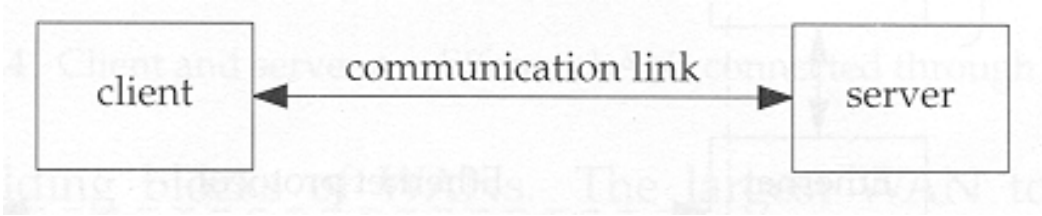
Εργαστήριο Δικτύων Υπολογιστών

3η Διάλεξη:

- Δικτυακός Προγραμματισμός
  Unix Socket programming

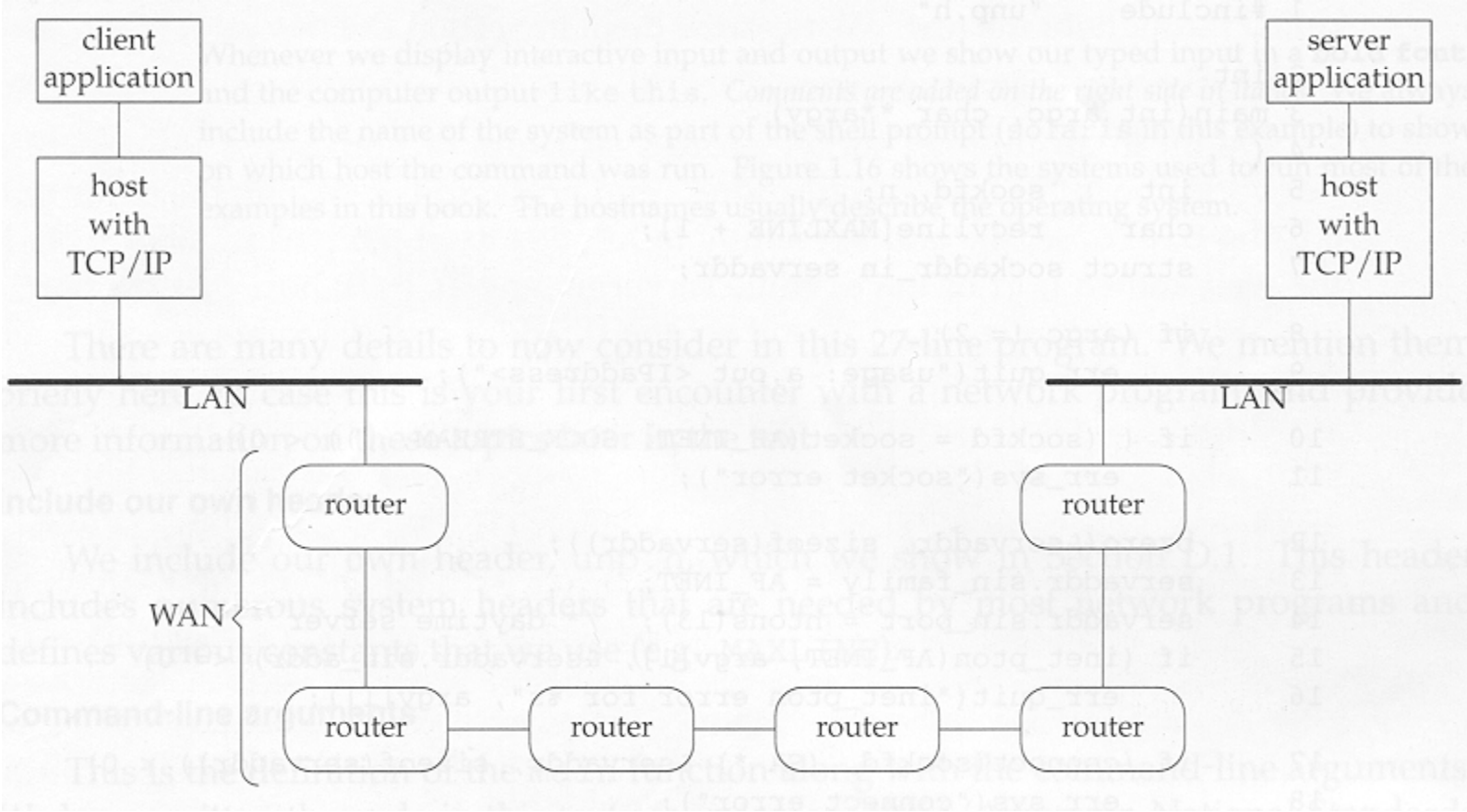
Μοντέλο client-server

- Ο βασικός τύπος δικτυακών εφαρμογών είναι *client - server*



- Η σχέση server και client μπορεί να είναι many-to-many.
  - Ένας server μπορεί να εξυπηρετεί ταυτόχρονα πολλούς clients.
  - Ένας client μπορεί να επιλέξει σε ποιους servers θέλει να συνδεθεί ταυτόχρονα.

Μοντέλο client-server

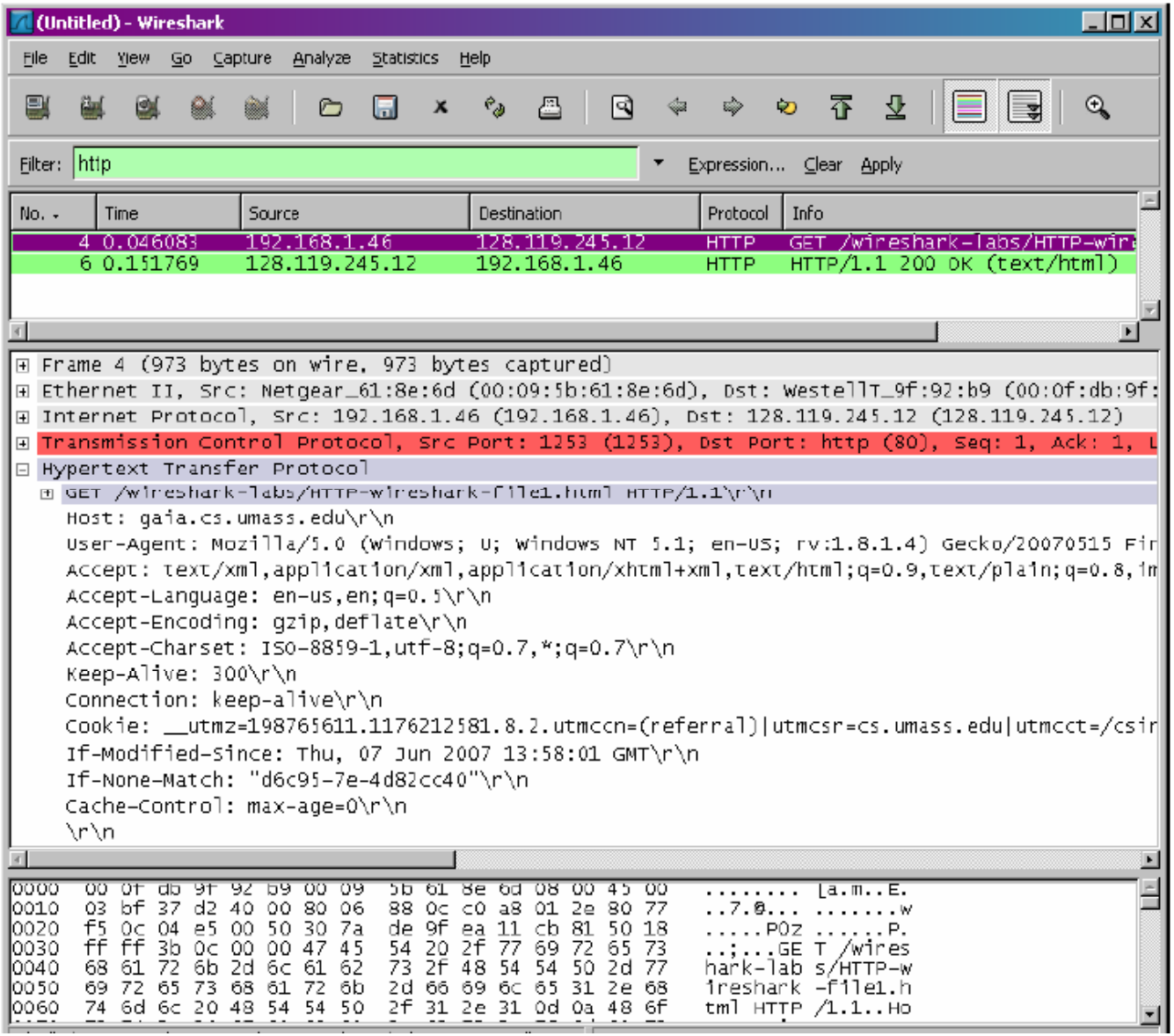


Παραδείγματα client/server

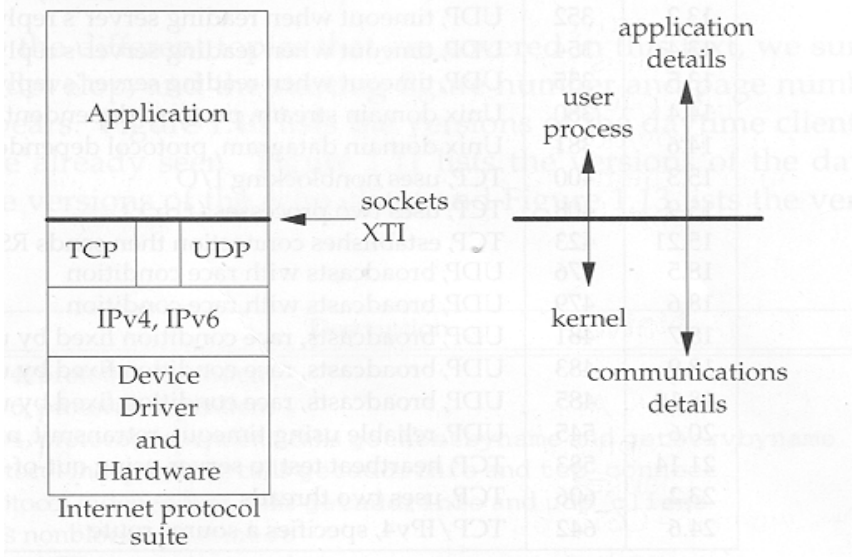
- HTTP - World Wide Web (WWW)
  - Client (web browser): Internet Explorer, mozilla firefox, opera …
  - Server: Apache, IIS (microsoft), …
- FTP
  - Client: CuteFTP, ακομα και οι web browser μπορούν να συνδεθούν σε FTP servers
  - Server: FileZila, CuteFTP, VSFTP
- DNS
  - Client: nslookup, dig
  - Server: BIND
- e-mail
  - Client: pine, outlook express, mozilla thunderbird,…
  - Server: Sendmail, qmail, MS Exchange
- Real audio, real video (streaming)
  - Client: Windows Media Player, Flash Player, QuickTime, Real Player
  - Server: Windows Media Services, Red5, Darwin Streaming Server, Helix
- Ακόμα και οι Peer-to-peer εφαρμογές (p2p) για να μεταφέρουν δεδομένα χρησιμοποιούν client/server τύπο

Στην πράξη….

- Παράδειγμα HTTP επικοινωνίας
- Wireshark (Πρώην Ethereal) packet capturing tool



Δικτυακός Προγραμματισμός



- Το Λ.Σ. έχει υλοποιημένη την στοίβα TCP/IP στον πυρήνα του
- Πως μπορούμε να προγραμματίσουμε πάνω από το TCP/IP?

⇨ Sockets API

Sockets in Unix

- **Socket API**
  (**A**pplication **P**rogramming **I**nterface)
- Εμφανίστηκε στο BSD4.1 UNIX, (1981)
- Έχει υιοθετηθεί από όλες τις παραλλαγές του Unix - Linux, και έχει μεταφερθεί και σε άλλα Λ.Σ.
- Σκοπός του Socket API είναι η γενική επικοινωνία μεταξύ διεργασιών (που τρέχουν στον ίδιο ή σε διαφορετικούς υπολογιστές)

➢ **Ο μηχανισμός των Sockets αποτελεί σήμερα το δημοφιλέστερο API προγραμματισμού και χρήσης της στοίβας πρωτοκόλλων TCP/IP**

socket

Μια διεπαφή: *τοπική, δημιουργημένη από μια διεργασία και ελεγχόμενη από το λειτουργικό σύστημα*.

Μια "πόρτα" μέσω της οποίας μια διεργασία μπορεί να στείλει και να δεχτεί μηνύματα σε/από κάποια άλλη διεργασία

Sockets, IP Addresses και Ports

- Η διεργασία που δημιουργεί το Socket το χειρίζεται μέσω ενός θετικού ακεραίου – socket descriptor (όπως ένα file descriptor)

- Για να επικοινωνήσουν οι άλλες διεργασίες χρειάζονται:
  - «Διεύθυνση». Το πεδίο από όπου λαμβάνει την «διεύθυνση» το socket αποτελεί το Domain ή Address family:
    - Unix Domain (AF_UNIX) (δεν θα το χρησιμοποιήσουμε)
    - Internet Domain (AF_INET). Κάθε socket στο Internet Domain ταυτοποιείται από την δυάδα:
      - IP Address του υπολογιστή
      - Port number - "θύρα" επικοινωνίας
  - Επίσης, οι άλλες διεργασίες πρέπει να ξέρουν το πρωτόκολλο μεταφοράς που χρησιμοποιείται (TCP ή UDP)

- **Κάθε κανάλι επικοινωνίας από άκρη-σε-άκρη (end-to-end connection) ταυτοποιείται μονοσήμαντα με την πεντάδα:**

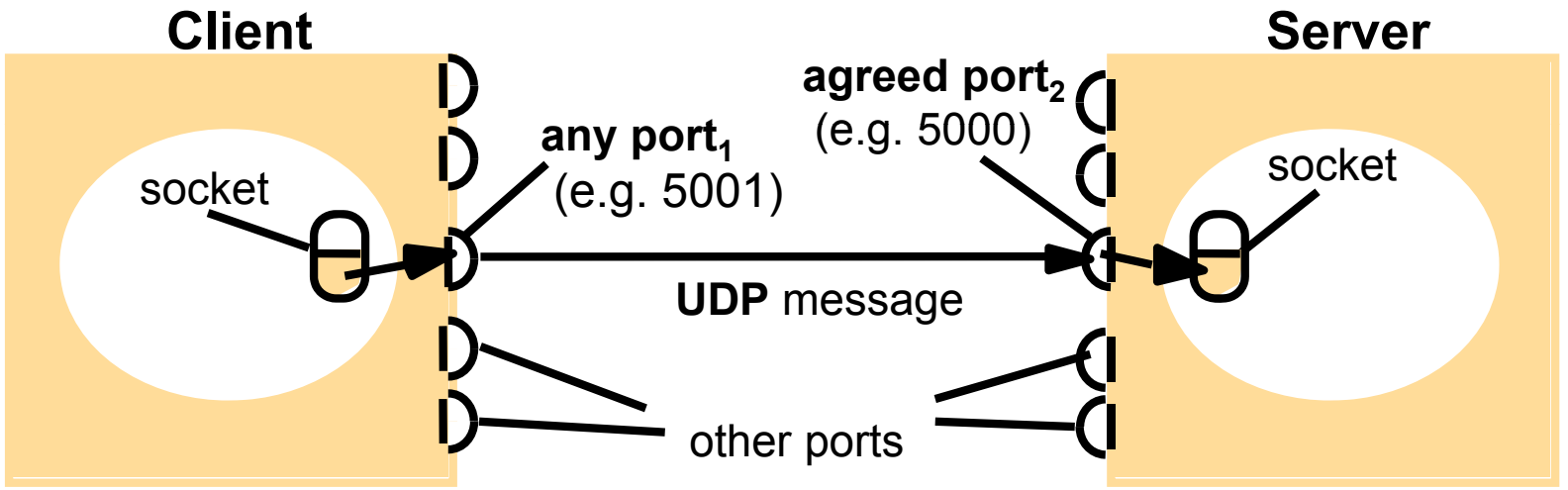
  { Protocol / IP address$_1$ / Port$_1$ / IP address$_2$ / Port$_2$ }

Stream / Datagram Communications

Η μετάδοση δεδομένων πάνω σε ένα κανάλι μπορεί να γίνει με 2 τρόπους:

1. Connection Oriented - Stream Communication (SOCK_STREAM)
   - Εγκατάσταση «κυκλώματος»
   - Το πρωτόκολλο επικοινωνίας είναι το TCP
     (παρέχει έλεγχο λαθών και επαναμεταδίδει τα χαμένα πακέτα)
   - Χρησιμοποιείται συνήθως σε εφαρμογές με καθορισμένη σχέση Client – Server
2. Connectionless - Datagram Communication (SOCK_DGRAM)
   - Το κανάλι δημιουργείται για κάθε μήνυμα ξεχωριστά
   - Το πρωτόκολλο επικοινωνίας είναι το UDP
     (Η παραλαβή των πακέτων δεν είναι εγγυημένη. Ο έλεγχος μετάδοσης, και τα fragmentation/reassembly των δεδομένων γίνονται στο επίπεδο εφαρμογής)
   - Χρησιμοποιείται συνήθως σε εφαρμογές που απαιτούν υψηλές ταχύτητες μετάδοσης με μικρά αυτόνομα “πακέτα”, χωρίς εγγυημένη αξιοπιστία

Sockets, IP addresses και Ports (Client/Server Example)

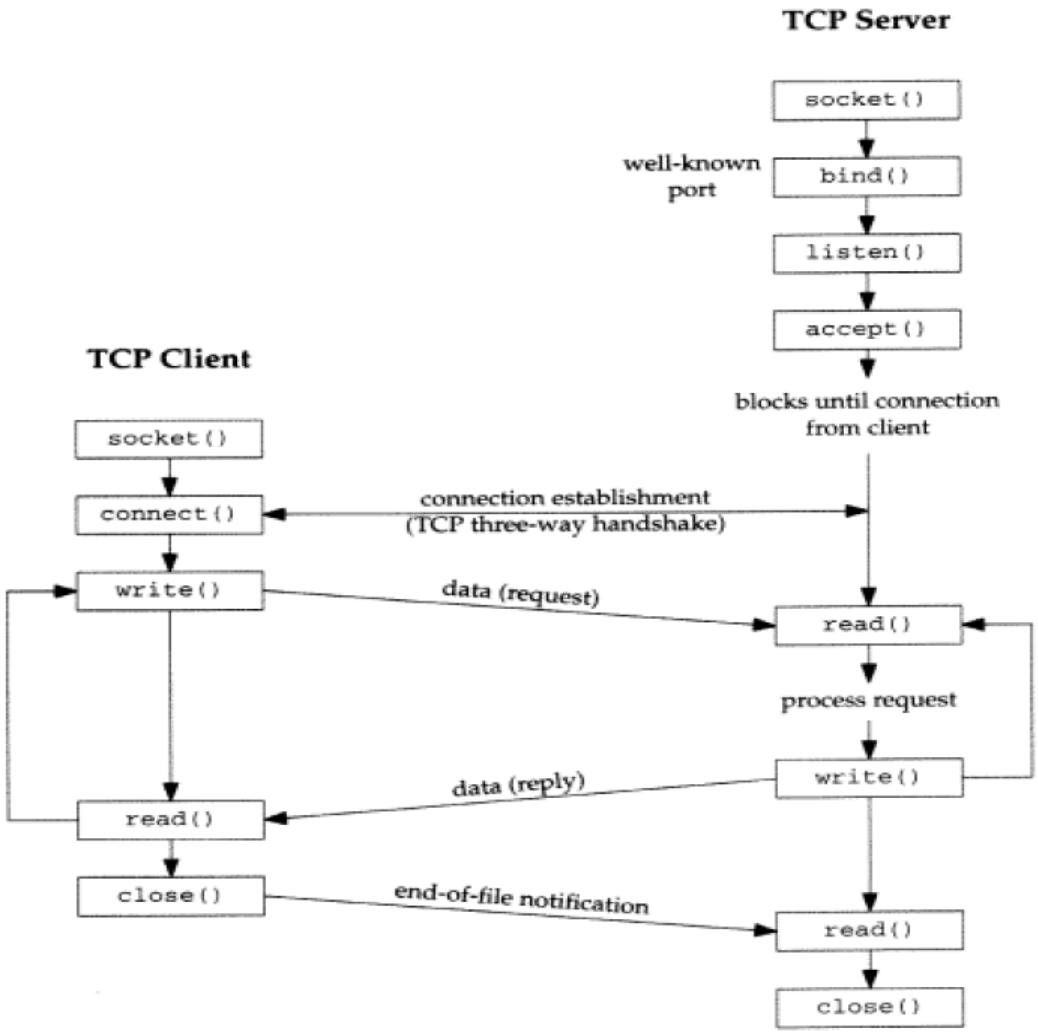


Η επικοινωνία από άκρη-σε-άκρη (end-to-end connection) ταυτοποιείται μονοσήμαντα με την πεντάδα:

{ Protocol / IP address1 / Port1 / IP address2 / Port2 } =

{ UDP / 150.140.141.135 / 5001 / 150.140.141.134 / 5000 }

TCP Client/Server interaction (Stream Communication)



TCP Sockets - Πρωτογενείς κλήσεις

- **Server**
  - create endpoint (`socket()`)
  - bind address (`bind ()`)
  - specify queue (`listen()`)
  - wait for connection (`accept()`)
  - transfer data (`read() - write()`)
- **Client**
  - create endpoint (`socket()`)
  - connect to server (`connect()`)
  - transfer data (`read() - write()`)

Πρωτογενείς κλήσεις για τα sockets

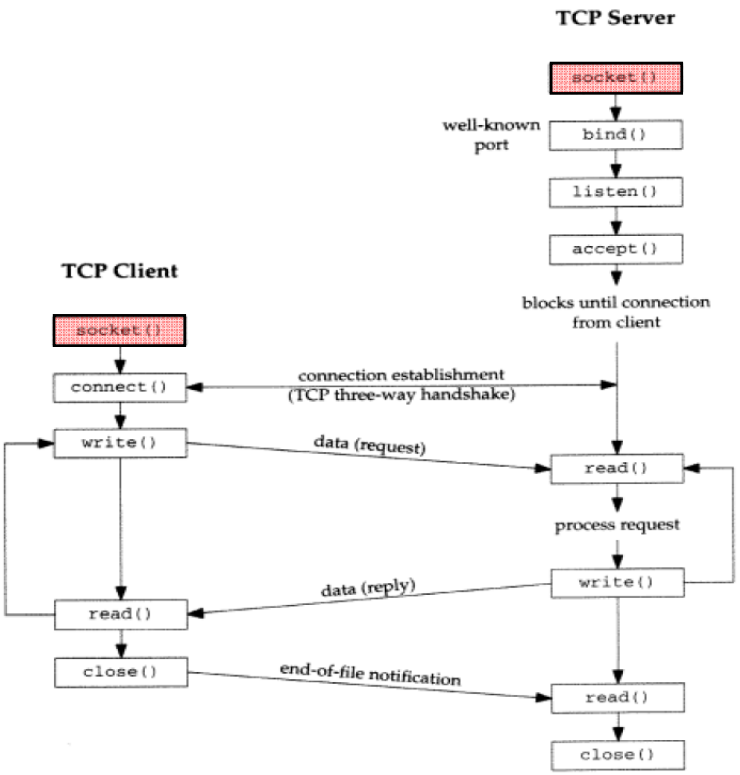
Οι οδηγίες (#include) που χρησιμοποιούνται για αυτές τις κλήσεις είναι οι :

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

Η κλήση “socket”

Αρχικοποίηση ενός socket

- Σύνταξη
  `int socket (int family, int type, int protocol)`
- Επιστρέφει ένα ακέραιο ανάλογο με αυτό που επιστρέφουν οι ρουτίνες διαχείρισης αρχείων (fd) **sockfd**



Η κλήση “socket”

family	Description
AF_INET	IPv4 protocols
AF_INET6	IPv6 protocols
AF_LOCAL	Unix domain protocols (Chapter 14)
AF_ROUTE	Routing sockets (Chapter 17)
AF_KEY	Key socket

Protocol *family* constants for `socket` function.

type	Description
SOCK_STREAM	stream socket
SOCK_DGRAM	datagram socket
SOCK_RAW	raw socket

type of socket for `socket` function.

	AF_INET	AF_INET6	AF_LOCAL	AF_ROUTE	AF_KEY
SOCK_STREAM	TCP	TCP	Yes		
SOCK_DGRAM	UDP	UDP	Yes		
SOCK_RAW	IPv4	IPv6		Yes	Yes

Combinations of *family* and *type* for the `socket` function.

Example:

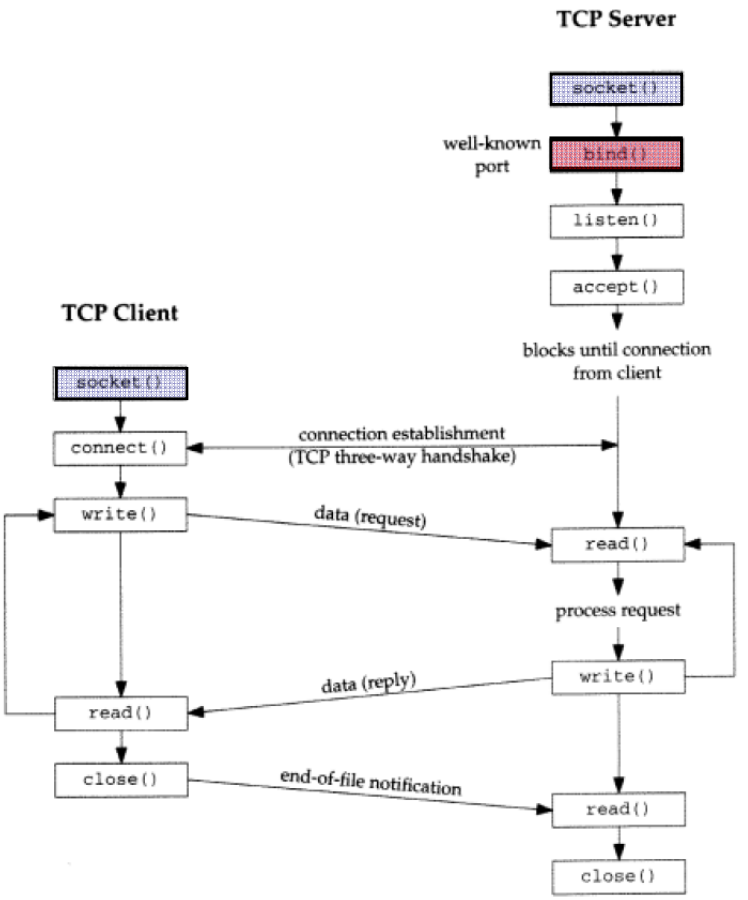
int **sockfd;**

sockfd = socket (AF_INET, SOCK_STREAM, 0);

Η κλήση “bind” – Ονομασία του Socket

Στην αρχική φάση το socket δεν έχει διεύθυνση

- Σύνταξη:
  `int bind (int sockfd, struct sockaddr * myaddr, int addrlen)`
- Συνδέει το **sockfd** που έχει επιστραφεί από την κλήση socket(), με μία τοπική διεύθυνση και θύρα (IP address, port number), γνωστοποιώντας στο σύστημα ότι τα μηνύματα που έρχονται στα συγκεκριμένα: (interface – port) απευθύνονται στη συγκεκριμένη διαδικασία.



Socket Address Structures

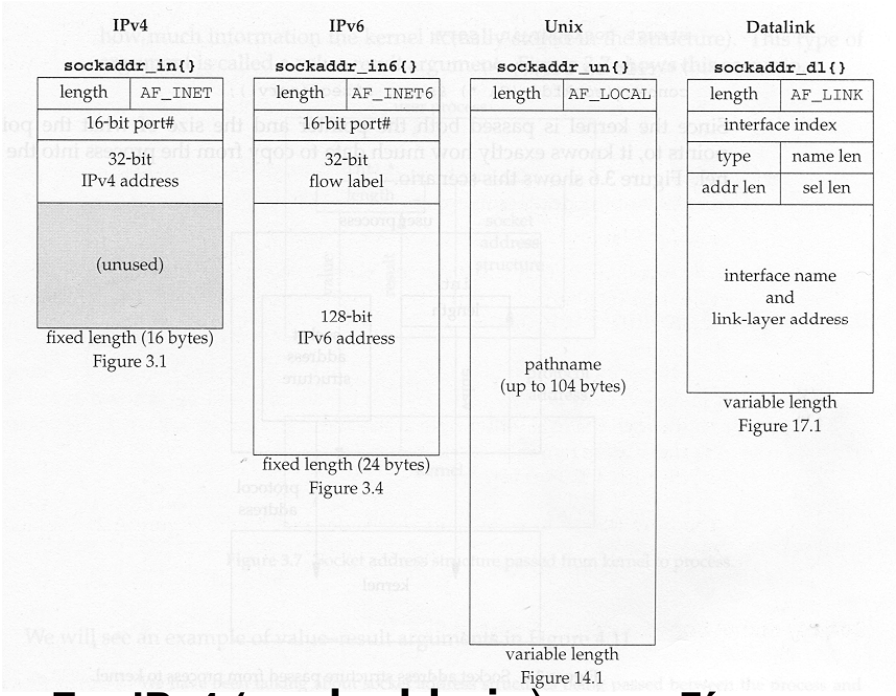
```
int bind (int sockfd, struct sockaddr * myaddr, int addrlen)
```

- Σκοπός των δομών Socket Address είναι να χρησιμοποιούνται από τις συναρτήσεις του socket API
- Είναι δομές που ξεκινούν με το χαρακτηριστικό `sockaddr_` το οποίο παίρνει κατάληξη ανάλογα με το πρωτόκολλο δικτύου.
  - IPv4: `sockaddr_in <netinet/in.h>`
  - IPv6: `sockaddr_in6`
- Υπάρχουν βοηθητικές συναρτήσεις για την μετατροπή σε `sockaddr_` δομές

  π.χ. `inet_addr`, `inet_ntoa`, `inet_pton`, `inet_ntop`
- Generic `sockaddr_` Structure

```
#include <sys/socket.h>
   struct sockaddr {
     uint8_t          sa_len;
     sa_family_t      sa_family;
     char             sa_data [14];
   };
```

Σύγκριση Socket Address Structures



IPv4

```
struct in_addr {
  in_addr_t   s_addr;          /* 32-bit IPv4 address */
                               /* network byte ordered */
};

struct sockaddr_in {
  uint8_t          sin_len;    /* length of structure (16) */
  sa_family_t      sin_family; /* AF_INET */
  in_port_t        sin_port;   /* 16-bit TCP or UDP port number */
                               /* network byte ordered */
  struct in_addr   sin_addr;   /* 32-bit IPv4 address */
                               /* network byte ordered */
  char             sin_zero[8]; /* unused */
};
```

Στο IPv4 (sockaddr_in), τα πεδία που μας ενδιαφέρουν είναι τα

sin_family: ίδιο με το Address family στην δημιουργία του socket (AF_INET)

sin_port: καθορίζει το port number

Δεν πρέπει να χρησιμοποιείται από κάποιο άλλο socket.
Αριθμοί από 1 εώς 1023 είναι δεμευμένοι για τις υπερεσίες του συστήματος. Συνήθως χρησιμοποιούνται αριθμοί πάνω από 5000.

sin_addr: μιά IP διέυθυνση του υπολογιστή (e.g. 150.140.141.134)

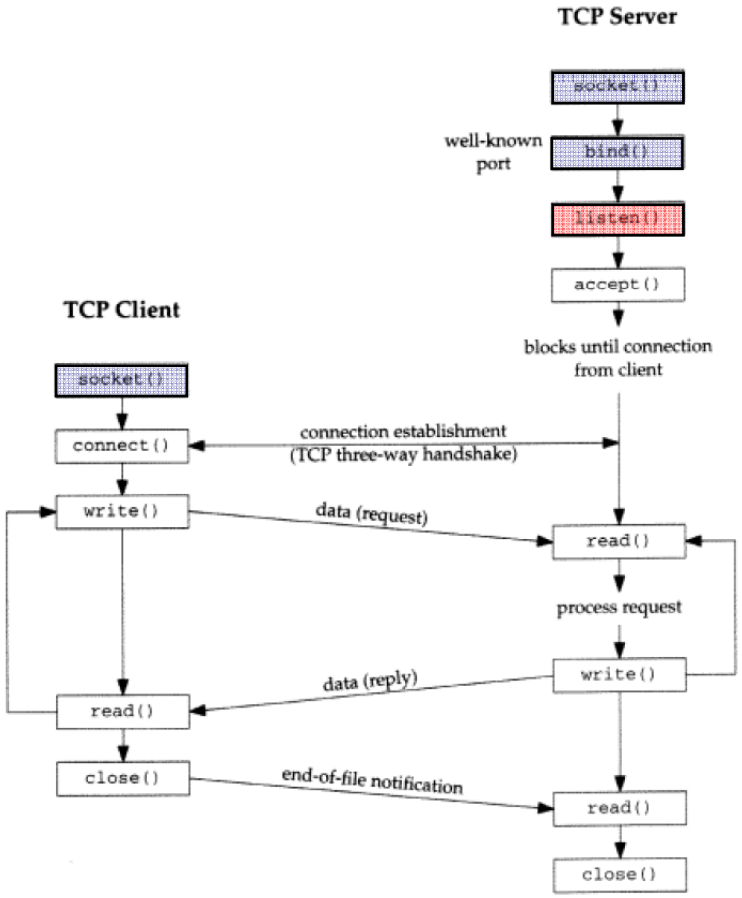
Server example (so far)

Example:

```
int sockfd;
sockfd = socket (AF_INET, SOCK_STREAM, 0);

struct sockaddr_in sin;                          /* socket address structure*/

sin.sin_family      = AF_INET;
sin.sin_port        = htons(5000);                        /* agreed port*/
sin.sin_addr.s_addr = htonl(INADDR_ANY);  /* accept connections to all
                                             computer interfaces */

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

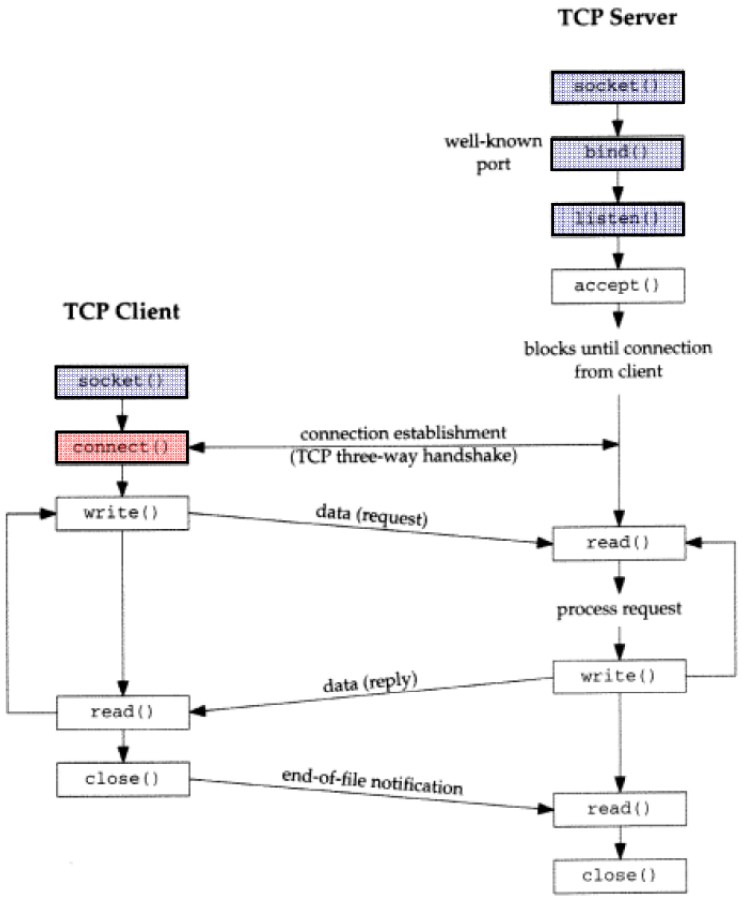
Η κλήση “listen” (Προαιρετική)

- Σύνταξη:
  `int listen (int sockfd, int backlog)`
- Χρησιμοποιείται από ένα connection-oriented εξυπηρετητή για να ενημερωθεί το σύστημα ότι ο (εξυπηρετητής) είναι έτοιμος να λάβει μηνύματα στο socket με descriptor το **sockfd**



Η κλήση “connect”

- Σύνταξη:
  `int connect (int sockfd_cl, struct sockaddr * servaddr, int addrlen)`
- Συνδέει τον descriptor **sockfd_cl** που έχει επιστραφεί με την κλήση socket() στον client, με μία IP address και έναν αριθμό θύρας (port) του server (μπορεί να βρίσκεται στο ίδιο ή σε κάποιο άλλο μηχάνημα)

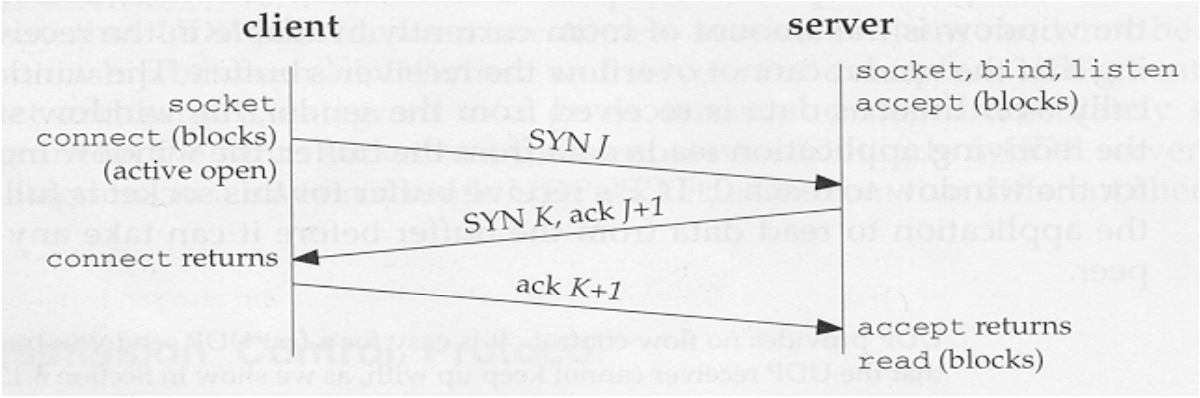


Client Example (so far)

```
int sockfd_cl;
sockfd_cl = socket (AF_INET, SOCK_STREAM, 0);

struct sockaddr_in server_addr;

server_addr.sin_family             = AF_INET;
server_addr.sin_port               = 5000;             /* agreed port*/
server_addr.sin_addr.s_addr =
        inet_addr("150.140.141.134");    /* server IP address */

connect (sockfd_cl, server_addr, sizeof(server_addr) );
```

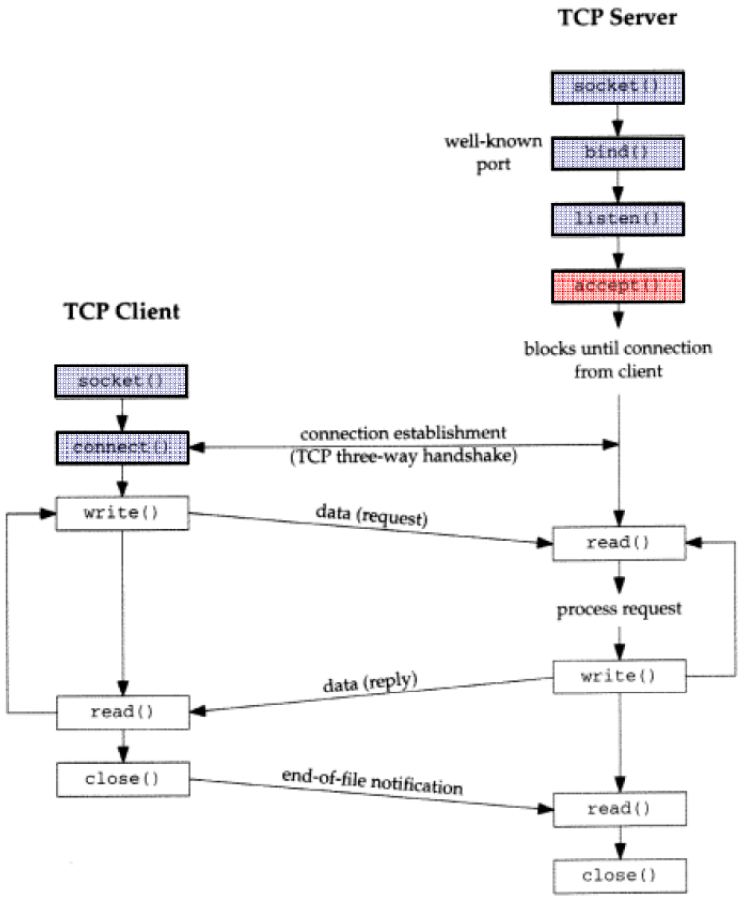
… Η κλήση “connect”

- Τα TCP sockets αρχικοποιούνται με «three-way handshaking»
- Γίνονται αυτόματα από το Socket API



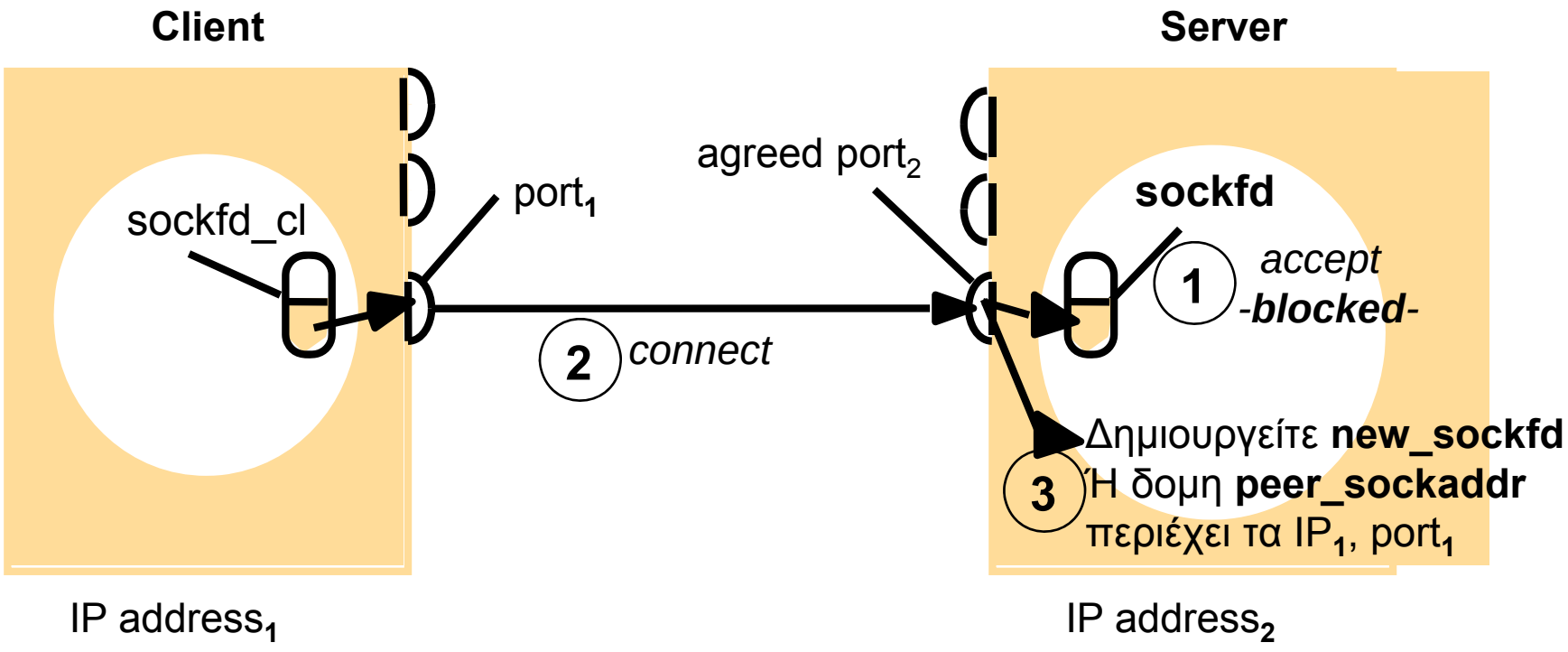
Η κλήση “accept”

- Σύνταξη:
  `int accept (sockfd, struct sockaddr *peer_sockaddr, int * addrlen)`
- Με την κλήση αυτή γίνεται αποδοχή κάποιας αίτησης σύνδεσης που περιμένει στην ουρά του **sockfd**
- Αν η ουρά είναι άδεια, η διεργασία **μπλοκάρει** μέχρι να εμφανιστεί κάποια αίτηση
- Μολις γίνει αποδοχή, η παράμετρος **peer_sockaddr** επιστρέφει πληροφορίες (port, IP address) για τον client
- Επιστρέφεται ένας νέος socket descriptor (**new_sockfd**), ο οποίος αποτελεί το άκρο ενός καινούργιου καναλιού (client – server channel)



… Η κλήση “accept”

```
new_sockfd = accept (sockfd, &peer_sockaddr, addrlen)
```



… Η κλήση “accept”

- Απο την στιγμή που η accept() επιστρέφει το νέο socket, το κανάλι επικοινωνίας με τον client έχει εγκατασταθεί
- Μέχρι να ξανακληθεί η accept(), ο server δεν μπορεί να δεχτεί άλλες κλήσεις για σύνδεση
- Οι κλήσεις που γίνονται όσο ο server βρίσκεται εκτός της accept() τοποθετούνται σε μια ουρά (default size 5 – μπορεί να αλλάξει με την κλήση listen())
- Ο συνήθης τρόπος επεξεργασίας της αίτησης από τον server είναι με fork()
- Με την fork() ο client επικοινωνεί με έναν «αφοσιωμένο» αντίγραφο του server, ενώ ο «αρχικός» server μπορεί να δέχεται νέες κλήσεις σύνδεσης (επιστρέφει στην accept())

... Server example (so far)

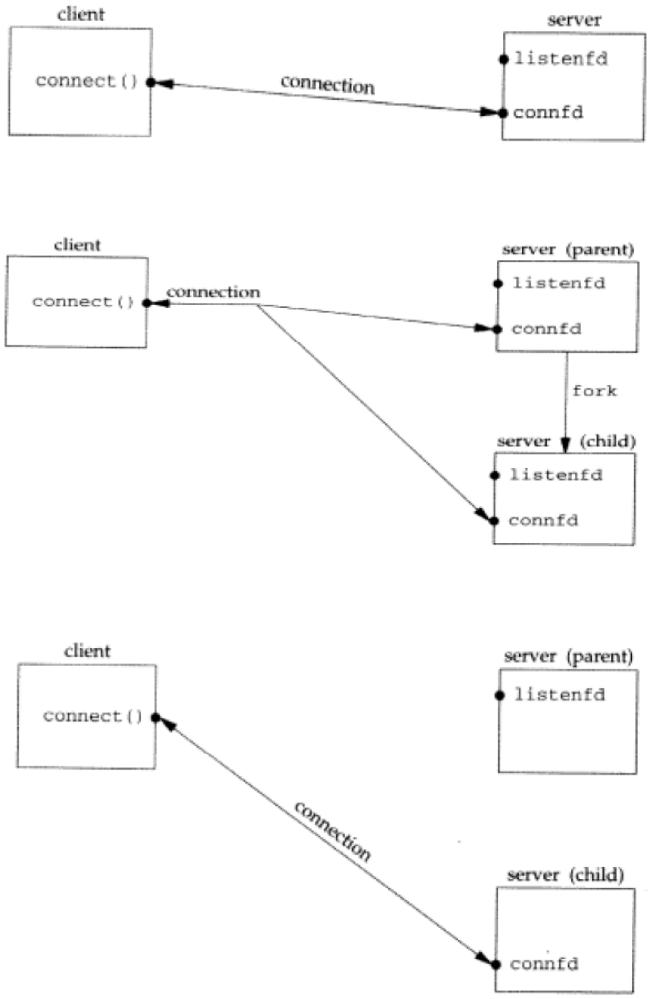
```
int sockfd;
sockfd = socket (AF_INET, SOCK_STREAM, 0);

struct sockaddr_in sin;

sin.sin_family          = AF_INET;
sin.sin_port            = htons(5000);          /* agreed port*/
sin.sin_addr.s_addr     = htonl(INADDR_ANY);    /* accept connections to all
                                                   computer interfaces */

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

listen(sockfd, LISTENQ);

Struct sockaddr_in peer_sockaddr;               /* client socket address structure*/
int new_sockfd;                                 /* new socket id */

new_sockfd = accept (sockfd, &peer_sockaddr, sizeof (peer_sockaddr) );
```

Η κλήση “fork”

- Σύνταξη:

```
#include <unistd.h>
int pid_t fork();
```

- Επιστρέφει 0 στο «παιδί»
- Το «παιδί» μπορεί να βρει τον «πατέρα» του καλώντας την getppid()
- Ο «πατέρας» μπορεί να έχει πολλά παιδιά

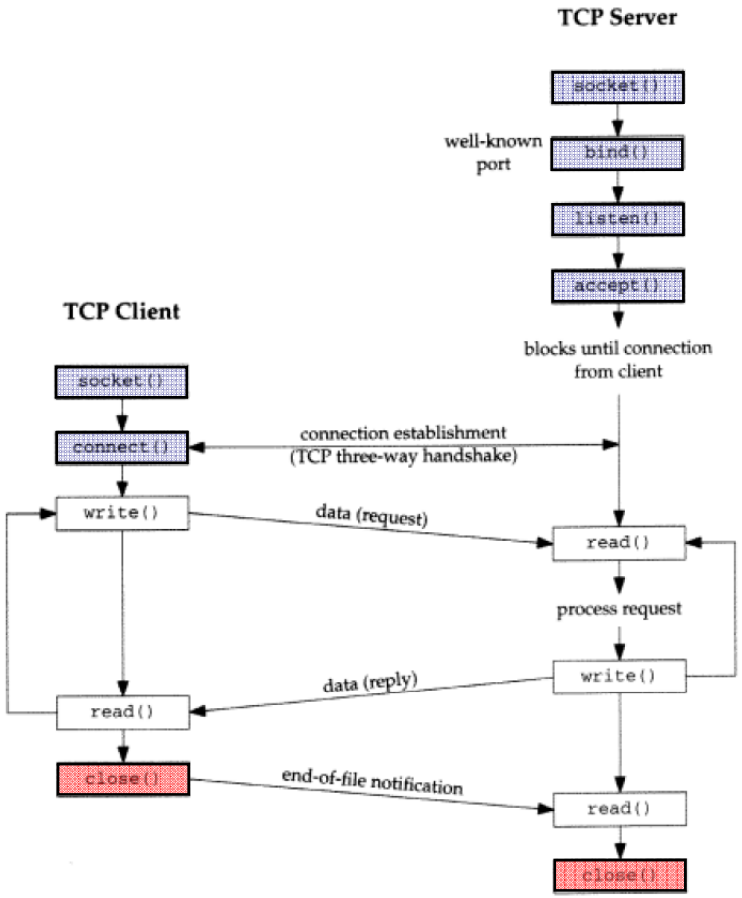


... Server example (so far)

```
int sockfd;
sockfd = socket (AF_INET, SOCK_STREAM, 0);

struct sockaddr_in sin;

sin.sin_family= AF_INET;
sin.sin_port   = htons(5000);                          /* agreed port*/
sin.sin_addr.s_addr= htonl(INADDR_ANY);                /* accept connections to all
                                                       computer interfaces */
bind (sockfd, &sin, sizeof (sin) );

listen(sockfd, LISTENQ);

for ( ; ; ;) {

      Struct sockaddr_in peer_sockaddr;                /* client socket address structure*/
      int new_sockfd;                                  /* new socket id */
      new_sockfd = accept (sockfd, &peer_sockaddr, sizeof (peer_sockaddr) );

      int childid;
      if (  (childid = fork() ) == 0 ) {               /* child process */
                ….                                     /* devoted server */
                                                       /* use new_sockfd */


      }
      …..                                              /* main-parent server continue*/
}
```

Η κλήση “close”

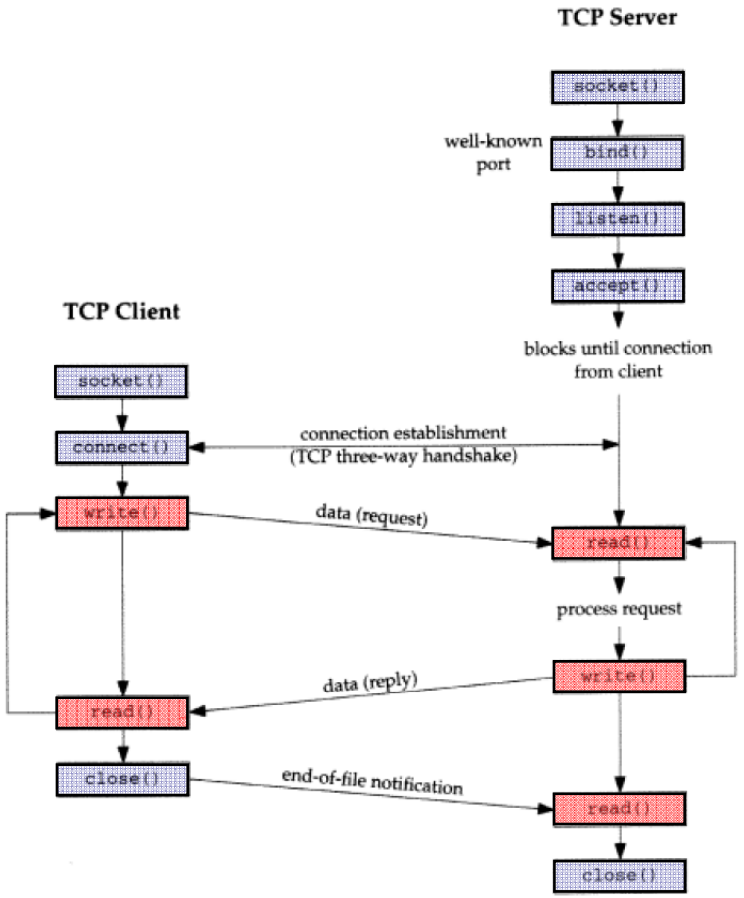
- Σύνταξη:
  `int close (int sockfd)`
- Κλείνει το socket που είχε δημιουργηθεί με την αντίστοιχη κλήση socket()



Επικοινωνία, Είσοδος και Έξοδος

Ο socket descriptor έχει την ίδια λειτουργικότητα με έναν file descriptor

- Μπορούν να χρησιμοποιηθούν read() και write()
- Όμως τα read/write μπορεί να μην γράψουν/διαβάσουν όσους χαρακτήρες τους ζητηθούν
- Για το λόγο αυτό, οι κλήσεις read/write πρέπει να «επιμένουν»



Συναρτήσεις readn, writen, readline

Υπάρχουν ειδικές συναρτήσεις που «επιμένουν» ώστε να γραφτούν και να διαβαστούν όσοι χαρακτήρες τους ζητηθούν

```
#include "unp.h"

ssize_t readn(int filedes, void *buff, size_t nbytes);

ssize_t writen(int filedes, const void *buff, size_t nbytes);

ssize_t readline(int filedes, void *buff, size_t maxlen);
```

All return: number of bytes read or written, –1 on error

readn

lib/readn.c

```
 1 #include    "unp.h"

 2 ssize_t                       /* Read "n" bytes from a descriptor. */
 3 readn(int fd, void *vptr, size_t n)
 4 {
 5     size_t  nleft;
 6     ssize_t nread;
 7     char    *ptr;

 8     ptr = vptr;
 9     nleft = n;
10     while (nleft > 0) {
11         if ( (nread = read(fd, ptr, nleft)) < 0) {
12             if (errno == EINTR)
13                 nread = 0;      /* and call read() again */
14             else
15                 return (-1);
16         } else if (nread == 0)
17             break;              /* EOF */
18         nleft -= nread;
19         ptr += nread;
20     }
21     return (n - nleft);         /* return >= 0 */
22 }
```

lib/readn.c

writen

lib/writen.c

```
 1 #include    "unp.h"

 2 ssize_t                       /* Write "n" bytes to a descriptor. */
 3 writen(int fd, const void *vptr, size_t n)
 4 {
 5     size_t  nleft;
 6     ssize_t nwritten;
 7     const char *ptr;

 8     ptr = vptr;
 9     nleft = n;
10     while (nleft > 0) {
11         if ( (nwritten = write(fd, ptr, nleft)) <= 0) {
12             if (errno == EINTR)
13                 nwritten = 0;   /* and call write() again */
14             else
15                 return (-1);    /* error */
16         }
17         nleft -= nwritten;
18         ptr += nwritten;
19     }
20     return (n);
21 }
```

lib/writen.c

readline

test/readline1.c

```
 1 #include    "unp.h"

 2 ssize_t
 3 readline(int fd, void *vptr, size_t maxlen)
 4 {
 5     ssize_t n, rc;
 6     char    c, *ptr;

 7     ptr = vptr;
 8     for (n = 1; n < maxlen; n++) {
 9       again:
10         if ( (rc = read(fd, &c, 1)) == 1) {
11             *ptr++ = c;
12             if (c == '\n')
13                 break;              /* newline is stored, like fgets() */
14         } else if (rc == 0) {
15             if (n == 1)
16                 return (0);         /* EOF, no data read */
17             else
18                 break;              /* EOF, some data was read */
19         } else {
20             if (errno == EINTR)
21                 goto again;
22             return (-1);            /* error, errno set by read() */
23         }
24     }

25     *ptr = 0;                       /* null terminate like fgets() */
26     return (n);
27 }
```

test/readline1.c

Μετατροπή διευθύνσεων?

Presentation (Internet standard dot notation a.b.c.d) ↔ Network address (struct in_addr), IPv4 only

```
#include <arpa/inet.h>
int inet_aton(const char *strptr, struct in_addr *addrptr);
```

Returns: 1 if string was valid, 0 on error

```
in_addr_t inet_addr(const char *strptr);
```

Returns: 32-bit binary network byte ordered IPv4 address; INADDR_NONE if error

```
char *inet_ntoa(struct in_addr inaddr);
```

Returns: pointer to dotted-decimal string

Network address (struct in_addr) ↔ Presentation (Internet standard dot notation a.b.c.d, or Classless Internet Domain Routing a.b.c.d/N), IPv4 and IPv6

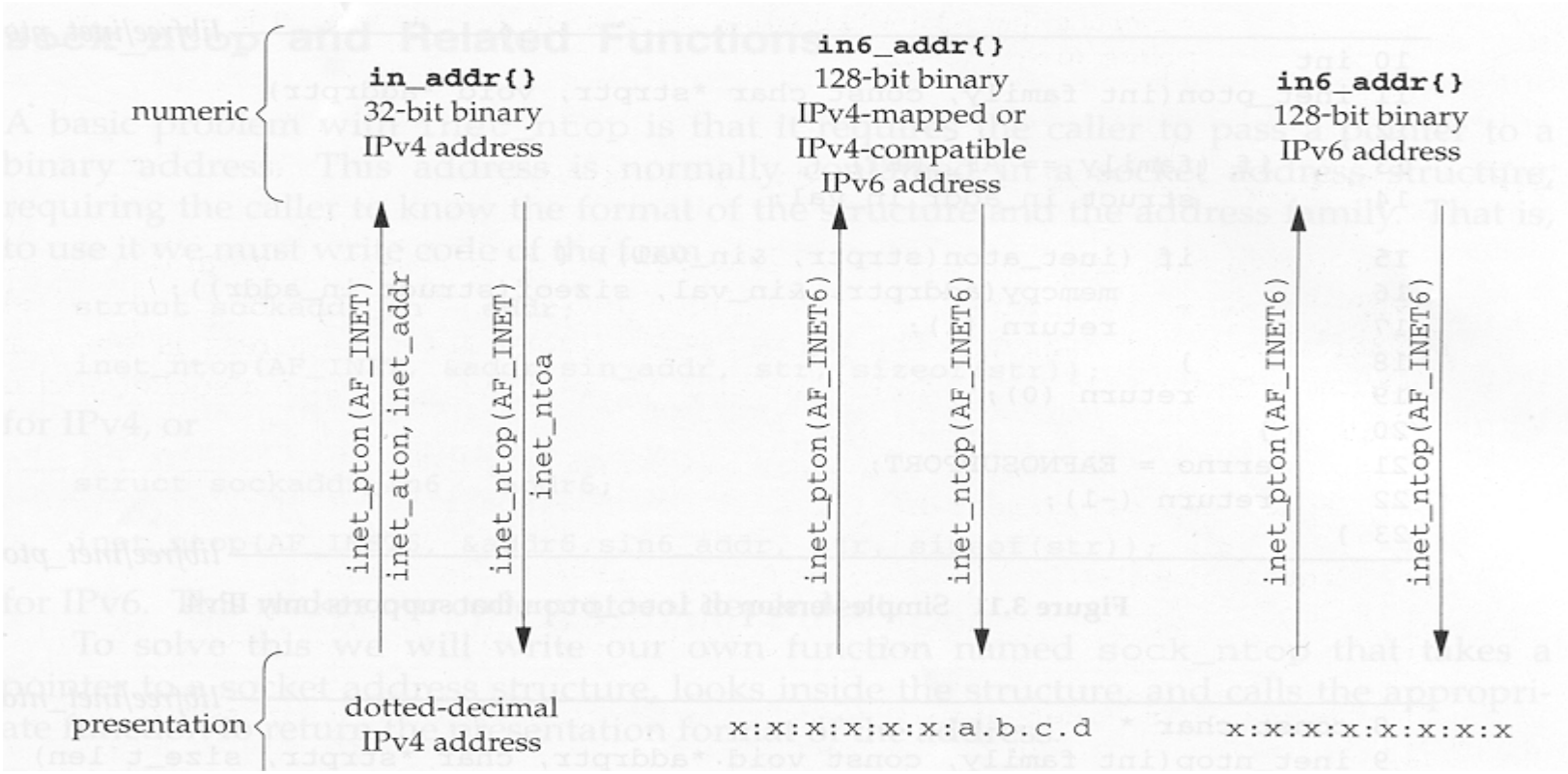
```
#include  <arpa/inet.h>
int inet_pton(int family, const char *strptr, void *addrptr);
```

Returns: 1 if OK, 0 if input not a valid presentation format, −1 on error

```
const char *inet_ntop(int family, const void *addrptr, char *strptr, size_t len);
```

Returns: pointer to result if OK, NULL on error

Summary of Address Conversion Functions



Πως κάνουμε resolve ένα όνομα ?

- Δεν μπορούμε να θυμόμαστε την IP address του server
- Χρησιμοποιούμε name aliases (e.g diogenis.ceid.upatras.gr) και την κλήση gethostbyname
- Αντίστροφα: gethostbyaddr

```
#include <netdb.h>

struct hostent *gethostbyname(const char *hostname);
```

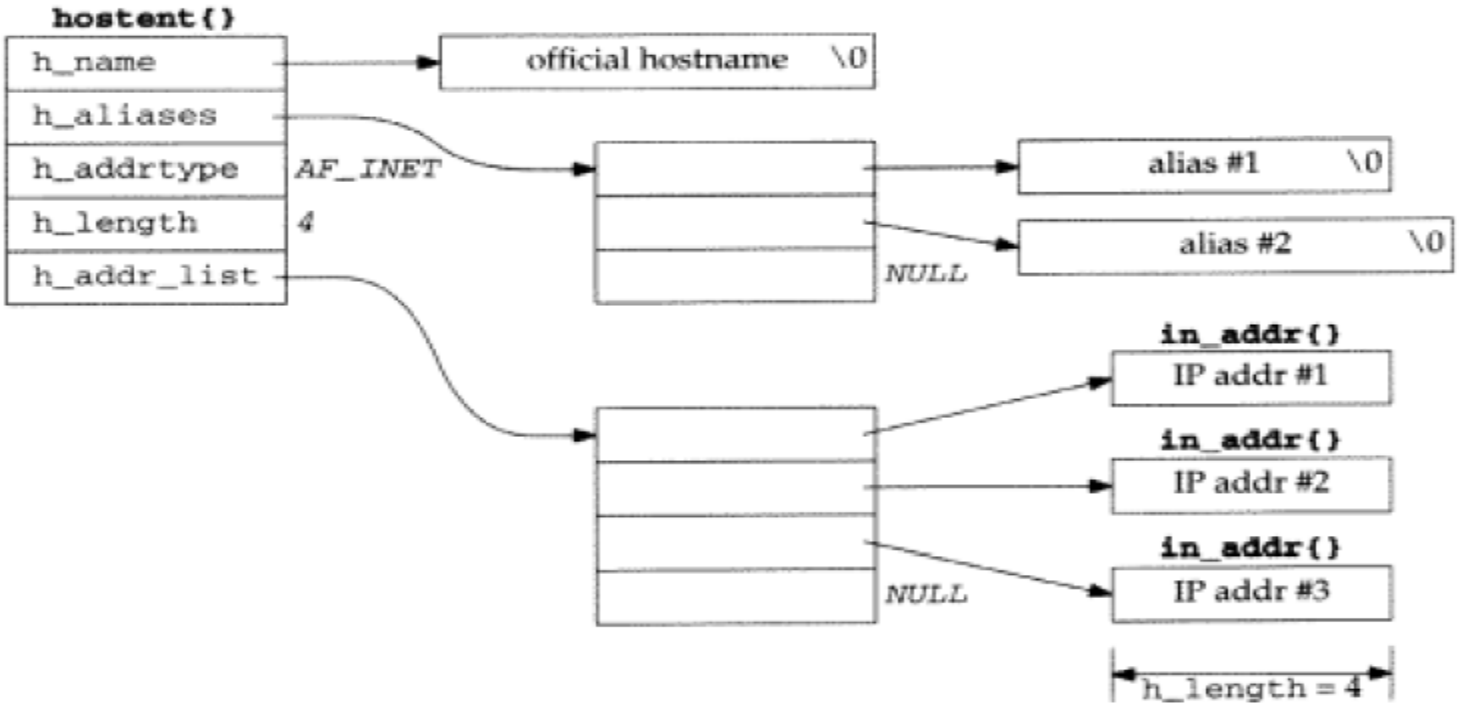
Returns: nonnull pointer if OK, NULL on error with h_errno set

```
#include <netdb.h>

struct hostent *gethostbyaddr(const char *addr, size_t len, int family);
```

Returns: nonnull pointer if OK, NULL on error with h_errno set

```
struct hostent {
  char  *h_name;       /* official (canonical) name of host */
  char **h_aliases;    /* pointer to array of pointers to alias names */
  int    h_addrtype;   /* host address type: AF_INET or AF_INET6 */
  int    h_length;     /* length of address: 4 or 16 */
  char **h_addr_list;  /* ptr to array of ptrs with IPv4 or IPv6 addrs */
};
```

hostent structure



Πως βρίσκουμε το port μιας υπηρεσίας ?

- Δεν μπορούμε να θυμόμαστε τα port numbers της υπηρεσίας
- Χρησιμοποιούμε το όνομα (e.g FTP) και την κλήση getservbyname
- Αντίστροφα: getservbyport

```
#include <netdb.h>

struct servent *getservbyname(const char *servname, const char *protoname);
```

Returns: nonnull pointer if OK, NULL on error

```
#include <netdb.h>

struct servent *getservbyport(int port, const char *protname);
```

Returns: nonnull pointer if OK, NULL on error

```
struct servent {
  char   *s_name;      /* official service name */
  char  **s_aliases;   /* alias list */
  int     s_port;      /* port number, network-byte order */
  char   *s_proto;     /* protocol to use */
};
```

Ένας απλός Server

tcpcliserv/tcpserv01.c

```
 1 #include    "unp.h"
 2 int
 3 main(int argc, char **argv)
 4 {
 5     int     listenfd, connfd;
 6     pid_t   childpid;
 7     socklen_t clilen;
 8     struct sockaddr_in cliaddr, servaddr;
 9     listenfd = Socket(AF_INET, SOCK_STREAM, 0);
10     bzero(&servaddr, sizeof(servaddr));
11     servaddr.sin_family = AF_INET;
12     servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
13     servaddr.sin_port = htons(SERV_PORT);
14     Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
15     Listen(listenfd, LISTENQ);
16     for ( ; ; ) {
17         clilen = sizeof(cliaddr);
18         connfd = Accept(listenfd, (SA *) &cliaddr, &clilen);
19         if ( (childpid = Fork()) == 0) {     /* child process */
20             Close(listenfd);    /* close listening socket */
21             str_echo(connfd);   /* process the request */
22             exit(0);
23         }
24         Close(connfd);          /* parent closes connected socket */
25     }
26 }
```

tcpcliserv/tcpserv01.c

… Ένας απλός Server

lib/str_echo.c

```
 1 #include    "unp.h"
 2 void
 3 str_echo(int sockfd)
 4 {
 5     ssize_t n;
 6     char    line[MAXLINE];
 7     for ( ; ; ) {
 8         if ( (n = Readline(sockfd, line, MAXLINE)) == 0)
 9             return;              /* connection closed by other end */
10         Writen(sockfd, line, n);
11     }
12 }
```

lib/str_echo.c

Ένας απλός Client

tcpcliserv/tcpcli01.c

```
 1 #include    "unp.h"
 2 int
 3 main(int argc, char **argv)
 4 {
 5     int     sockfd;
 6     struct sockaddr_in servaddr;
 7     if (argc != 2)
 8         err_quit("usage: tcpcli <IPaddress>");
 9     sockfd = Socket(AF_INET, SOCK_STREAM, 0);
10     bzero(&servaddr, sizeof(servaddr));
11     servaddr.sin_family = AF_INET;
12     servaddr.sin_port = htons(SERV_PORT);
13     Inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
14     Connect(sockfd, (SA *) &servaddr, sizeof(servaddr));
15     str_cli(stdin, sockfd);      /* do it all */
16     exit(0);
17 }
```

tcpcliserv/tcpcli01.c

… Ένας απλός Client

lib/str_cli.c

```
 1 #include    "unp.h"

 2 void
 3 str_cli(FILE *fp, int sockfd)
 4 {
 5     char    sendline[MAXLINE], recvline[MAXLINE];

 6     while (Fgets(sendline, MAXLINE, fp) != NULL) {

 7         Writen(sockfd, sendline, strlen(sendline));

 8         if (Readline(sockfd, recvline, MAXLINE) == 0)
 9             err_quit("str_cli: server terminated prematurely");

10         Fputs(recvline, stdout);
11     }
12 }
```

lib/str_cli.c