Universita' di Messina — CdL in Informatica — Reti e Sistemi Distribuiti Mod.B

Hands-On #5

I/O multiplexing con `select()`: da un client a molti

Last Update: March 19, 2026

Scenario

Il server TCP che hai scritto nell'HO#1 funziona correttamente con un singolo client. Ma prova ad aprire due terminali e a connetterti due volte contemporaneamente: il secondo client resta in attesa finche' il primo non si disconnette. Il server e' bloccato su `recv()` e non puo' fare altro.

Questo non e' un bug nel codice — e' un limite architetturale. `recv()` e' una chiamata bloccante: sospende l'intero processo finche' arrivano dati da *quel* socket. Nel frattempo tutti gli altri client vengono ignorati.

In questo hands-on imparerai ad usare `select()` per risolvere il problema: un solo thread, nessun blocco, N client gestiti contemporaneamente.

Concetti chiave da tenere a mente

Socket di ascolto e socket di connessione

Quando scrivi un server TCP usi in realta' due tipi di socket molto diversi tra loro.

Il **socket di ascolto** (`server_fd`) viene creato una volta sola con `socket()`, configurato con `bind()` e attivato con `listen()`. Non scambia mai dati con i client: il suo unico scopo e' ricevere *richieste* di connessione. Quando `select()` segnala che `server_fd` e' pronto, significa che un nuovo client vuole connettersi — si chiama `accept()`.

Il **socket di connessione** (`client_fd`) viene creato da `accept()` ogni volta che un client si connette. E' il canale dedicato a quella specifica connessione: su di esso si usano `recv()` e `send()`. Ne esiste uno per ogni client connesso contemporaneamente.

	Socket di ascolto	**Socket di connessione**
Quanti	Uno solo, per tutta la vita del server	Uno per ogni client connesso
Creato da	`socket()` + `bind()` + `listen()`	`accept()`
Usato per	Ricevere richieste di connessione	`recv()` / `send()`
Pronto in select	Nuovo client vuole connettersi	Dati disponibili o disconnessione

Le macro di `fd_set`

`select()` lavora con insiemi di file descriptor rappresentati dalla struttura `fd_set`, manipolata tramite quattro macro:

Macro	Significato
`FD_ZERO(&set)`	Azzera il set (obbligatorio prima di ogni ciclo)
`FD_SET(fd, &set)`	Aggiunge `fd` al set
`FD_CLR(fd, &set)`	Rimuove `fd` dal set
`FD_ISSET(fd, &set)`	Vero se `fd` e' pronto (usato dopo `select`)

[!] `select()` distrugge il set in input

Dopo la chiamata, `fd_set` contiene **solo** i fd pronti — tutti gli altri vengono rimossi. Devi ricostruire il set da zero ad ogni iterazione del loop, usando un array separato come "rubrica" dei client attivi.

Il primo argomento di `select()`: `nfds`

`select()` prende come primo argomento `nfds`, che deve essere il valore del **fd piu' grande tra tutti quelli monitorati, piu' uno**. Il kernel itera sui fd da 0 fino a `nfds-1`: se `nfds` e' troppo piccolo, alcuni fd non vengono monitorati; se e' troppo grande, si sprecano iterazioni inutili.

Listing 1: Come calcolare nfds

```c
int max_fd = server_fd;  /* unico fd all'avvio */

/* quando arriva un nuovo client con fd=7: */
if (client_fd > max_fd)
    max_fd = client_fd;  /* aggiorna il massimo */

/* nella chiamata a select: */
select(max_fd + 1, &read_set, NULL, NULL, NULL);
```

[!] Ricalcola `max_fd` alla disconnessione

Quando un client si disconnette, il suo fd viene chiuso e lo slot liberato. Se quel fd era il massimo, `max_fd` va ricalcolato scorrendo l'intera rubrica. Non farlo causa un comportamento indefinito.

I tre valori di ritorno di `select()`

Listing 2: Gestione corretta del valore di ritorno

```c
int n = select(max_fd + 1, &read_set, NULL, NULL, NULL);

if (n < 0) {
    if (errno == EINTR) continue; /* segnale: riprova */
    perror("select"); break;      /* errore vero      */
}
if (n == 0) {
    /* timeout scaduto: nessun fd pronto */
}
if (n > 0) {
    /* n fd sono pronti: controlla con FD_ISSET */
}
```

Codice da analizzare: `server_select.c`

Ti viene fornito il file `server_select.c`: un server multi-client funzionante che usa `select()`. Leggi il codice con attenzione prima di procedere con gli esercizi. Osserva in particolare:

- Come viene inizializzato `client_fds[]` e perche' a `-1`
- Perche' `max_fd` parte da `server_fd`
- Dove e come viene ricostruito il set ad ogni ciclo
- La differenza tra l'evento su `server_fd` e quello su `client_fds[i]`
- Come viene aggiornato `max_fd` all'arrivo e alla partenza di un client

Listing 3: Struttura centrale di server_select.c

```c
/* socket di ascolto: uno solo, per tutta la vita del server */
int server_fd = socket(AF_INET, SOCK_STREAM, 0);
bind(server_fd, ...); listen(server_fd, 5);

/* rubrica dei socket di connessione: uno per client */
int client_fds[MAX_CLIENTS];
for (int i = 0; i < MAX_CLIENTS; i++) client_fds[i] = -1;
int max_fd = server_fd;

while (1) {
    /* 1. ricostruisci il set da zero ad ogni ciclo */
    fd_set read_set;
    FD_ZERO(&read_set);
    FD_SET(server_fd, &read_set);
    for (int i = 0; i < MAX_CLIENTS; i++)
        if (client_fds[i] != -1)
            FD_SET(client_fds[i], &read_set);

    /* 2. blocca finche' almeno un fd e' pronto */
    int n = select(max_fd + 1, &read_set, NULL, NULL, NULL);

    /* 3. nuovo client? */
    if (FD_ISSET(server_fd, &read_set)) {
        int client_fd = accept(server_fd, NULL, NULL);
        /* salva in client_fds[], aggiorna max_fd */
    }

    /* 4. dati da un client esistente? */
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (client_fds[i] != -1 &&
            FD_ISSET(client_fds[i], &read_set)) {
            ssize_t r = recv(client_fds[i], buf, BUFLEN-1, 0);
            if (r <= 0) { close(client_fds[i]);
                          client_fds[i] = -1; }
            else { /* elabora buf */ }
        }
    }
}
```

Esercizi

Esercizio 1 — Analisi di `server_select.c`

Compila ed esegui `server_select.c`. Leggilo con attenzione riga per riga, poi rispondi alle seguenti domande *guardando il codice e osservando l'output del server*:

1. Qual e' la differenza tra `server_fd` e `client_fd`? Hanno lo stesso ruolo? Quante volte viene creato ciascuno durante una sessione con tre client?
2. Perche' `client_fds[]` viene inizializzato a `-1` e non a `0`? Cosa succederebbe se usassi `0` come valore sentinella?
3. Perche' `max_fd` parte da `server_fd` e non da `0`? Cosa succederebbe se passassi `0` come primo argomento a `select()`?
4. Perche' `FD_ZERO` e `FD_SET` vengono chiamati ad ogni iterazione del loop e non una volta sola prima del loop?
5. Connetti tre client con `nc 127.0.0.1 8080`. Disconnetti il secondo (CTRL+C). Il server crasha? Lo slot viene liberato? Come lo vedi nell'output?
6. Cosa stampa il server se si connettono tre client e ognuno manda un messaggio quasi contemporaneamente? `select()` ritorna sempre con `n=1` oppure a volte con `n > 1`?

▷ Per osservare n > 1

Usa lo script `test_multifd.sh` fornito:

```
./test_multifd.sh 5
```

Poi aggiungi questa riga al server subito prima del loop sui client e ricompila:

```
printf(">>> select() ritorna: %d fd pronti\n", n);
```

Esercizio 2 — Modifica `server_tcp.c` per gestire piu' client

Parti da `server_tcp.c` dell'HO#1 e trasformalo in un server multi-client usando `select()`, in modo che funzioni correttamente con lo script `test_multifd.sh`.
Il tuo obiettivo e' che questo comando:

```
./test_multifd.sh 5
```

produca sull'output del server cinque messaggi ricevuti da cinque fd diversi, senza che il server si blocchi o perda connessioni.

Le modifiche necessarie sono esattamente tre:

1. **Struttura dati** — sostituisci il singolo `client_fd` con una rubrica e un contatore del fd massimo:

```
int client_fds[MAX_CLIENTS];
for (int i = 0; i < MAX_CLIENTS; i++) client_fds[i] = -1;
int max_fd = server_fd;
```

2. **Accettazione connessioni** — sostituisci `accept()` fisso con `select()` + `FD_ISSET(server_fd)`. Il set va ricostruito ad ogni ciclo con `FD_ZERO` + `FD_SET` su tutti i fd attivi.
3. **Ricezione dati** — sostituisci il `recv()` diretto con un loop che controlla tutti i client con `FD_ISSET`, chiama `recv()` solo su quelli pronti e gestisce la disconnessione impostando lo slot a `-1`.

Verifica il funzionamento:

- `./test_multifd.sh 1` — un solo client: deve funzionare esattamente come il server originale dell'HO#1
- `./test_multifd.sh 5` — cinque client: tutti i messaggi devono arrivare, nessuno deve essere perso
- `./test_multifd.sh 10` — dieci client: il server deve reggere senza crash

[!] Aggiungi SO_REUSEADDR

Subito dopo `socket()` aggiungi questa opzione per poter riavviare il server immediatamente senza aspettare che il sistema operativo rilasci la porta:

```
int opt = 1;
setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR,
           &opt, sizeof(opt));
```

[!] Ricalcola max_fd alla disconnessione

Quando un client si disconnette e il suo fd era il massimo, `max_fd` va ricalcolato scorrendo l'intera rubrica. Non farlo causa un comportamento indefinito di `select()`.

File da consegnare: `server_tcp_select.c` e report PDF con le risposte alle domande dell'Esercizio 1.

Consegna: inviare zip al docente (`roberto.marino@unime.it`)

Prossimo passo: `select()` scala male oltre qualche migliaio di connessioni — O(n) ad ogni ciclo. Prossimamente scopriremo epoll(), che è O(1) per notifica e sta alla base di server blasonati come Nginx o Redis.