=1;
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
void* thread1(void *arg) {
   pthread_mutex_lock(&m);
   printf("Primo thread (parametro: %d)\n", (*arg);
   a++; b++;
   pthread_mutex_unlock(&m);
}
void* thread2(void *arg) {
   pthread_mutex_lock(&m);
   printf("Secondo thread (parametro: %d)\n", *arg);
   b=b*2; a=a*2;
   pthread_mutex_unlock(&m);
}
```

Continua ⇨

Esempio: uso dei mutex (2 di 2)

```
main() {
   pthread_t threadid1, threadid2;
   int i = 1, j=2;
   pthread_create(&threadid1, NULL, thread1, (void *)&i);
   pthread_create(&threadid2, NULL, thread2, (void *)&j);
   pthread_join(threadid1, NULL);
   pthread_join(threadid2, NULL);
   printf("Valori finali: a=%d b=%d\n", a, b);
}
```


Esempio: inizializzazione dinamica (1 di 2)

```
#include <pthread.h>
int a=1; b=1;
pthread_mutex_t m;
void* thread1(void *arg) {
   pthread_mutex_lock(&m);
   printf("Primo thread (parametro: %d)\n", (*arg);
   a++; b++;
   pthread_mutex_unlock(&m);
}
void* thread2(void *arg) {
   pthread_mutex_lock(&m);
   printf("Secondo thread (parametro: %d)\n", *arg);
   b=b*2; a=a*2;
   pthread_mutex_unlock(&m);
}
```

Continua ⇨

Esempio: inizializzazione dinamica (2 di 2)

```
main() {
   pthread_t threadid1, threadid2;
   int i = 1, j=2;
   pthread_mutex_init(&m, NULL);
   pthread_create(&threadid1, NULL, thread1, (void *)&i);
   pthread_create(&threadid2, NULL, thread2, (void *)&j);
   pthread_join(threadid1, NULL);
   pthread_join(threadid2, NULL);
   printf("Valori finali: a=%d b=%d\n", a, b);
   pthread_mutex_destroy(&m);
}
```

Semafori classici

Semafori classici (generali) (1 di 2)

- Semafori il cui valore può essere impostato dal programmatore
  - utilizzati per casi più generali di sincronizzazione
  - esempio: produttore consumatore
- Interfaccia
  - operazione wait
  - operazione post (signal)

Semafori classici (generali) (2 di 2)

- Semafori classici e standard POSIX
  - non presenti nella prima versione dello standard
  - introdotti insieme come estensione real-time con lo standard IEEE POSIX 1003.1b (1993)
- Utilizzo
  - associati al tipo *sem_t*
  - includere l'header `#include <semaphore.h>`

Semafori classici: inizializzazione (1 di 2)

- Richiedono un'inizializzazione esplicita da parte del programmatore

```
int sem_init( sem_t *sem, int pshared,
              unsigned int value )
```

- *sem_t* *sem
  - puntatore al semaforo da inizializzare
- *int* pshared
  - se 1 il semaforo è condiviso tra processi
  - se 0 il semaforo è privato del processo
  - nell'attuale implementazione deve essere posto a 0

Semafori classici: inizializzazione (2 di 2)

```
int sem_init( sem_t *sem, int pshared,
              unsigned int value )
```

- *unsigned int* *value
  - valore da assegnare al semaforo
- Valore di ritorno
  - 0 in caso di successo, -1 altrimenti con la variabile errno settata a EINVAL se il semaforo supera il valore SEM_VALUE_MAX

Semafori classici: operazione wait

- Due varianti
  - bloccante (standard)
  - non bloccante (utile per evitare deadlock)

```
int sem_wait( sem_t *sem )
int sem_trywait( sem_t *sem )
```

- *pthread_mutex_t* *mutex
  - puntatore al semaforo da decrementare
- Valore di ritorno
  - sempre 0 per la wait semplice
  - trywait restituisce -1 se il semaforo ha valore 0
    - ⇨ setta la variabile errno a EAGAIN

Semafori classici: operazione post e distruzione

```
int sem_post( sem_t *sem )
int sem_destroy( sem_t *sem )
```

- *sem_t* *sem
  - puntatore al semaforo da incrementare/distruggere
- Valore di ritorno
  - 0 in caso di successo, -1 altrimenti con la variabile errno settata in base al tipo di errore
    - ⇨ EINVAL se il semaforo supera il valore SEM_VALUE_MAX dopo l'incremento (sem_post)
    - ⇨ EBUSY se almeno un thread è bloccato sul semaforo (sem_destroy)

Semafori classici: lettura del valore

```
int sem_getvalue( sem_t *sem, int *sval )
```

- *sem_t *sem*
  - puntatore del semaforo di cui leggere il valore
- *int* *sval
  - valore del semaforo
- Valore di ritorno
  - sempre 0

Esempio: lettori e scrittori (1 di 4)

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#define LUN 20
#define CICLI 1
#define DELAY 100000
struct {
  char scritta[LUN+1];
  /* Variabili per la gestione del buffer */
  int primo, ultimo;
  /* Variabili semaforiche */
  sem_t mutex, piene, vuote;
} shared;
void *scrittore1(void *); void *scrittore2(void *);
void *lettore(void *);
```

Continua ⇨

Esempio: lettori e scrittori (2 di 4)

```
int main(void) {
  pthread_t s1TID, s2TID, lTID;
  int res, i;
  shared.primo = shared.ultimo = 0;
  sem_init(&shared.mutex, 0, 1);
  sem_init(&shared.piene, 0, 0);
  sem_init(&shared.vuote, 0, LUN);
  pthread_create(&lTID, NULL, lettore, NULL);
  pthread_create(&s1TID, NULL, scrittore1, NULL);
  pthread_create(&s2TID, NULL, scrittore2, NULL);
  pthread_join(s1TID, NULL);
  pthread_join(s2TID, NULL);
  pthread_join(lTID, NULL);
}
```

Continua ⇨

Esempio: lettori e scrittori (3 di 4)

```
void *scrittore1(void *in) {
  int i, j, k;
  for (i=0; i<CICLI; i++) {
    for(k=0; k<LUN; k++) {
      sem_wait(&shared.vuote); /* Controllo che il buffer non sia pieno */
      sem_wait(&shared.mutex);        /* Acquisisco la mutua esclusione */
      shared.scritta[shared.ultimo] = '-';       /* Operazioni sui dati */
      shared.ultimo = (shared.ultimo+1)%(LUN);
      sem_post(&shared.mutex);                        /* Libero il mutex */
      sem_post(&shared.piene);    /* Segnalo l'aggiunta di un carattere */
      for(j=0; j<DELAY; j++);              /* ... perdo un po' di tempo */
    }
  }
  return NULL;
}
```

Continua ⇨

Esempio: lettori e scrittori (4 di 4)

```
void *lettore(void *in) {
  int i, j, k; char local[LUN+1]; local[LUN] = 0;      /* Buffer locale */
  for (i=0; i<2*CICLI; i++) {
    for(k=0; k<LUN; k++) {
      sem_wait(&shared.piene); /* Controllo che il buffer non sia vuoto */
      sem_wait(&shared.mutex);        /* Acquisisco la mutua esclusione */
      local[k] = shared.scritta[shared.primo];   /* Operazioni sui dati */
      shared.primo = (shared.primo+1)%(LUN);
      sem_post(&shared.mutex);                        /* Libero il mutex */
      sem_post(&shared.vuote);     /* Segnalo che ho letto un carattere */
      for(j=0; j<DELAY; j++);              /* ... perdo un po' di tempo */
    }
  }
  return NULL;
}
```

Variabili condition

Variabili condition

- Oggetti di sincronizzazione su cui un processo si può bloccare in attesa
  - associate ad una condizione logica arbitraria (*predicato*)
  - generalizzazione dei semafori
  - nuovo tipo `pthread_cond_t`
  - attributi variabili condizione di tipo `pthread_condattr_t`

Variabili condition: inizializzazione

```
int pthread_cond_init( pthread_cond_t *cond,
    pthread_condattr_t *cond_attr )
int pthread_cond_destroy( pthread_cond_t *cond )
int pthread_condattr_init( pthread_condattr_t *attr )
int pthread_condattr_destroy( pthread_condattr_t *attr)
```

- **pthread_cond_**
  - inizializzazione/distruzione variabili condition
- **pthread_condattr_**
  - inizializzazione/distruzione attributi condition
    - ⇨ condivisione della condizione tra più processi

Variabili condition: 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

Variabili condition: operazione wait

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

- *pthread_cond_t* *cond
  - puntatore all'oggetto condizione su cui bloccarsi
- *pthread_mutex_t* *mutex
  - puntatore all'oggetto mutex da sbloccare
- Valore di ritorno
  - sempre 0

Variabili condition: operazione signal

- Due varianti
  - standard (sblocca un solo thread bloccato)
  - broadcast (sblocca tutti i thread bloccati)

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

- *pthread_cond_t* *cond
  - puntatore al l'oggetto condizione
- Valore di ritorno
  - sempre 0

Variabili condition: valutazione

- 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)
    ptread_cond_wait(&cond, &mutex);
computation();
pthread_mutex_unlock(&mutex);
```


Esempio: 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);
    }
    pthread_mutex_unlock(&count_mutex);
    for (j=0; j<1000; j++)
      result = result + (double)random();
    }
  pthread_exit(NULL);
}
```

Continua ⇨

Esempio: 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);
}
```

Continua ⇨

Esempio: incremento contatore (3 di 3)

```
int main (int argc, char *argv[]) {
  int i, rc;
  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 termination */
  ...
  /* Clean up and exit */
  pthread_attr_destroy(&attr);
  pthread_mutex_destroy(&count_mutex);
  pthread_cond_destroy(&count_threshold_cv);
  pthread_exit(NULL);
}
```