Laboratorio di Reti e Sistemi Distribuiti

5: I/O Multiplexing

Roberto Marino, PhD[1]
roberto.marino@unime.it

[1]Dipartimento di Matematica, Informatica, Fisica e Scienze della Terra
Future Computing Research Laboratory
Università di Messina

Last Update: 5th March 2025

I/O Multiplexing

Definizione

Tecnica che permette a un singolo thread di monitorare multipli file descriptor per:

- Lettura
- Scrittura
- Eccezioni

Perchè è importante?

- Evitare busy waiting
- Gestire connessioni concorrenti usando un singolo thread
- Ottimizzare l'utilizzo delle risorse

La syscall select()

```c
#include <sys/select.h>

int select(int nfds,
           fd_set *readfds,
           fd_set *writefds,
           fd_set *exceptfds,
           struct timeval *timeout);
```

- `nfds`: Numero massimo FD + 1
- `readfds`: Set di FD da monitorare per **lettura**
- `writefds`: Set di FD da monitorare per **scrittura**
- `exceptfds`: Set di FD da monitorare per **eccezioni**
- `timeout`: Timeout di attesa (struct timeval o NULL)

Select Man Page

readfds

I descrittori di file di questo insieme vengono osservati per vedere se sono pronti per la lettura. Un descrittore di file è pronto per la lettura se l'operazione di lettura non si blocca. In particolare, un descrittore di file è pronto anche alla fine del file.

writefds

I descrittori di file in questo set vengono osservati per vedere se sono pronti per la scrittura. Un descrittore di file è pronto per la scrittura se un'operazione di scrittura non si blocca.

Select Man Page

exceptfds

I descrittori di file di questo set vengono osservati per "condizioni eccezionali" (esempio: il client chiude bruscamente la connessione o il socket riceve un RST)

nfds

Questo argomento deve essere impostato **al valore del descrittore di file con il numero più alto in uno qualsiasi dei tre insiemi, più 1**. Vengono controllati i descrittori di file indicati in ciascun insieme, fino a questo limite. Limite di 1024 file descriptor!!

select(): valori di ritorno

- > 0: Numero **totale** di file descriptors per cui si è verificato un evento (lettura, scrittura, eccezione)
- 0: Timeout scaduto (nessun file descriptor pronto)
- -1: Errore (controllare errno)

Come funziona a basso livello la select()?

- select() è una system call che interagisce direttamente con il kernel per monitorare lo stato di multipli file descriptor (FD). Sposta la gestione delle connessioni multiple in kernel space, in un certo senso "fuori dal processo".
- Il kernel mantiene per ogni FD **una lista di processi in attesa di eventi** (es: lista dei processi in attesa di dati su un socket). Quando dati arrivano su un socket, il kernel itera la lista dei processi in attesa e li risveglia.

Come funziona a basso livello la select()?

1. **Preparazione dei File Descriptor:** L'utente definisce tre set di FD (readfds, writefds, exceptfds) da monitorare. Questi set sono copiati dal userspace al kernelspace via system call.
2. **Registrazione nel Kernel:** Il kernel aggiunge il processo chiamante alle code di attesa associate a ciascun FD monitorato.(Esempio: Per un socket in attesa di dati, il processo viene messo in coda agli eventi di ricezione di quel socket.)
3. **Attesa Eventi:** Il kernel sospende l'esecuzione del processo (se non è specificato un timeout di 0).
   Il processo viene spostato nello stato "interruptible sleep" (attesa interrompibile).
4. **Notifica degli Eventi:**
   Quando almeno un FD diventa pronto o scade il timeout il kernel "sveglia" il processo. Quindi i set di FD vengono modificati per riflettere solo i FD pronti, select() restituisce al processo il numero di FD pronti.
5. **Pulizia:** Il kernel rimuove il processo dalle code di attesa dei FD non più monitorati.

Esempio di utilizzo 1

```c
fd_set readfds;
FD_ZERO(&readfds);
FD_SET(sockfd, &readfds);

// Attende (blocca) fino a 5 secondi + 0 microsecondi
struct timeval tv = {5, 0};
int ready = select(sockfd + 1, &readfds, NULL, NULL, &tv);

if (ready == 0) {
    printf("Timeout scaduto\n");
} else if (ready > 0) {
    printf("FD pronto!\n");
}
```

Esempio di utilizzo 2

```c
#include <sys/select.h>
#include <stdio.h>

int main() {
    int sockfd = ...; // Socket gi  connesso
    fd_set readfds, exceptfds;
    struct timeval tv;

    FD_ZERO(&readfds);
    FD_ZERO(&exceptfds);
    FD_SET(sockfd, &readfds);
    FD_SET(sockfd, &exceptfds);

    tv.tv_sec = 5;
    tv.tv_usec = 0;

    int ready = select(sockfd + 1, &readfds, NULL, &exceptfds, &tv);
```

Esempio di utilizzo 2

```
1
2       if (ready == -1) {
3           perror("select failed");
4       } else if (ready == 0) {
5           printf("Timeout scaduto.\n");
6       } else {
7           if (FD_ISSET(sockfd, &readfds)) {
8               printf("Dati disponibili per lettura.\n");
9           }
10          if (FD_ISSET(sockfd, &exceptfds)) {
11              printf("Eccezione rilevata sul socket.\n");
12          }
13      }
14
15      return 0;
16  }
```

Domande

1. La select() è una chiamata bloccante o non bloccante?
2. Qual'è la sua complessità computazionale?

Risposte

1. La select() è una chiamata bloccante o non bloccante?
2. Qual'è la sua complessità computazionale?

1. Dipende dal valore di timeout (specificato in timeval)
2. La sua complessità è O(n), dove **n** è il numero di FD monitorati
3. Una alternativa migliore è epoll() che è O(1)

Macro Utili e Struttura Timeval

Inserire, rimuovere e verificare FD:

```
FD_ZERO(&set);           // Inizializza set
FD_SET(fd, &set);        // Aggiungi FD al set
FD_CLR(fd, &set);        // Rimuovi FD dal set
FD_ISSET(fd, &set);      // Verifica presenza FD
```

Struttura di definizione del timeout:

```
struct timeval {
    time_t      tv_sec;  // Secondi
    suseconds_t tv_usec; // Microsecondi (0-999999)
};
```