

**Corso di Laurea in Ingegneria delle Telecomunicazioni**

**Corso di Reti di Calcolatori**

**Docente: Simon Pietro Romano**

**[spromano@unina.it](mailto:spromano@unina.it)**

# **Le socket di Berkeley** (2a parte)

a cura di Marcello Esposito ([mesposit@unina.it](mailto:mesposit@unina.it))

# La gestione delle opzioni sulle socket

- ◆ Le opzioni che si possono impostare su una socket, ne influenzano il comportamento di default.
- ◆ Ci sono diverse possibilità per gestire le opzioni:
  - la funzione `setsockopt()`
  - la funzione `fcntl()`
  - la funzione `ioctl()`
- ◆ La prima è applicabile esclusivamente alle socket, mentre le seguenti due sono system call standard Unix.

# setsockopt () e getsockopt () (1)

```
#include <sys/types.h>
#include <sys/socket.h>

int getsockopt(int sockfd, int level, int optname,
               char *optval, int *optlen);
int setsockopt(int sockfd, int level, int optname,
               char *optval, int optlen);
```

- ◆ sockfd è un descrittore di socket;
- ◆ level specifica l'entità nel sistema che deve interpretare la particolare opzione;
- ◆ optname indica quale opzione si desidera impostare (set) o prelevare (get);
- ◆ optval è un puntatore contenente il valore dell'opzione da impostare (set) o da prelevare (get);
- ◆ optlen è la lunghezza della struttura puntata da optval, ed è un parametro di ingresso per set e di ingresso-uscita per get.

## setsockopt () e getsockopt () (2)

- ◆ Le opzioni disponibili sono molto numerose e dedicate alle più svariate esigenze.
- ◆ Per esempio:
  - `level=SOCKET, optname=SO_RCVBUF` si riferisce alla grandezza del buffer di ricezione. Questa dimensione può essere letta (`get`) o anche impostata (`set`).
  - `level=SOCKET, optname=SO_REUSEADDR` indica che per la socket in questione è possibile il riutilizzo di un indirizzo già assegnato (cosa che in genere provocherebbe un errore nella `bind`).

## Il problema della concorrenza (1)

- ◆ Immaginiamo un semplice programma di chat a due partecipanti.
- ◆ In ciascun istante, ogni processo deve:
  - inviare i messaggi che provengono dalla tastiera all'altro end-point;
  - visualizzare i messaggi che provengono su una socket dall'altro end-point.
- ◆ Non si sa quale dei due eventi si verifica per primo e non è quindi possibile scegliere di leggere uno dei due dispositivi di ingresso.
- ◆ Se infatti si scegliesse, per esempio, di leggere prima la tastiera, il programma resterebbe bloccato in attesa di input, anche se intanto arrivano dati dall'altro end-point attraverso l'apposita socket.

## La system-call `select()` (1)

- ◆ Il problema può essere risolto attraverso l'utilizzo della system-call `select()`.
- ◆ Essa permette al flusso di programma di bloccarsi finché almeno uno tra  $n$  file-descriptors si renda pronto per la lettura o per la scrittura.
- ◆ Questo genere di attesa viene definito *polling*.
- ◆ Quando ciò accade la `select()` si sblocca restituendo i particolari file-descriptors che hanno variato il loro stato.

## La system-call select () (2)

```
#include <sys/types.h>
#include <sys/time.h>

int select(int maxfdp1, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);

FD_ZERO(fd_set *fdset);           /* clear all bits */
FD_SET(int fd, fd_set *fdset);    /* turn on a bit */
FD_CLR(int fd, fd_set *fdset);    /* turn off a bit */
FD_ISSET(int fd, fd_set *fdset);  /* test a bit */

struct timeval {
    long tv_sec;           /* seconds */
    long tv_usec;         /* microseconds */
};
```

## La system-call `select()` (3)

- ◆ La `select()` controlla quale tra i primi `maxfdp1` descrittori si rende pronto per la lettura, per la scrittura o risulta soggetto a condizioni eccezionali.
- ◆ Il tipo `fd_set` è un vettore di bit, ciascuno associato ad un descrittore.
- ◆ Inizialmente gli `fdset` devono essere correttamente impostati con `FD_ZERO()` ed `FD_SET()`, e, dopo la `select()`, testati con `FD_ISSET()`.
- ◆ Se `timeout` vale `NULL`, la `select()` è bloccante e non si sblocca finché una delle condizioni si sia verificata.
- ◆ Se `timeout` punta ad una struttura `struct timeval`, la `select()` è bloccante ma, al massimo dopo il tempo specificato, si sblocca se niente è ancora accaduto.
  - in quest'ultimo caso, `timeout` può anche puntare ad una `struct timeval` i cui campi sono nulli, così da effettuare realmente un polling.
- ◆ Restituisce il numero totale di descrittori pronti.

# Esempio di `select()` (1)

- ◆ Supponiamo di volerci mettere in attesa che una tra 3 socket sia pronta ad essere letta.
- ◆ Di seguito sono riportate le azioni da compiere:

```
void main() {
    int sockfd1, sockfd2, sockfd3;
    fd_set readfds;
    int n;
    struct timeval tv;
    int maxfd;

    //Istanzio tre socket
    sockfd1 = socket(AF_INET, SOCK_DGRAM, 0);
    sockfd2 = socket(AF_INET, SOCK_DGRAM, 0);
    sockfd3 = socket(AF_INET, SOCK_DGRAM, 0);

    ...

    //Inizializzo ed imposto la variabile readfds con ciò che voglio testare
    FD_ZERO(&readfds);
    FD_SET(sockfd1, &readfds);
    FD_SET(sockfd2, &readfds);
    FD_SET(sockfd3, &readfds);
}
```

## Esempio di select () (2)

```
//Inizializzo tv per un'attesa massima di 5 secondi
tv.tv_sec = 5;
tv.tv_usec = 0;

//Calcolo il massimo tra i file-descriptor da testare
maxfd = sockfd1;
maxfd = (maxfd > sockfd2 ? maxfd : sockfd2);
maxfd = (maxfd > sockfd3 ? maxfd : sockfd3);

n = select(maxfd + 1, &readfds, NULL, NULL, &tv);

if (n > 0) {
    if (FD_ISSET(sockfd1, &readfds)) { //è per caso il primo ad essere pronto?
        ...
    }
    ... //o magari gli altri?
}

...
close(sockfd1);
close(sockfd2);
close(sockfd3);
}
```

## Esempio di `select()` (3)

- ◆ Se i valori di (`sockfd1`, `sockfd2`, `sockfd3`) fossero per esempio (3, 8, 10) e `sockfd2` fosse il primo e l'unico descrittore a divenire pronto per la lettura, la situazione sarebbe la seguente:

- Subito dopo la `FD_ZERO()`

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
readfds	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Subito dopo le `FD_SET()`

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
readfds	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0

- Subito dopo la `select()`

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
readfds	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

# Le Raw Socket

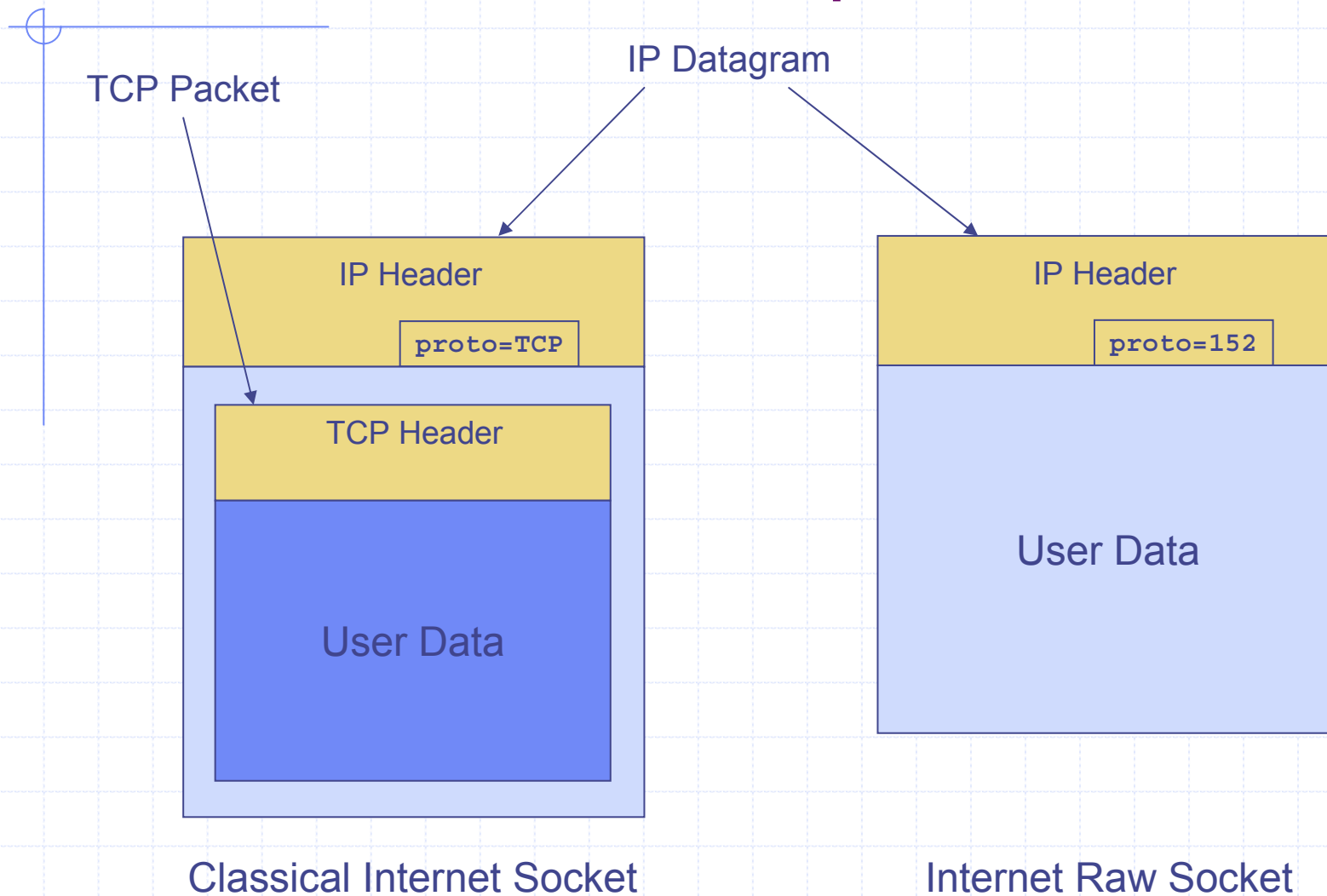
- ◆ La chiamata

```
sockfd = socket(AF_INET, SOCK_RAW, protocol);
```

produce l'istanziamento di una socket definita "Raw" (rozza).

- ◆ Una tale socket si aggancia direttamente al livello IP, con la conseguenza che è possibile di fatto creare un nuovo protocollo di quarto livello.
- ◆ Ogni messaggio viene quindi incapsulato direttamente in un pacchetto IP.
- ◆ Il valore di `protocol` identifica il numero che si desidera assegnare al protocollo in questione. Questo numero sarà pertanto il valore dell'omonimo campo nell'intestazione dei datagrammi IP inviati da una tale socket.

# Le Raw Socket: i pacchetti inviati



Classical Internet Socket

Internet Raw Socket

# Le Raw Socket: accesso all'header IP

- ◆ Una volta allocata una raw socket, con una chiamata del tipo:

```
int hincl = 1;  
setsockopt(sockfd, IPPROTO_IP, IP_HDRINCL, &hincl,  
sizeof(hincl));
```

è possibile gestire completamente l'accesso all'intestazione del pacchetto IP, cambiando i singoli campi a proprio piacimento.

- ◆ Questa tecnica è destinata comunque a soddisfare esigenze molto particolari e, pertanto, non particolarmente diffuse.