

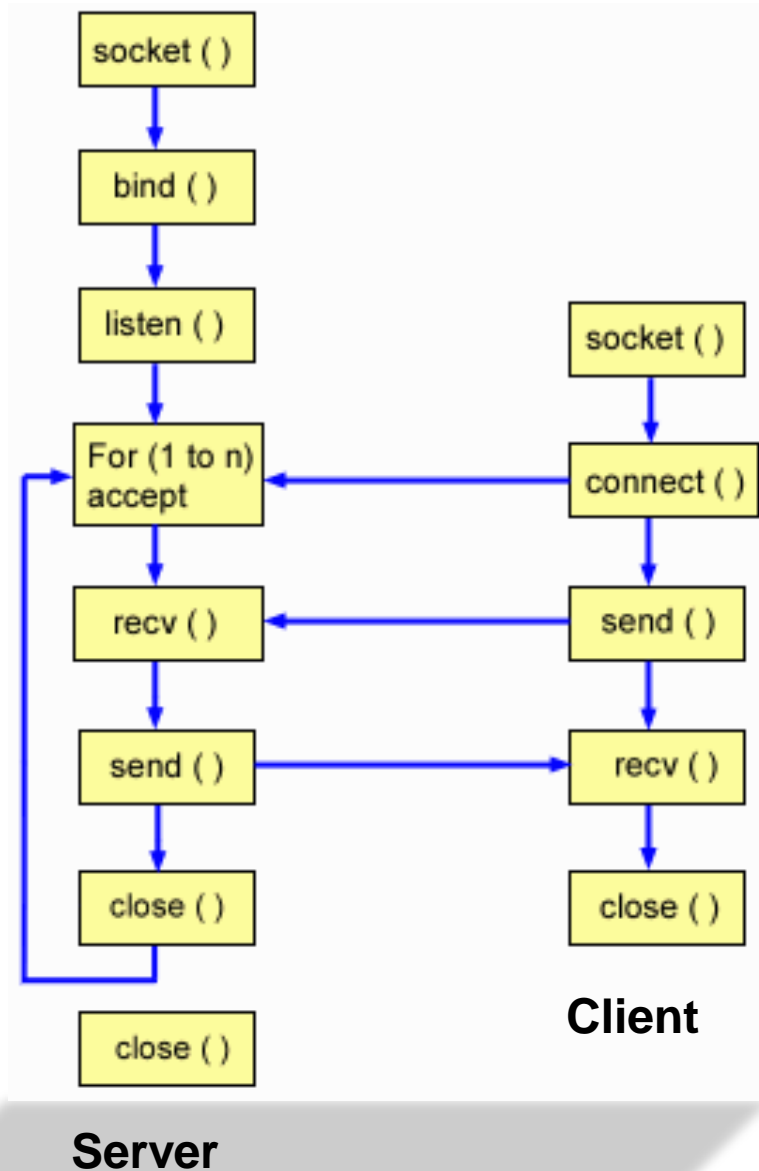
Programarea in retea (II)

Lenuta Alboaie
adria@info.uaic.ro

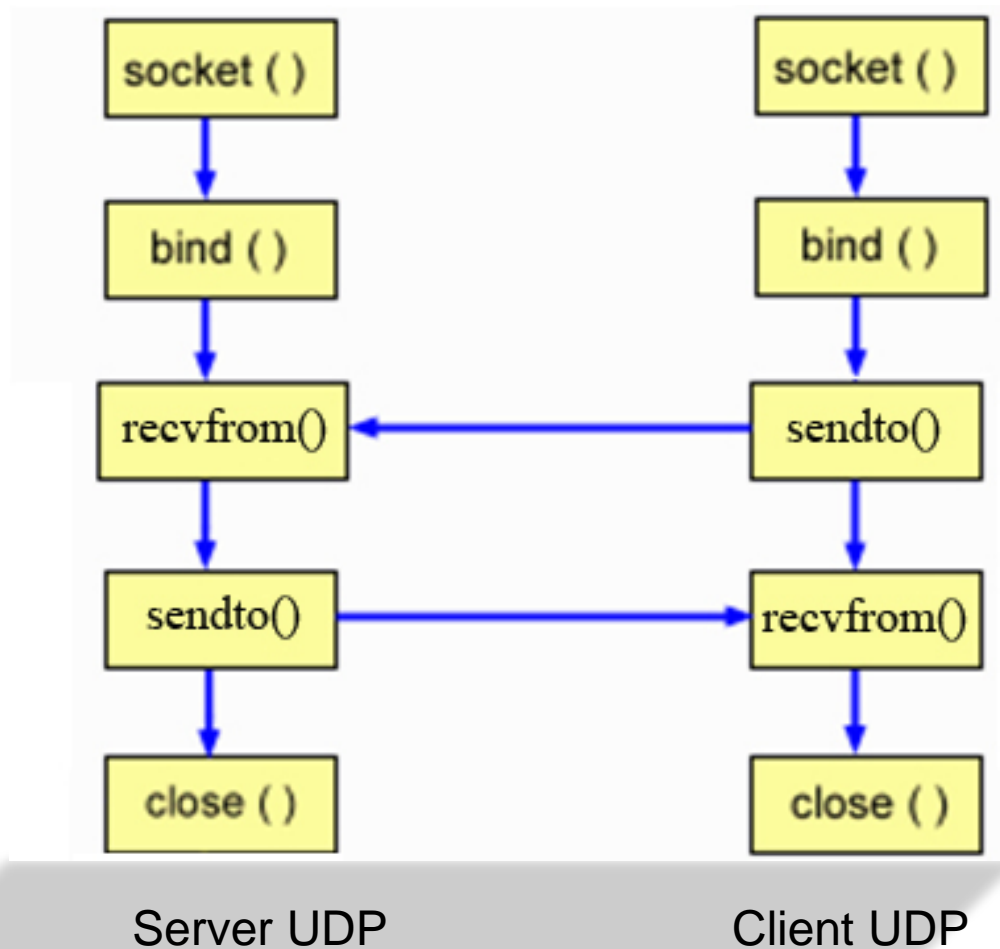
Cuprins

- ... sa ne amintim: client/server TCP iterativ - primitive
- Modelul client/server UDP
- Primitive I/O
- Aspecte de programare avansata Internet
- API-ului *socket* – discutii si critici

Model server/client TCP



Client/Server UDP



Client/Server UDP

- Pentru **socket()** se va folosi **SOCK_DGRAM**
- Apelurile **listen()**, **accept()**, **connect()** nu vor mai fi utilizate in mod uzual
- Pentru scriere de datagrame se pot folosi **sendto()** sau **send()** (mai general)
- Pentru citire de datagrame se pot folosi **recvfrom()** sau **recv()**
- Nimeni nu garanteaza ca datele expediate au ajuns la destinatar sau nu sunt duplicate

Client/Server UDP

- Socket-urile UDP pot fi “conectate”: clientul poate folosi `connect()` pentru a specifica adresa (IP, port) a punctului terminal (serverul) – **pseudo-conexiuni**:
 - Utilitate: trimiterea mai multor datagrame la acelasi server, fara a mai specifica adresa serverului pentru fiecare datagrama in parte
 - Pentru UDP, `connect()` va retine doar informatiile despre punctul terminal, fara a se initia nici un schimb de date
 - Desi `connect()` raporteaza succes, nu inseamna ca adresa punctului terminal e valida sau serverul este disponibil

Client/Server UDP

- Pseudo-conexiuni UDP
 - Se poate utiliza `shutdown()` pentru a opri directionat transmiterea de date, dar nu se va trimite nici un mesaj partenerului de conversatie
 - Primitiva `close()` poate fi apelata si pentru a elimina o pseudo-conexiune



Alte primitive | I/O

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

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

```
int send (int sockfd, char *buff, int nbytes, int flags);
```

```
int recv (int sockfd, char *buff, int nbytes, int flags);
```

- Pot fi folosite in cadrul comunicatiilor orientate conexiune sau pentru pseudo-conexiuni

Apelurile **send()** si **recv()** presupun că sunt cunoscute toate elementele unei asocieri, adică a fost efectuat în prealabil un apel **connect()**

- Primele 3 argumente sunt similare cu cele de la **write()**, respectiv **read()**
- Argumentul a patrulea este de regulă 0, dar poate avea si alte valori care precizează condiții de efectuare a apelului
- Cele 2 apeluri returnează la execuție normală lungimea transferului în octeți

Alte primitive | I/O

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

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

```
int sendto ( int sockfd, char *buff, int nbytes, int flags,  
            struct sockaddr *to, int addrlen);
```

```
int recvfrom (int sockfd, char *buff, int nbytes, int flags,  
              struct sockaddr *from, int *addrlen);
```

- Sunt folosite pentru comunicatii neorientate conexiune
- La **sendto()** si **recvfrom()** elementele pentru identificarea nodului la distanță se specifică în apel, prin ultimele 2 argumente
- Cele 2 apeluri returnează la execuție normală lungimea transferului în octeți



Alte primitive | I/O

```
#include <sys/uio.h>
```

```
ssize_t readv (int fd, const struct iovec *iov, int iovcnt);
```

```
ssize_t writev (int fd, const struct iovec *iov, int iovcnt);
```

- Mai generale decât read()/write(), ofera posibilitatea de a lucra cu date aflate in zone necontigue de memorie

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

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

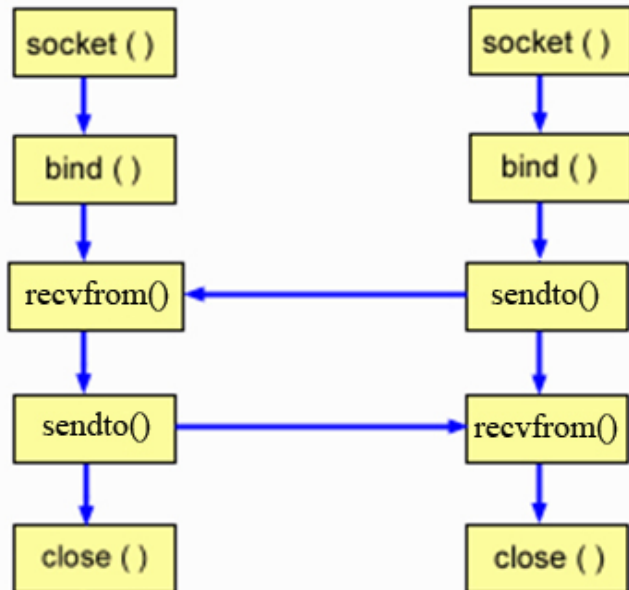
```
ssize_t recvmsg (int s, struct msghdr *msg, int flags);
```

```
ssize_t sendmsg (int s, const struct msghdr *msg, int flags);
```

- Receptioneaza/transmite mesaje extragindu-le din structura *msghdr*

DEMO

Exemplu de server/client UDP



Server UDP

Client UDP



Alte primitive | informatii

- **getpeername()** – returneaza informatii despre celalalt capat al conexiunii

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

```
int getpeername (int sockfd, struct sockaddr *addr,  
                 socklen_t *addrlen);
```

- **getsockname()** – returneaza informatii asupra socketului(local) specificat → (adresa la care este atasat)

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

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

```
int getsockname( int sockfd, struct sockaddr * addr,  
                 socklen_t * addrlen);
```

Programare retea avansata

- Optiuni atasate socket-urilor
 - `getsockopt()` si `setsockopt()`
- Multiplexare I/O

Primitive | optiuni

- **Optiuni** atasate *socket*-urilor
 - Atribute utilizate pentru consultarea sau modificarea unui comportament, general ori specific unui protocol, pentru unele (tipuri de) *socket*-uri
 - Tipuri de valori:
 - Booleene (*flag*-uri)
 - Mai “complexe”:
int, timeval, in_addr, sock_addr, etc

Primitive | optiuni

- **getsockopt()** – consultarea optiunilor

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

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

```
int getsockopt (int sockfd, int level, int optname, void *optval,  
socklen_t *optlen);
```

Numele, valoarea si lungimea optiunii

Level - indica daca optiunea este generala sau specifica unui protocol

Exemplu:

```
len = sizeof (optval);
```

```
getsockopt (sockfd, SOL_SOCKET, SO_REUSEADDR, &optval, &len);
```

Primitive | optiuni

- **setsockopt()** – setarea optiunilor

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

```
int setsockopt (int sockfd, int level, int optname, void *optval,  
                socklen_t *optlen);
```

Returneaza:

- 0 = succes
- -1 = eroare: EBADF, ENOTSOCK, ENOPROTOOPT,...

Numele, valoarea si lungimea
optiunii

Primitive | optiuni

Optiuni generale

- Independente de protocol
- Unele suportate doar de anumite tipuri de socketuri (`SOCK_DGRAM`, `SOCK_STREAM`)
 - `SO_BROADCAST`
 - `SO_ERROR`
 - `SO_KEEPALIVE`
 - `SO_LINGER`
 - `SO_RCVBUF`, `SO_SNDBUF`
 - `SO_REUSEADDR`
 - `SO_OOBINLINE`
 - ...

[<http://www.beej.us/guide/bgnet/output/html/multipage/setsockoptman.html>]

Primitive | optiuni

- **SO_BROADCAST** (boolean)
 - Activeaza/dezactiveaza trimiterea de date in regim broadcast
 - Utilizata doar pentru SOCK_DGRAM
 - Previne anumite aplicatii sa nu trimita in mod neadecvat *broadcast*-uri
- **SO_ERROR** (int)
 - Indica eroarea survenita (similara lui errno)
 - Poate fi folosita cu primitiva **getsockopt()**
- **SO_KEEPALIVE** (boolean)
 - Folosita pentru SOCK_STREAM
 - Se va trimite o informatie de proba celui alt punct terminal daca nu s-a realizat schimb de date timp indelungat
 - Utilizata de TCP (e.g., telnet): permite proceselor sa determine daca procesul/gazda cealalta a picat

Primitive | optiuni

- **SO_LINGER** (struct **linger**)

- Controleaza daca si dupa cit timp un apel de inchidere a conexiunii va astepta confirmari(ACK-uri) de la punctul terminal
- Folosita doar pentru socket-uri orientate-conexiune pentru a ne asigura ca un apel `close()` nu va returna imediat
- Valorile vor fi de tipul:

```
struct linger {  
    int l_onoff;    /* interpretat ca boolean */  
    int l_linger;   /* timpul in secunde*/  
}
```

- **l_onoff = 0**: `close()` returneaza imediat, dar datele netrimise sunt transmise
- **l_onoff != 0** si **l_linger = 0**: `close()` returneaza imediat si datele netrimise sunt sterse
- **l_onoff != 0** si **l_linger != 0**: `close()` nu returneaza pina cind datele netrimise sunt transmise (sau conexiunea este inchisa de sistemul remote)

Primitive | optiuni

- **SO_LINGER** – Exemplu

```
int result;
```

```
struct linger lin;
```

```
lin.l_onoff=1 ; /*0 -> l_linger este ignorata */
```

```
lin.l_linger=1; /* 0 = pierderea datelor; nonzero= asteptare pina  
se trimit datele */
```

```
result= setsockopt( sockfd,  
                    SOL_SOCKET,  
                    SO_LINGER,  
                    &lin, sizeof(lin));
```



Primitive | optiuni

- **SO_RCVBUF/SO_SNDBUF** (int)
 - Modifica dimensiunile *buffer*-elor de receptionare sau de trimitere a datelor
 - Utilizate pentru SOCK_DGRAM si SOCK_STREAM

- **Exemplu:**

```
int result; int bufsize = 10000;
```

```
result= setsockopt (s, SOL_SOCKET, SO_SNDBUF, &bufsize,  
sizeof(bufsize));
```



Primitive | optiuni

- **SO_REUSEADDR** – (boolean)
 - Permite atasarea la o adresa(port) deja in uz
 - nu incalca regula de asociere unica realizata de bind
 - Folosita pentru ca un *socket pasiv* sa poata folosi un port deja utilizat de alte procese

Stare 1

```
Active connections (including servers)
Proto Recv-Q Send-Q Local Address Foreign Address (state)
tcp      0      0 *.2000          *.*             LISTEN
```

Stare 2

```
Proto Recv-Q Send-Q Local Address Foreign Address (state)
tcp 0 0 192.6.250.100.2000 192.6.250.101.4000 ESTABLISHED
tcp 0 0 *.2000 *.*             LISTEN
```

- Daca *daemon*-ul care asculta la portul 2000 este *killed*, incercarea de restart a *daemon*-ului va esua daca **SO_REUSEADDR** nu este setat

Exemplu

```
int optval = 1;
```

```
setsockopt (sockfd, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval));
```

```
bind (sockfd, &sin, sizeof(sin) );
```

Primitive | optiuni

Optiuni specifice protocolului IP

- **IP_TOS** permite setarea cimpului “Type Of Service” (e.g., ICMP) din antetul IP
- **IP_TTL** permite setarea cimpului “Time To Live” din antetul IP

Exista si optiuni pentru IPv6.(RFC 2460,2462)

-**IPV6_V6ONLY**, ...

Primitive | optiuni

Optiuni specifice protocolului TCP

- **TCP_KEEPALIVE** seteaza timpul de asteptare daca **SO_KEEPALIVE** este activat
- **TCP_MAXSEG** stabileste lungimea maxima a unui segment (nu toate implementarile permit modificarea acestei valori de catre aplicatie)
- **TCP_NODELAY** seteaza dezactivarea algoritmului Nagle (reducerea numarului de pachete de dimensiuni mici intr-o retea WAN; TCP va trimite intotdeauna pachete de marime maxima, daca este posibil) – utilizata pentru generatori de pachete mici (e.g., clienti interactivi precum *telnet*)

Multiplexare I/O

- Posibilitatea de a monitoriza mai multi descriptori I/O
 - Un client TCP generic (e.g., *telnet*)
 - Un client interactiv (e.g., *ftp*, *scp*, *browser Web*,...)
 - Un server care poate manipula mai multe protocoale (TCP si UDP) simultan
 - Rezolvarea unor situatii neasteptate (i.e. caderea unui server in mijlocul comunicarii)
- Exemplu: datele citite de la intrarea standard trebuie scrise la un socket, iar datele receptionate prin retea trebuie afisate la iesirea standard

Multiplexare I/O | solutii

- Utilizarea mecanismului neblokant folosind primitivele **fnctl()** / **ioctl()**
- Utilizarea mecanismului asincron
- Folosirea **alarm()** pentru a intrerupe apelurile de sistem lente
- Utilizarea unor *procese/thread*-uri multiple (*multitasking*)
- Folosirea unor primitive care suporta verificarea existentei datelor de intrare de la descriptori de citire multipli: **select()** si **poll()**



Multiplexare I/O | solutii

- Utilizarea mecanismului neblokant folosind primitiva **fcntl()**
 - Se seteaza apelurile I/O ca neblocante
`int flags;`
`flags = fcntl (sd, F_GETFL, 0);`
`fcntl(sd, F_SETFL, flags | O_NONBLOCK);`
 - Daca nu sunt date disponibile un apel **read()** va intoarce -1 sau daca nu este suficient spatiu in *buffer* un apel **write()** va intoarce -1 (cu eroarea **EAGAIN**)

Multiplexare I/O | solutii

Utilizarea mecanismului neblokant folosind primitiva **ioctl()**

```
#include <sys/ioctl.h>
```

```
ioctl (sd, FIOCNBIO, &arg);
```

-arg este un pointer la un int
-Daca int are valoare 0, socketul este setat in mod blocant
-Daca int are valoare 1, socketul este setat in mod neblokant

Daca socketul este in mod neblokant, urmatoarele apeluri sunt afectate astfel:

- **accept()** – daca nu este prezenta nici o cerere, *accept()* returneaza cu eroarea **EWOULDBLOCK**
- **connect()** – daca conexiunea nu se poate stabili imediat, *connect()* returneaza cu eroarea **EINPROGRESS**
- **recv()** – daca nu exista date de primit, *recv()* returneaza -1 cu eroarea **EWOULDBLOCK**
- **send()** – daca nu exista spatiu in buffer pentru ca datele sa fie transmise, *send()* returneaza -1 cu eroarea **EWOULDBLOCK**

Multiplexare I/O | solutii

Trimiterea si receptarea datelor in mod asincron

- **Problema:** In conditiile in care *socket*-urile sunt create implicit in mod blocant (I/O), cum s-ar putea instiinta procesul atunci cind se intimpla “ceva” la un socket?
- **Socket**-urile **asincrone** permit trimiterea unui semnal (**SIGIO**) procesului
- Socket-urile asincrone permit utilizatorului separarea “*procesarilor socket*” de alte procesari
- Generarea semnalului **SIGIO** este dependenta de protocol

Multiplexare I/O | solutii

Trimiterea si receptarea datelor in mod asincron

- Pentru TCP semnalul SIGIO poate aparea cind:
 - Conexiunea a fost complet stabilita
 - O cerere de deconectare a fost initiata
 - Cererea de deconectare a fost realizata complet
 - *shutdown()* pentru o directie a comunicatiei
 - Au aparut date de la celalalt punct terminal
 - Datele au fost trimise
 - Eroare

Multiplexare I/O | solutii

Trimiterea si receptarea datelor in mod asincron

- Pentru UDP semnalul SIGIO apare cind:
 - Se receptioneaza o datagrama
 - Apare o eroare
- Putem permite proceselor sa realizeze alte activitati si sa monitorizeze transferurile UDP

Multiplexare I/O | solutii

Trimiterea si receptarea datelor in mod asincron

- Implementarea

- *Socket*-ul trebuie setat ca fiind asincron

- ```
#include <sys/unistd.h>
```

- ```
#include <sys/fcntl.h>
```

- ```
int fcntl (int s, int cmd, long arg)
```

Exemplu:

```
int sd = socket(PF_INET, SOCK_STREAM, 0);
```

```
fcntl (sd, F_SETFL, O_ASYNC); /* setarea asincrona I/O */
```



# Multiplexare I/O | solutii

- Utilizarea **alarmelor**

```
while(...){
```

```
 signal (SIGALRM, alarmHandler);
```

```
 alarm (MAX_TIME);
```

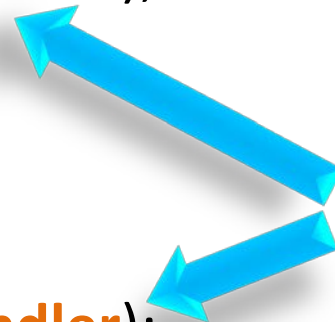
```
 read (0,...);
```

```
 signal (SIGALRM, alarmHandler);
```

```
 alarm (MAX_TIME);
```

```
 read (tcpsock,...);...
```

```
}
```



**Functie scrisa de  
programator**



# Multitasking

## Servere concurente – per-client process

## Servere concurente pre-forked

- Se creeaza un numar de procese copil imediat la initializare, fiecare proces liber interactionind cu un anumit client

## Servere concurente pre-threaded

- Ca mai sus, dar se folosesc *thread*-uri (fire de executie) in locul proceselor (vezi POSIX *threads* –pthread.h)
- Exemplu: serverul Apache

## Probleme:

- Numarul de clienti mai mare decit numarul de *proces/thread*-uri
- Numarul de *proces/thread*-uri prea mare fata de numarul de clienti
- *OS overhead*
- ..... (*curs viitor*)

# Multiplexare I/O | solutii

Probleme care apar:

- Folosind apeluri **neblocante**, se utilizeaza intens procesorul
- Pentru **alarm()**, care este valoarea optima a constantei **MAX\_TIME**?

# Multiplexarea I/O | select()

- Permite utilizarea apelurilor blocante pentru un set de descriptori (fisiere, pipe-uri, socket-uri,...)
- Suspenda programul pana cand descriptori din liste sunt pregatiti de operatii de I/O

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

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

```
#include <unistd.h>
```

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

Valoarea maxima a  
descript. plus 1

Numarul  
descriptorilor  
de citire,  
scriere,  
exceptie

Timpul de asteptare



# Multiplexarea I/O | select()

Manipularea elementelor multimii de descriptori (tipul **fd\_set**) se realizeaza folosindu-se macrourele:

|                                        |                                                                    |
|----------------------------------------|--------------------------------------------------------------------|
| <b>FD_ZERO</b> (fd_set *set);          | Sterge multimea de descriptori din set.                            |
| <b>FD_SET</b> (int fd, fd_set *set);   | Adauga descriptorul <b>fd</b> in multimea set.                     |
| <b>FD_CLR</b> (int fd, fd_set *set);   | Sterge descriptorul <b>fd</b> din multimea set.                    |
| <b>FD_ISSET</b> (int fd, fd_set *set); | Testeaza daca descriptorul <b>fd</b> apartine sau nu multimii set. |

# Multiplexarea I/O | select()

Pentru timpul de asteptare se foloseste structura definita in **sys/time.h**:

```
struct timeval {
 long tv_sec; /* secunde */
 long tv_usec; /* microsecunde */
}
```

- Daca **timeout** este NULL, **select()** va returna imediat
- Daca **timeout** este !=0 specifica intervalul de timp in care **select()** va astepta

# Multiplexarea I/O | `select()`

Un descriptor de socket este gata de **citire** daca:

- Exista octeti receptionati in *buffer*-ul de intrare (se poate face **`read()`** care va returna `>0`)
- O conexiune TCP a receptionat FIN (**`read()`** returneaza `0`)
- *Socket*-ul e *socket* de ascultare si nr. de conexiuni complete este nenul (se poate utiliza **`accept()`** )
- A aparut o eroare la *socket* (**`read()`** returneaza `-1`, cu **`errno`** setat) – erorile pot filtrate via **`getsockopt()`** cu optiunea **`SO_ERROR`**

# Multiplexarea I/O | `select()`

Un descriptor de *socket* este gata de **scriere** daca:

- Exista un numar de octeti disponibili in buffer-ul de scriere (**`write()`** va returna  $> 0$ )
- Conexiunea in sensul scrierii este inchisa  
(incercarea de **`write()`** va duce la generarea **`SIGPIPE`**)
- A aparut o eroare de scriere (**`write()`** returneaza  $-1$ , cu `errno` setat) – erorile pot fi filtrate via **`getsockopt()`** cu optiunea **`SO_ERROR`**



# Multiplexarea I/O | select()

- Un descriptor de socket este gata de **exceptie** daca:
  - Exista date *out-of-band* sau *socket*-ul este marcat ca *out-of-band* (curs viitor optional 😊)
  - Daca capatul *remote* a *socket*-ului TCP a fost inchis in timp ce date erau pe canal; urmatoarea operatie de read/write va intoarce ECONNRESET

# Multiplexarea I/O | select()

**select()** poate returna

- Numarul descriptorilor pregatiti pentru o operatiune de citire, scriere sau exceptie
- Valoarea 0 – timpul s-a scurs, nici un descriptor nu este gata
- Valoarea -1 in caz de eroare

Utilizarea lui **select()** – pasii generali:

- Declararea unei variabile de tip **fd\_set**
- Initializarea multimii cu **FD\_ZERO()**
- Adaugarea cu **FD\_SET()** a fiecarui descriptor dorit a fi monitorizat
- Apelarea primitivei **select()**
- La intoarcerea cu succes, verificarea cu **FD\_ISSET()** a descriptorilor pregatiti pentru I/O

Demo

Exemplu de utilizare a  
primitivei **select()**

# Socket-uri BSD | utilizare

- Serviciile Internet (serviciile folosesc *socket*-urile pentru comunicarea între *host-uri remote*)
  - Exemplu de aplicatii distribuite
    - World Wide Web
    - Accesul *remote* la o baza de date
    - Distribuirea de *task*-uri mai multor *hosturi*
    - Jocuri on-line
    - ...

# Socket-uri BSD | critici

API-ul bazat pe socket-uri BSD are o serie de limitari:

- Are o complexitate ridicata, deoarece a fost proiectata sa suporte familii de protocoale multiple (dar rar folosite in practica)
- Nu este portabila (unele apeluri/tipuri au alte denumiri/reprezentari pe alte platforme; numele fisierelor - *antet.h* depind de sistem)
- Exemplu: la WinSock descriptorii de *socket* sunt pointeri, in contrast cu implementarile Unix care folosesc intregi

# Rezumat

- ... sa ne amintim: client/server TCP iterativ - primitive
- Modelul client/server UDP
- Primitive I/O
- Aspecte de programare avansata Internet
- API-ului *socket* – discutii si critici



# Intrebari?

Intrebari?