Esercitazione [1]

Processi e Thread

Leonardo Aniello - aniello@dis.uniroma1.it

Sistemi di Calcolo - Secondo modulo (SC2)
Programmazione dei Sistemi di Calcolo Multi-Nodo

Corso di Laurea in Ingegneria Informatica e Automatica

A.A. 2014-2015

Sommario

- Processi e thread: breve riepilogo
- Obiettivi dell’esercitazione
- Misurazione performance
- Tempi di lancio/terminazione
- Accesso concorrente a variabili condivise

Processi vs Thread

- Performance
  Lanciare/terminare un thread è più veloce rispetto a lanciare/terminare un processo
- Memoria
  La creazione di un processo (tramite fork) comporta la copia dell'intera memoria del padre, mentre i thread la condividono
- Comunicazione
  I thread di uno stesso processo possono usare la sua memoria per comunicare tra loro, mentre la comunicazione tra processi richiede meccanismi aggiuntivi

Processi in C - fork e wait

```
pid_t pid;
pid = fork();
if (pid == -1) {
      // gestione errore
      exit(EXIT_FAILURE);
} else if (pid == 0) {
      // codice processo figlio
      exit(EXIT_SUCCESS);
} else {
      // codice processo padre
      wait(0); // attende il termine del figlio
      exit(EXIT_SUCCESS);
}
```

Thread in C - pthread create e join

```
void* thread_stuff(void *arg) {
      // codice thread
      return NULL;
}
…
…
pthread_t thread; int c;
c = pthread_create(&thread,NULL,thread_stuff,NULL);
if (c != 0) {
      // gestione errore
      exit(EXIT_FAILURE);
}
pthread_join(thread, NULL); // attesa termine thread
```

Obiettivi Esercitazione [1]

1. Capire le differenze tra processi e thread in termini di ritardi di lancio/terminazione, e da cosa dipendono
2. Imparare le basi della programmazione multi-threading
   a. Impostare un'applicazione multi-threading
   b. Implementare meccanismi di comunicazione tra thread basati su variabili condivise
   c. Comprendere le problematiche legate all'accesso concorrente

Misurazione performance

- Tempo di esecuzione di una porzione di codice
- In fase di compilazione
  - Includere il sorgente performance.c
  - Linkare le librerie run time (-lrt) e math (-lm)

```
#include "performance.h"
…
timer t; begin(&t);
// porzione di codice di cui cronometrare l’esecuzione
end(&t); unsigned long int time;
time=get_seconds(&t); time=get_milliseconds(&t);
time=get_microseconds(&t); time=get_nanoseconds(&t);
```

Tempi di lancio/terminazione

- Misurazione della reattività (tempo richiesto) di processi e thread nelle fasi di lancio e terminazione
- Sorgente: `reactivity.c`
- Compilazione:
  ```
  gcc -o reactivity reactivity.c performance.c -lpthread -lrt -lm
  ```
- Vengono eseguiti N test per calcolare le reattività medie
  - N argomento
  - Viene calcolato lo speedup: $\frac{reattività\ processi}{reattività\ thread}$

Tempi di lancio/terminazione - Esercizio

- Nella slide 3 vengono elencate alcune differenze tra processi e thread
- È possibile sfruttarne una per rendere ancor meno reattivo il lancio di un nuovo processo ed aumentare di conseguenza lo speedup rispetto ai thread
- Esercizio
  - Individuare la differenza da sfruttare
  - Capire come sfruttarla per rallentare l'avvio dei processi
  - Modificare reactivity.c di conseguenza
  - Verificare l'aumento dello speedup

Accesso concorrente a variabili condivise

- Cosa succede quando più thread accedono in scrittura ad una variabile condivisa in concorrenza?
- Sorgente: `concurrent_threads.c`
- Compilazione:
```
gcc -o concurrent_threads
concurrent_threads.c -lpthread
```
- N thread in parallelo che aggiungono M volte un valore V ad una variabile condivisa (inizializzata a 0)
- La variabile condivisa alla fine dovrebbe valere N*M*V, succede sempre?

Accesso concorrente a variabili condivise - Esercizio

- Nelle prossime esercitazioni verranno presentati dei meccanismi di sincronizzazione per risolvere questi problemi
- È comunque possibile implementare una soluzione che non usa meccanismi di sincronizzazione e che mantiene la seguente semantica:
  - N thread effettuano in parallelo M incrementi di valore V
  - Al termine, il main thread verifica che tali incrementi equivalgano complessivamente a N*M*V
  - Suggerimento: lavorare sulle strutture dati per evitare accessi concorrenti in scrittura