Esercitazione [09]

Server multi-process/thread

Riccardo Lazzeretti - lazzeretti@diag.uniroma1.it

Sistemi di Calcolo 2
Programmazione dei Sistemi di Calcolo Multi-Nodo

Corso di Laurea in Ingegneria Informatica e Automatica

Sommario

- Correzione esercitazione precedente
- Server multi-process
- Esercizio: completare server multi-process
- Server multi-thread
- Esercizio: completare server multi-thread

Parallelismo lato server

- Nelle scorse esercitazioni abbiamo visto server «seriali»:
  - Viene servita una connessione alla volta
  - Connessioni che arrivano nel mentre vengono messe in coda…
  - …e verranno processate sequenzialmente al termine della connessione attualmente servita
  - Questo comporta dei tempi di attesa crescenti all'aumentare del numero di connessioni in coda!
  - ➢ La soluzione consiste nel disaccoppiare l'accettazione delle connessioni dalla loro elaborazione
  - ➢ Una volta accettata, una connessione viene elaborata in un processo o thread dedicato, così il server può subito rimettersi in attesa di altre connessioni da accettare

Server multi-process

- Per ogni connessione accettata, viene lanciato un nuovo processo figlio tramite `fork()`
  - Il figlio deve chiudere il descrittore della socket usata dal server per accettare le connessioni
  - Analogamente, il padre deve chiudere il descrittore della socket relativa alla connessione appena accettata
  - Una volta completata l'elaborazione della connessione, il processo figlio esce
- Elevato overhead legato alla creazione di nuovo processo per ogni connessione
- Complessa gestione di eventuali strutture dati condivise (tramite file, pipe, memoria condivisa oppure anche socket)

Server multi-process

```
while (1) {
        int client = accept(server, ……);
        <gestione errori>

        pid_t pid = fork();
        if (pid == -1) {
                <gestione errori>
        } else if (pid == 0) {
                close(server);
                <elaborazione connessione client>
                _exit(0);
        } else {
                close(client);
        }
}
```

Esercizio: EchoServer multi-process

- Completare il codice dell'EchoServer in modalità multi-process
- Sorgenti
  - `Makefile`
  - Client: `client.c`
  - Server: `server.c`
    - compilazione: `-DSERVER_MPROC` vs `-DSERVER_SERIAL`
- Suggerimento: seguire i blocchi di commenti inseriti nel codice
- Altro suggerimento:
  Per monitorare a runtime il numero di istanze di processi attivi in un certo momento, lanciare da terminale il comando:

```
ps -e -O ppid | head -1; ps -e -O ppid | grep multiprocess
```

Server multi-thread

- Per ogni connessione accettata, viene lanciato un nuovo thread tramite `pthread_create()`
  - Oltre ai parametri application-specific, il nuovo thread avrà bisogno del descrittore della socket relativa alla connessione appena accettata
  - A differenza del server multi-process, non è necessario chiudere alcun descrittore di socket (perché?)
  - Una volta completata l'elaborazione della connessione, il thread termina
  - Il main thread può voler fare detach dei thread creati
- Minore overhead rispetto al server multi-process
- Gestione più semplice di eventuali strutture dati condivise
- Un crash in un thread causa un crash in tutto il processo!

Server multi-thread

```
while (1) {
      int client = accept(server, ……);
      <gestione errori>

      pthread_t t;
      t_args = ……
      <includere client in t_args>
      pthread_create(&t,NULL,handler,(void*)t_args);
      <gestione errori>
      pthread_detach(t);
}
```

Esercizio proposto: EchoServer multi-thread

- Completare il codice dell'EchoServer in modalità multi-thread
- Sorgenti
  - `Makefile`
  - Client: `client.c`
  - Server: `server.c`
    - compilazione: `-DSERVER_MTHREAD` vs `-DSERVER_SERIAL`
- Suggerimento: seguire i blocchi di commenti inseriti nel codice
- Altro suggerimento:
  Per monitorare a runtime il numero di istanze di processi/thread attivi in un certo momento, lanciare da terminale il comando:

```
ps -e -T | head -1 ; ps -e -T | grep multithread
```