Corso di Laboratorio di Sistemi Operativi
A.A. 2016–2017

Lezione 17

Ivan Scagnetto
ivan.scagnetto@uniud.it
Nicola Gigante
gigante.nicola@spes.uniud.it

Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Università degli Studi di Udine

A.A. 2016–2017 - Primo Semestre - 06/12/2016

Multithreading

Parte 1

I thread

Nei sistemi operativi moderni, ogni processo può contenere uno o più flussi di esecuzione, chiamati **thread**.

- Ogni thread viene eseguito in modo indipendente ma condivide la maggior parte delle risorse del processo con gli altri thread:
  - Codice
  - Spazio degli indirizzi
  - File aperti, handler dei segnali, ...
- I singoli thread mantengono invece separate le risorse legate al proprio flusso di esecuzione:
  - Program counter
  - Registri della CPU
  - Stack

I thread

Usi e motivazioni

I thread sono la primitiva di base per ottenere **concorrenza** e **parallelismo** all'interno di un singolo processo.

- Oggigiorno il multithreading è un paradigma fondamentale data l'evoluzione degli attuali sistemi multicore.
- Avendo un address space comune, la creazione dei thread ed il cambio di contesto sono notevolmente meno costosi rispetto ai corrispondenti meccanismi che riguardano i processi.
- Inoltre, la comunicazione tra thread è molto efficiente, senza la necessità di meccanismi di comunicazione inter-thread ad-hoc.
- Anche nei sistemi con un'unica CPU si hanno dei benefici. Infatti, è possibile sfruttare i tempi di latenza delle operazioni di I/O di un thread per eseguirne nel frattempo un altro.

I thread

Problematiche

La programmazione multithread presenta delle problematiche che la rendono particolarmente difficoltosa:

- La concorrenza spesso richiede che i thread accedano a dati **condivisi**, ma ciò deve avvenire evitando sempre **race condition**, ovvero scritture contemporanee sulle stesse locazioni di memoria.
- L'interazione tra thread va quindi sempre sincronizzata tramite opportune primitive, che vanno usate in modo corretto.
- Il **nondeterminismo** dato dall'imprevedibilità dello scheduler rende molto difficile l'individuazione e la risoluzione di bug dovuti all'errata sincronizzazione di thread. Occorre quindi molta disciplina nella scrittura del codice.

L'interfaccia POSIX per i thread

La libreria `pthread` (POSIX thread), è l'interfaccia standard nei sistemi UNIX per la creazione e la manipolazione di thread.

Per utilizzare le funzioni `pthread` è necessario linkare l'apposita libreria:

```
$ clang -lpthread programma.c -o programma
```

Creare un thread

La funzione principale per creare un thread è `pthread_create()`:

```
#include <pthread.h>
int pthread_create(pthread_t *thread, pthread_attr_t *attr,
                   void * (*start_routine)(void *), void *arg);
```

Significato degli argomenti, in ordine:

1. Un puntatore ad una variabile di tipo `pthread_t` che funge da **handler** per rappresentare il thread creato.
2. Un puntatore ad una struttura contenente opzioni e configurazioni aggiuntive per il comportamento del thread.
3. Un puntatore ad una funzione che verrà eseguita dal nuovo thread. La funzione deve accettare e restituire un `void*`.
4. Un puntatore a `void` che verrà passato come argomento alla funzione lanciata dal nuovo thread.

Creare un thread

Esempio

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

void *print_msg(void *ptr);

int main()
{
  char msg1[] = "Thread 1";
  char msg2[] = "Thread 2";

  pthread_t thread1, thread2;

  pthread_create(&thread1, NULL, print_msg, (void *)msg1);
  pthread_create(&thread2 ,NULL, print_msg, (void *)msg2);

  pthread_join(thread1, NULL);
  pthread_join(thread2, NULL);
  return 0;
}

void *print_msg(void *ptr)
{
  char *arg = (char *) ptr;

  while(1)
    printf("%s\n", arg);

  return NULL;
}
```

Attributi di un thread

Il parametro di tipo `struct pthread_attr_t` di `pthread_create()` è detto **thread attribute object**, e specifica diversi parametri che influenzano la vita del thread.

- L'oggetto va inizializzato e distrutto con le apposite funzioni:

```
struct pthread_attr_t attr;
pthread_attr_init(&attr); // inizializza a valori di default
// impostare i parametri in attr con le apposite funzioni
pthread_create(&thread, &attr, func, arg);
pthread_attr_destroy(&attr); // dismettere l'attributes object
```

- Il valore di ogni singolo attributo X si imposta con l'apposita funzione `pthread_attr_setX()`.

Attributi di un thread

Tra gli attributi che è possibile impostare troviamo:

Attributo	`pthread_attr_setX`	Significato
Scheduling policy	`sched_policy`	Le politiche di scheduling relative al thread. Può essere lasciato il valore di default `SCHED_OTHER` oppure richiesto uno scheduling **round robin** o **FIFO**.
Sched. Inheritance	`inheritsched`	Specifica se il thread eredita le policy di scheduling del thread padre.
Contention Scope	`scope`	Specifica se il thread compete, per l'uso della CPU, con tutti gli altri thread/processi del sistema o solo quelli del processo (`PTHREAD_SCOPE_SYSTEM` oppure `PTHREAD_SCOPE_PROCESS`).
Scheduling prioriy	`schedparam`	Specifica la priorità associata al thread.

Terminazione di un thread

L'esecuzione di un thread termina quando la sua funzione principale ritorna, o quando viene chiamata la funzione `pthread_exit()`:

```
void pthread_exit(void *retval);
```

La funzione `pthread_join()` permette ad un thread di aspettare la fine di un altro di cui abbia l'handler:

```
int pthread_join(pthread_t th, void **value_ptr);
```

La funzione blocca il thread corrente in attesa della terminazione del thread identificato da `th`. Il valore di ritorno del thread viene scritto in `*value_ptr`.

Parallelismo tramite threads

Esempio `parallel_max.c`

Nel file di esempio in allegato trovate un esempio d'uso del multithreading per effettuare una computazione in **parallelo** su più threads.

- Il programma `parallel_max.c` legge il contenuto di un file in modo binario, lo interpreta come un array di numeri interi, e stampa il numero più alto tra quelli letti.
- Il calcolo del numero più alto avviene in parallelo su due thread the operano su due diverse metà dell'array.

Nota: Non è necessario sincronizzare l'accesso dei diversi thread alla memoria condivisa perchè tali accessi avvengono sempre esclusivamente in lettura (e comunque su parti separate dell'array).

Esercizi

Esercizi

Per mercoledì 14 dicembre

1. Modificare l'esempio `parallel_max.c` in modo da permettere all'utente di specificare il numero di threads in cui si vuole suddividere il lavoro.
2. Modificare l'esempio `upperserver.c` della lezione 16 per fare in modo che le diverse connessioni in entrata vengano gestite da diversi thread dello stesso processo invece che da più processi.