L'API socket ed i daemon

Massimo Bernaschi

Istituto per le Applicazioni del Calcolo 'Mauro Picone'

Consiglio Nazionale delle Ricerche

Viale del Policlinico, 137 - 00161 Rome - Italy

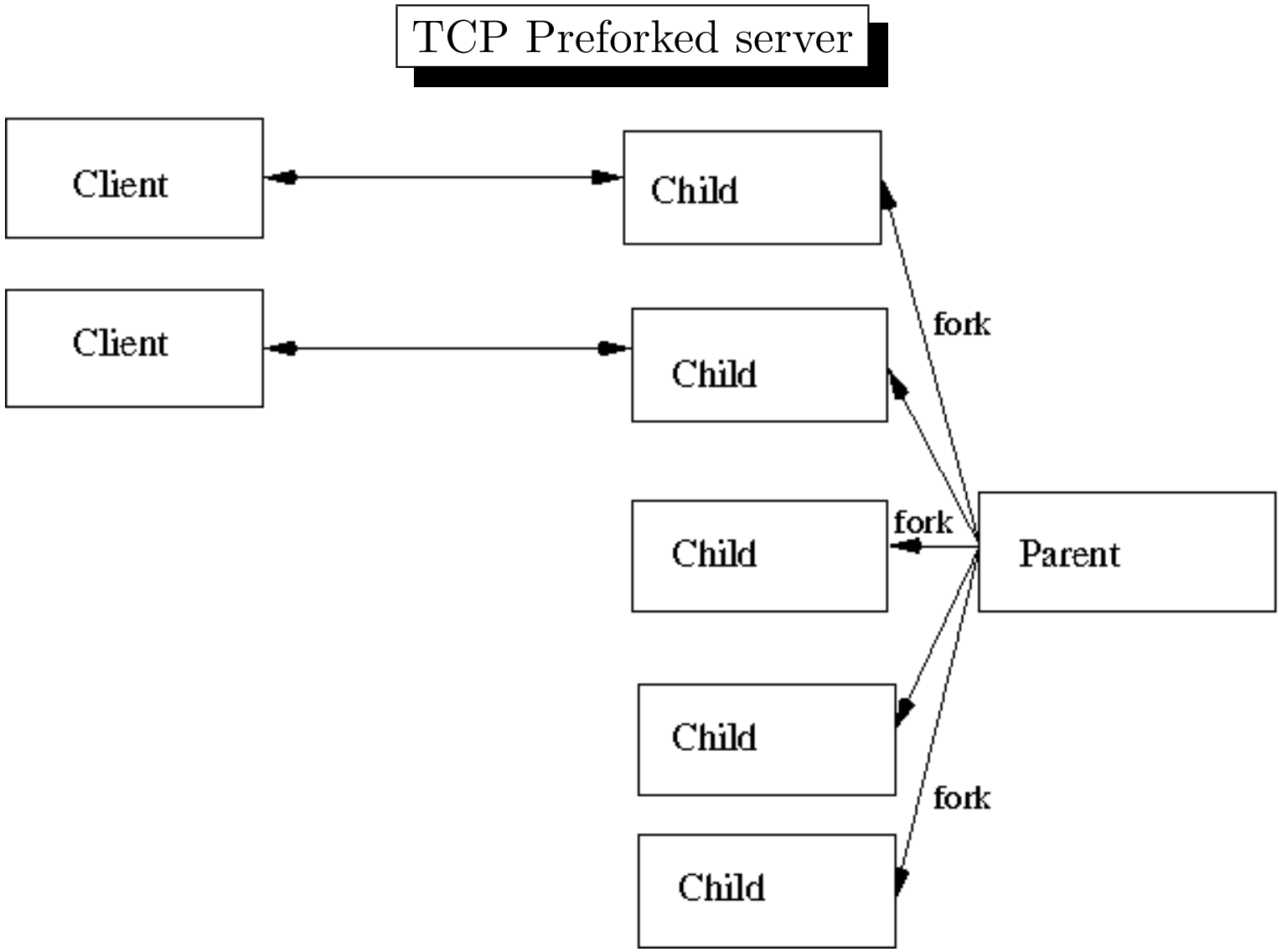
http://www.iac.cnr.it/

e-mail: massimo@iac.rm.cnr.it

Alternative nel disegno di applicazioni client-server

- Il più semplice server *concorrente* invoca `fork` per far partire un processo figlio per ogni client.
- Una prima alternativa è un singolo server TCP che utilizza la `select` per gestire un generico numero di client.
- È possibile modificare il server concorrente per creare un thread per client invece di un processo per client.
  - la `fork` è molto onerosa. Sebbene la maggior parte delle implementazioni attuali usi il meccanismo di *copy-on-write* per le pagine di memoria, tutti i descrittori vengono duplicati nel processo figlio insieme a molte altre informazioni.
  - Una qualche forma di IPC è richiesta per passare informazioni tra il processo padre ed il figlio *dopo* la fork.

– I *threads* possono attenuare l'impatto di questo tipo di problemi. La creazione di thread è da 10 a 100 volte più semplice della creazione di processi.


TCP Preforked server
Client
Client
Child
Child
Child
Child
Child
fork
fork
fork
Parent

TCP Preforked server

Diversi possibili alternative:

- Nessuna forma di locking attorno all'`accept`.
  - Soffre dell'effetto *thundering herd* perché tutti gli N processi figli sono risvegliati anche se solo uno ottiene la connessione.
  - Quando i processi figli sono bloccati nella chiamata nell'`accept`, l'algoritmo di scheduling del kernel distribuisce le connessioni uniformemente tra tutti i processi figli.
  - Quando più processi sono bloccati sullo stesso descrittore, è consigliabile bloccarli in una funzione come l'`accept`, invece di bloccarli nella `select` che, a causa del tipo di struttura dati utilizzata, richiede di risvegliare **tutti** i processi in attesa su un certo descrittore.

- File locking attorno all'`accept`.
  - Se il Sistema Operativo non permette di avere processi multipli che chiamano l'`accept` sullo stesso descrittore in stato *listening*, non appena il client inizia la connessione al server, una chiamata all'`accept` in uno dei processi figli ritorna `EPROTO`.
    (era quello che succedeva nelle vecchie versioni di Solaris).
  - Una possibile soluzione è che l'applicazione utilizzi un meccanismo di *lock* attorno all'`accept`, in modo che solo un processo per volta sia bloccato nell'`accept`.
  - Per il locking si può utilizzare il meccanismo di file locking Posix.

- Thread locking attorno all'`accept`.
  - Come già detto, si può utilizzare il thread locking non solo tra threads all'interno di un dato processo, ma anche per il locking tra processi distinti.
  - La variabile mutex deve, ovviamente, essere in una zona di memoria condivisa tra tutti i thread.
  - Alla libreria thread sotto Unix/Linux deve essere specificato che il mutex è condiviso tra processi distinti (deve supportare l'attributo `PTHREAD_PROCESS_SHARED`).

  Un possibile meccanismo per condividere la memoria è l'uso del file mapping (funzione `mmap`) del device `/dev/zero`.

  Con Windows si può accedere il Mutex tra processi distinti con l'apposita primitiva.

- Passaggio del descrittore.
  - Un'alternativa è avere solo il processo padre che chiama l'`accept` e quindi passa il socket connesso al processo figlio.
  - Questa tecnica richiede una qualche forma di passaggio di descrittori tra processi distinti.
  - Il processo padre tiene conto di quali processi figli sono già impegnati (la selezione è esplicita).

TCP server prethreaded

- `accept` per thread con *mutex locking*.
  - Ovviamente tra thread non c'è ragione di usare un meccanismo basato su file locking per controllare l'`accept`.
  - Sui kernel derivati da BSD non è necessario effettuare locking attorno all'`accept`.
  - Attenzione! Questa soluzione può essere più lenta di quella basata su mutex locking.

- `accept` nel thread principale
  - Ovviamente non è necessario passare un descrittore da un thread ad un altro dato che i thread condividono tutti i descrittori.
  - Il thread "ricevente" ha solo bisogno di sapere quale è il descrittore che deve utilizzare.
  - Attenzione! Questa soluzione, apparentemente semplice, può essere più lenta di quella in cui ogni thread invoca l'`accept`.

Riassunto dell'alternative

1. server iterativo;
2. server concorrente, una `fork` per client;
3. prefork in cui ogni figlio invoca l'`accept`;
4. prefork con file locking per proteggere l'`accept`;
5. prefork con mutex per proteggere `accept`;
6. prefork con passaggio del descrittore del socket al figlio;
7. server concorrente, crea un thread per client;
8. prethreaded con mutex per proteggere l'`accept`;
9. prethreaded con thread principale che invoca l'`accept`.

Riassunto

- Creare un insieme di processi figli o un insieme di thread riduce l'overhead di processamento di una richiesta.
- Il numero di processi figli (o thread) dovrebbe essere gestito dinamicamente.
- Alcune implementazioni permettono a più processi figli o thread di invocare `accept` mentre in altre implementazioni è richiesto un "lock" per l'invocazione dell'`accept`.
- In genere è più semplice avere tutti i processi figli o i thread che invocano `accept` ed è più veloce che avere un singolo thread che invoca `accept` e quindi passa il descrittore.
- È preferibile avere tutti i processi figli o i thread bloccati in una chiamata all'`accept` piuttosto che bloccati in una `select`.

Come *daemonizare* un processo

- `fork:` il processo padre termina ed il figlio prosegue.
- `setsid:` è la funzione Posix che crea una nuova *sessione*. Il processo diventa il leader
  - della nuova sessione
  - del nuovo *process group*.

  Il processo non ha terminale di controllo.

- Ignorare `SIGHUP` e invocare ancora `fork`. Il primo processo figlio termina lasciano il secondo in esecuzione. In questo modo il daemon non può acquisire automaticamente un *controlling terminal* nel caso in cui aprisse un terminale (come dispositivo).

  È necessario ignorare SIGHUP altrimenti, quando il session leader (cioè il primo processo figlio) termina, a tutti i processi delle sessione viene inviato il segnale `SIGHUP`.

- si può utilizzare `syslog` invece di `fprintf` per riportare eventuali errori.
- Cambiare la *working directory* ed effettuare il reset della *file mode creation mask*.
  - se il daemon crea propri file, i permessi ereditati non impattano sui permessi dei nuovi file.
- Chiudere qualunque descrittore aperto. Ricordiamo che non esiste una primitiva UNIX standard per sapere quale è il descrittore più grande (come numero) in uso.

  Alcuni daemon aprono `/dev/null` in lettura e scrittura. In questo modo si garantisce che esista uno standard input e ed uno standard output.

  Questo impedisce il fallimento di funzioni di libreria invocate dal daemon che possono tentare di leggere dallo standard input

oppure di scrivere su standard output o standard error.

- Normalmente `SIGHUP` (o altri segnali come `SIGINT` o `SIGWINCH` che il kernel non dovrebbe mai inviare al daemon) sono usati per informare il daemon che il file di configurazione è cambiato.