Thread Posix

Agenda

- Introduzione
- Libreria thread
  - Creazione e terminazione di thread
  - Sincronizzazione tra thread
  - Variabili condition – cenni
- Esempi ed esercizi

Fonti

- “POSIX Thread Programming”
  http://www.llnl.gov/computing/tutorials/workshops/workshop/pthreads/MAIN.html
- “Pthreads Programming”
  B. Nichols et al. O'Reilly and Associates.
- “Threads Primer”
  B. Lewis and D. Berg. Prentice Hall
- “Programming With POSIX Threads”
  D. Butenhof. Addison Wesley

Introduzione: **Thread vs. processi**

- Processo
  - unità di possesso di risorse creato dal S.O.
- Possiede
  - PID, process group ID, UID, and GID
  - Ambiente
  - Text, Dati, Stack, Heap
  - Registri
  - Descrittori di file
  - Gestore dei segnali
  - Meccanismi di IPC (code di messaggi, pipe, semafori, memoria condivisa)

Introduzione: **Thread vs. processi**

- Thread
  - Unità di esecuzione
  - Esiste all'interno di un processo
- Possiede
  - Stack pointer
  - Registri
  - Proprietà di scheduling (algoritmo, priorità)
  - Set di segnali pendenti o bloccati
  - Dati specifici a una thread

Thread

- Thread multiple eseguono all'interno dello stesso spazio di indirizzamento
- Conseguenze
  - Modifiche effettuate da una thread ad una risorsa condivisa sono visibili da tutte le altre thread
  - Possibile lettura e scrittura sulla stessa locazione di memoria
  - E' necessaria esplicita sincronizzazione da parte del programmatore
  - Thread sono paritetiche (no gerarchia)

Perché thread?

- Motivo essenziale: prestazioni
- Costo rispetto alla creazione di un processo molto inferiore
- Costo di IPC molto inferiore
- Aumento del parallelismo nelle applicazioni

Quali thread?

- Numerose implementazioni
- Implementazione standard
  - POSIX thread (pthread)
    - Definite come API in C
    - Implementate tramite
    - header/include file **`pthread.h`**
    - **`libpthread.a`**
  - In Linux
    - **`gcc <file.c> -lpthread`**
    - **`<file.c>`** deve includere **`pthread.h`**

Programmi multithreaded

- Per sfruttare i vantaggi delle thread, un programma deve poter essere organizzato in una serie di task indipendenti eseguibili in modo concorrente
- Esempi
  - procedure che possono essere sovrapposte nel tempo o eseguite in un qualunque ordine
  - Procedure che si bloccano per attese potenzialmente lunghe
  - Procedure che devono rispondere ad eventi asincroni
  - Procedure che sono più/meno importanti di altre

Programmi multithreaded

- Modelli tipici di programmi multithreaded
  - Manager/worker
    - Thread manager gestisce gli input e assegna operazioni a altre thread worker
  - Pipeline
    - Un task viene spezzato in una serie di sottooperazioni (implementate da thread distinte) da eseguire in serie e in modo concorrente
  - Peer
    - Simile al modello manager/worker model
    - Dopo che la thread principale crea le altre, partecipa al lavoro

API Pthread

- Tre categorie di funzioni
  - Gestione thread
  - Sincronizzazione (mutex) tra thread
  - Comunicazione tra thread

Prefisso della routine	Utilizzo
pthread_	Subrutine generica per gestione thread
pthread_attr	Oggetto attributo
pthread_mutex	Mutex
pthread_mutexattr	Oggetto attributo per mutex
pthread_cond	Variabile condition
pthread_condattr	Oggetto attributo per variabile condition
pthread_key	Chiave specifica per la thread

Creazione di thread

`int pthread_create(thread, attr, start_routine, arg)`

- **`pthread_t *thread:`** thread ID (TID)
  - Necessario controllo sul valore corretto del valore
- **`pthread_attr_t *attr:`** per settare attributi della thread
  - **`NULL=`** valori di default
  - Per altre modalità vedi dopo
- **`(void*)(*start_routine)(void*)`**
  - la funzione che verrà eseguita dalla thread
- **`void *arg:`** argomento per la **`start_routine`**
  - Se necessari più parametri
    - Costruire una **`struct`** con i parametri desiderati
    - Passare un puntatore alla **`struct`** (cast a **`(void*)`**)
- Valore di ritorno: diverso da 0 → errore

Terminazione di thread

`int pthread_exit (void* status)`

- Termina una thread
- **`status`**: valore di terminazione
  - Tipicamente = NULL
- Non chiude eventuali file aperti

Modi alternativi per terminare una thread

- **`exit()`** sul processo
- Termine della **`start_routine`**

Terminazione di thread

- NOTE
  - Se il **`main()`** termina prima delle thread che ha creato, ed esce con **`pthread_exit()`**, le altre thread continuano l'esecuzione
  - In caso contrario (no **`pthread_exit()`**) le thread vengono terminate insieme al processo
  - **`pthread_exit()`** non chiude i file
    - File aperti dentro la thread rimarranno aperti dopo la terminazione delle thread

Esempio

```
/* hello.c
Creazione di 5 thread che stampano "Hello World!" */
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS     5

/* Routine per la thread */
void *PrintHello(void *threadID)
{
   printf("\n%d: Hello World!\n", threadID);
   pthread_exit(NULL);
}
```


Esempio (cont.)

```
int main(){
   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);
}
```

Esempio: parametri

```
/* hello_arg2.c
Il puntatore (void *) consente di passare più
argomenti in una struct */
/* inserire include, compreso pthread.h*/
#define NUM_THREADS 8
char *messages[NUM_THREADS];
/* struct per il passaggio di parametri alla
thread */
struct thread_data {
   int thread_id;
   int sum;
   char *message;
};
/* definizione dei parametri per le thread */
struct thread_data thread_buf[NUM_THREADS];
```


Esempio: parametri (cont.)

```
/ start routine per le thread */
void *PrintHello(void *thread_arg){
   struct thread_data *my_data;
   int taskid, sum;
   char* hello_msg;

   sleep(1);
   my_data = (struct thread_data *) thread_arg;
   taskid = my_data->thread_id;
   sum = my_data->sum;
   hello_msg = my_data->message;
   printf("Thread %d: %s sum=%d\n", taskid,
           hello_msg, sum);
   pthread_exit(NULL);
}
```

Esempio: parametri (cont.)

```
int main(){
   pthread_t threads[NUM_THREADS];
   int rc, t, sum=0;
   messages[0] = "English: Hello world!";
   …
   messages[7] = "Latin: Orbis, te saluto!";
   for(t=0; t < NUM_THREADS, t++){
     sum = sum + t;
     thread_buf[t].thread_id = t;
     thread_buf[t].sum = sum;
     thread_buf[t].message = messages[t];
     printf("Creating thread %d\n", t);
     rc=pthread_create(&threads[t],NULL,
            printHello, (void*) &thread_buf[t]);
```


Esempio: parametri (cont.)

```
  if (rc){
      printf("ERROR:return code is %d", rc);
      exit(-1);
  }
  pthread_exit(NULL);
}
```

Identificazione di thread

`int pthread_self ()`

- ritorna alla thread chiamante l'ID che il sistema assegna

`int pthread_equal(thread1,thread2)`

- Confronta due thread ID, se diversi ritorna 0, altrimenti un valore diverso da 0

Sincronizzazione tra thread

- Il "join" di thread è uno dei modi per realizzare la sincronizzazione tra thread

`int pthread_join(tid,status)`

- Blocca la thread chiamante fino a che la thread specificata da **`tid`** termina
- L'argomento **`status`** contiene lo stato di terminazione della thread **`tid`** indicato tramite il parametro **`status`** di **`pthread_exit()`**

Sincronizzazione tra thread

- Quando una thread viene creata, il campo **`attr`** definisce se
  - se ne può fare il join (joinable)
  - non se ne può fare il join (detached)

Join/detach di thread: Funzioni

- **`int pthread_attr_init(attr)`**
  - Argomento **`attr`** di tipo **`pthread_attr_t*`**
  - Per inizializzare l'oggetto attributo **`attr`**
- **`int pthread_attr_setdetachstate(attr,detachstate)`**
  - Argomento **`detachstate`** di tipo **`int`**
  - Per settare l'oggetto attributo **`attr`** al valore **`detachstate`**
    - PTHREAD_CREATE_JOINABLE oppure
    - PTHREAD_CREATE_DETACHED
- **`int pthread_detach(threadid)`**
  - Argomento **`threadid`** di tipo **`pthread_t`**
  - Per mettere la thread **`threadid`** nello stato detached

Join/detach di thread: Funzioni

- **`int pthread_attr_getdetachstate(attr,detachstate)`**
  - Memorizza il valore dell'attributo **`attr`** nell'argomento **`detachstate`**
- **`int pthread_attr_destroy(attr)`**
  - Libera la memoria associata all'attributo **`attr`**

Join/detach di thread

1. Dichiarare una variabile attributo di tipo **`pthread_attr_t`**
2. Inizializzare la variabile attributo con la funzione **`pthread_attr_init()`**
3. Settare il valore desiderato della variabile attributo tramite **`pthread_attr_setdetachstate()`**
4. Creare la thread con la variabile attributo
5. Cancellare le risorse usate dalla variabile attributo con la funzione **`pthread_attr_destroy()`**

- La funzione **`pthread_detach()`** permette di fare il detach esplicito di una thread anche se era stata creata come joinable

Esempio

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS   3

void *BusyWork(void *null) {
   int i;
   double result=0.0;
   for (i=0; i<100000; i++) {
     result = result/2 + (double)random();
   }
   printf("Thread result = %d\n",result);
   pthread_exit((void *) 0);
}
```


Esempio (cont.)

```
int main()
{
   pthread_t thread[NUM_THREADS];
   pthread_attr_t attr;                          (1)
   int rc, t, status;

   /* Initialize and set
    thread detached attribute */
   pthread_attr_init(&attr);                     (2)
   pthread_attr_setdetachstate(&attr,
                PTHREAD_CREATE_JOINABLE);        (3)
```

Esempio (cont.)

```
/* creation of threads, they are joinable */
for(t=0; t < NUM_THREADS; t++) {
    printf("Creating thread %d\n", t);
    rc = pthread_create(&thread[t], &attr,      (4)
                        BusyWork, NULL);
    if (rc != 0) {
      printf("ERROR; return code from
              pthread_create() is %d\n", rc);
      exit(-1);
    }
}
```


Esempio (cont.)

```
/*Free attribute and wait for the other threads*/
pthread_attr_destroy(&attr);          (5)
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);}
```

Mutex

- Meccanismo base per l'accesso protetto ad una risorsa condivisa (sezione critica)
- Una sola thread alla volta può fare il *lock* di un mutex
- Le altre thread in competizione sono messe in attesa
- Variabili mutex
  - Tipo **`pthread_mutex_t`**
  - Devono essere inizializzate prima dell'uso
    - Staticamente all'atto della dichiarazione
      - Es: **`pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;`**
    - Dinamicamente con opportuna funzione
      - Es: **`pthread_mutex_init (mutex,attr)`**

Mutex – Funzioni

- **`int pthread_mutex_init (mutex,attr)`**
  - Per inizializzare una variabile **`mutex`** come specificato dall'attributo **`attr`**
- **`int pthread_mutex_destroy(mutex)`**
  - Per distruggere una variabile **`mutex`**
- **`int pthread_mutexattr_init (attr)`**
  - Per inizializzare un attributo per una variabile mutex
- **`int pthread_mutexattr_destroy (attr)`**
  - Per distruggere un attributo usato per una variabile mutex

Mutex – Funzioni

- Il parametro **`mutex`** rappresenta *l'indirizzo* del mutex che si inizializza/distrugge
- Il parametro **`attr`** viene usato per definire proprietà di un mutex
  - Default = NULL
  - Attributi definiti per usi avanzati
- Tipica invocazione

```
pthread_mutex_t s;
...
pthread_mutex_init(&s, NULL);
```


Mutex – Attributi per usi avanzati

- Gli attributi determinano il tipo di mutex
  - Fast mutex (default)
  - Recursive mutex
  - Error checking mutex
- Il tipo determina il comportamento del mutex rispetto a operazioni di lock da parte di una thread su un mutex già posseduto dalla thread stessa
  - Fast: thread viene bloccata per sempre
  - Recursive: thread incrementa numero di lock e prosegue
    - Per sbloccare il mutex bisogna chiamare la unlock un numero di volte pari al numero di lock effettuati
  - Error checking: thread ritorna errore e prosegue

Mutex – Attributi per usi avanzati

- E' possibile definire mutex dei 3 tipi tramite costanti di inizializzazione
  - Recursive
    ```
    pthread_mutex_t m =
    PTHREAD_RECURSIVE_MUTEX_INITIALIZER_NP;
    ```
  - Error checking
    ```
    pthread_mutex_t m =
    PTHREAD_ERRORCHECK_MUTEX_INITIALIZER_NP;
    ```
  - Fast
    ```
    pthread_mutex_t m =
    PTHREAD_MUTEX_INITIALIZER;
    ```


Lock/unlock di mutex

- **`int pthread_mutex_lock (mutex)`**
  - Acquisisce un lock attraverso la variabile **`mutex`**
  - Se il mutex è già posseduto da un'altra thread, la chiamata si blocca fino a che il lock è disponibile (fast mutex)
- **`int pthread_mutex_trylock (mutex)`**
  - Tenta di acquisire un mutex
  - Se il mutex è posseduto da un'altra thread, ritorna immediatamente con un codice di errore "busy"
    - In Linux: codice errore **`EBUSY`**
- **`int pthread_mutex_unlock (mutex)`**
  - Sblocca il lock posseduto dalla thread chiamante (fast mutex)

Esempio

```
/* mutex1.c
   utilizzo di mutex */
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
/* This is the lock for thread synchronization */
pthread_mutex_t my_sync;
/* starting routine */
void* compute_thread(void* dummy) {
/* Lock the mutex when its our turn */
   pthread_mutex_lock(&my_sync);
   sleep(5);
   printf("%s", dummy);
   fflush(stdout);
   pthread_mutex_unlock(&my_sync);
   return;
}
```


Esempio (cont.)

```
int main() {
   pthread_t tid;
   pthread_attr_t attr;
   char hello[] = "Hello, ";
   /* Initialize the thread attributes */
   pthread_attr_init(&attr);
   /* Initialize the mutex (default attributes) */
   pthread_mutex_init(&my_sync,NULL);
   /* Create another thread. ID is returned in &tid */
   pthread_create(&tid, &attr, compute_thread, hello);
   sleep(1);   /* Let the thread get started */
   /* Lock the mutex when it's our turn to do work */
   pthread_mutex_lock(&my_sync); /*change it with trylock!*/
   sleep(2);
   printf("thread");
   printf("\n");
   pthread_mutex_unlock(&my_sync);
   exit(0);
}
```

Esempio (cont.)

- Nell'esempio precedente provare a sostituire nel main
  `pthread_mutex_lock(&my_sync);`
  con
  `pthread_mutex_trylock(&my_sync);`

- Cosa succede?

Condition Variable – cenni

- Meccanismo addizionale di sincronizzazione
- Permettono di sincronizzare thread in base ai valori dei dati senza busy waiting
- Senza condition variable, le thread devono testare in busy waiting (eventualmente in una sezione critica) il valore della condizione desiderata
- Le variabili condition sono sempre associate all'uso di un mutex