

# cEsercitazione 2020-10-13

---

Durante la prima parte della lezione viene mostrato l'esercizio 3.1: realizzare un programma che ci permetta di giocare a TicTacToe in remoto.

L'esercizio è abbastanza semplice;

**\*\* work in progress \*\***

## Funzione select

---

Il problema con i tipi di client e server che abbiamo realizzato finora, è il fatto che se effettuiamo una **recvfrom()** non possiamo eseguirne un'altra finchè la prima non è stata eseguita; questo avviene perchè questo tipo di operazione è **bloccante**.

## Il problema

---

```
int s1;
int s2;

// creazione socket 1
// creazione socket 2

// bind socket 1
// bind socket 2

while(1){
    recvfrom(s1, buf, sizeof(buf), ...);
    // processare il buf

    recvfrom(s2, buf, sizeof(buf), ...);
```

```
// processare il buf  
}
```

il problema è chiaramente il fatto che se vogliamo leggere da due socket diverse, per leggere dalla seconda dobbiamo attendere che la prima operazione venga completata.

## Introduzione alla funzione select()

La funzione select() è anch'essa bloccante, ma non si pone in ascolto in modo da ricevere dei dati da un singolo canale di comunicazione, ma è in **ascolto**, ed attende **che si verifichi un evento**; nel momento in cui si verifica un evento, sarà possibile leggere dal canale di comunicazione.

La lettura è subordinata al risultato della select(): se la select ci dice che si è verificato un evento di lettura, sarà effettuata una lettura.

Il primo parametro è un intero che viene usato per specificare quali descrittori di socket voglio monitorare; è in grado di osservare eventi su una qualsiasi socket. Invece di invocare la **recvfrom()** usiamo una **select()**.

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

- **maxfds** è un intero che deve essere maggiore dei descrittori che vogliamo monitorare; se ad esempio vogliamo monitorare i descrittori da 0-9, maxfd sarà 10.
- **readfds** sono set di descrittori di lettura; è essenzialmente il set di descrittori di socket sui quali si è verificato un evento che consente di procedere con un'operazione di lettura. In poche parole: set di descrittori che sono pronti alla lettura.
- **writefds** sono set di descrittori di scrittura; è essenzialmente il set di descrittori di socket sui quali si è verificato un evento che consente di procedere con un'operazione di scrittura. In poche parole: set di descrittori che sono pronti alla

scrittura.

- **exceptfds** sono set di descrittori di lettura; l'insieme di descrittori di socket su cui si è verificato un evento eccezionale.

Infine, abbiamo una struttura che ci consente di specificare per quanto tempo la select dovrà bloccare il processo invocante; possiamo usare la struttura **timeval**, il valore '0' (la select non bloccherà il processo) oppure possiamo usare il valore **NULL** (la select blocca il processo all'infinito.)

```
struct timeval {  
    long tv_sec; // secondi  
    long tv_usec; // microsecondi  
}
```

| possiamo sospendere il processo per il tempo da noi desiderato.

## Altre funzioni

Oltre alla funzione select() abbiamo anche altre funzioni che sono usate a corredo della principale:

```
FD_ZERO(fd_set *fds); // consente di azzerare i set che vengono usati nella funzione select()  
FD_SET(int fd, fd_set *fds); // specifichiamo quale descrittore vogliamo monitorare in lettura  
FD_ISSET(int fd, fd_set *fds); // va a verificare se nell'insieme monitorato dalla select vi è fra i  
descrittori per i quali si è verificato un evento di lettura il descrittore specificato.(fd descrittore, fds  
fd set)
```

- **FD\_ZERO**: ad esempio, una volta che abbiamo deciso di metterci in ascolto sugli eventi di lettura, e abbiamo preparato la variabile di tipo fd\_set per gli eventi di lettura, la inizializziamo invocando proprio **FD\_ZERO**.

## Esempio di utilizzo

---

```

int s1, s2;
fd_set readfds;

// creazione e bind di s1 ed s2
while(1){
    FD_ZERO(&readfds);
    FD_SET(s1, &readfds); // aggiungiamo s1 all fd set
    FD_SET(s2, &readfds); // aggiungiamo s2 all fd set

    if(Select(s2+1, &readfds, 0, 0, 0) < 0){ // assumiamo che s2 > s1
        perror("select");
        exit(1);
    }

    // eseguita quando c'è un evento per s1
    if(FD_ISSET(s1, &readfds)){
        recvfrom(s1, buf, sizeof(buf), ...);
        // processiamo il buffer
    }
    // eseguita quando c'è un evento per s2
    if(FD_ISSET(s2, &readfds)){
        recvfrom(s2, buf, sizeof(buf), ...);
        // processiamo il buffer
    }
}

```

**Importante:** `Select(s2+1, &readfds, 0, 0, 0)` Quando invochiamo la `select()` in questo modo stiamo dicendo che :

- Con `s2+1` assumiamo che `s2 > s1` e quindi monitoriamo entrambi;
- Con `&readfds` stiamo passando il set di descrittori che vogliamo monitorare
- `0`: non sono interessato ad eventi di scrittura
- `0`: non sono interessato ad eventi eccezionali

- 0: non sono interessato alla temporizzazione (select() sospende il processo finché non arriva un evento).

L'operazione successiva è quella di controllare se si sono verificati eventi per una delle nostre socket con `if(FD_ISSET(s2, &readfds)){ ... }`

## Select nelle socket orientate ai flussi

---

```
int fd;
int newfd[10];
int next;

fd_set readfds;
FD_ZERO(&readfds);

while(1){
    FD_SET(fd, &readfds);
    select(maxfd + 1, &readfds, 0, 0, 0); // attendiamo per un evento di lettura

    if(FD_ISSET(fd, &readfds)){
        newfd[++next] = accept(fd, ...);
        FD_ZERO(&readfds);
        FD_SET(newfd[next], &readfds);

        if(newfd[next] > maxfd)
            maxfd = newfd[next];
    }

    if(FD_ISSET(newfd[i], &readfds)){
        read(newfd[i], buf, sizeof(buf));
        // processiamo il buffer
    }
}
```

Possiamo intercettare un evento di lettura che riguarda la disponibilità della richiesta di connessione, o per intercettare un evento di lettura che riguarda la possibilità di leggere dati da una socket connessa.

la `select` (unica) viene usata inizialmente per verificare se si verifica un evento di lettura utile per invocare `accept`: verifichiamo (dopo la `select` che si è messa in ascolto su un set di socket) se è presente la socket di benvenuto; per capire se c'è un evento di lettura che consente di leggere una richiesta di connessione, dobbiamo operare sul descrittore della socket di benvenuto (prevista dal lato del server.); questo controllo ci permette di proseguire quindi con l'**`accept()`**.

L'`accept` restituisce un descrittore; se voglio monitorare questo nuovo descrittore dal punto di vista della lettura, va inserito all'interno del set di descrittori che stiamo monitorando.

---

fine lezione 9