Δικτυακός προγραμματισμός

(hands on)

Εργαστήριο Δικτύων Υπολογιστών 2014-2015

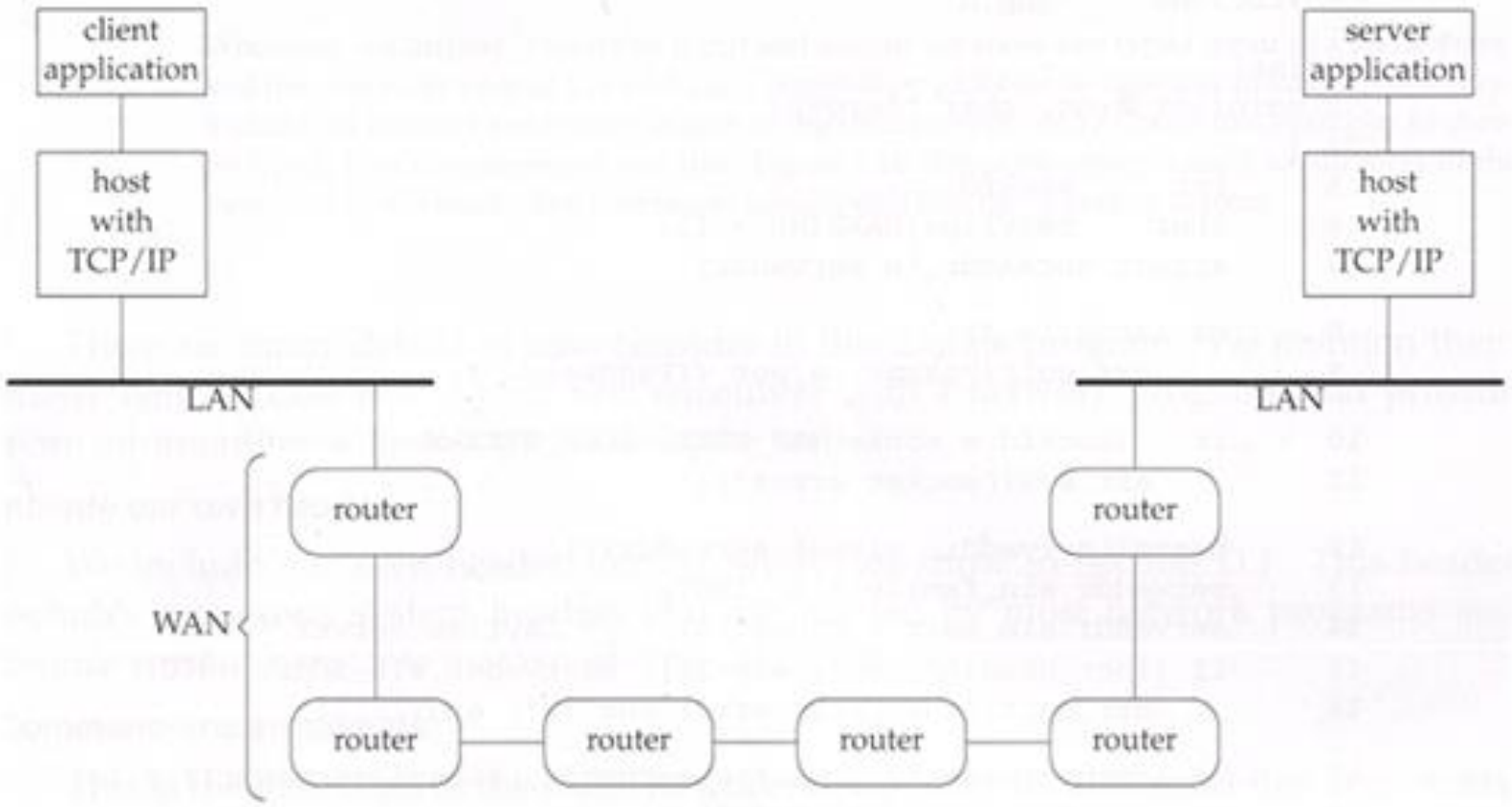
Τμήμα Μηχανικών Η/Υ και Πληροφορικής

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

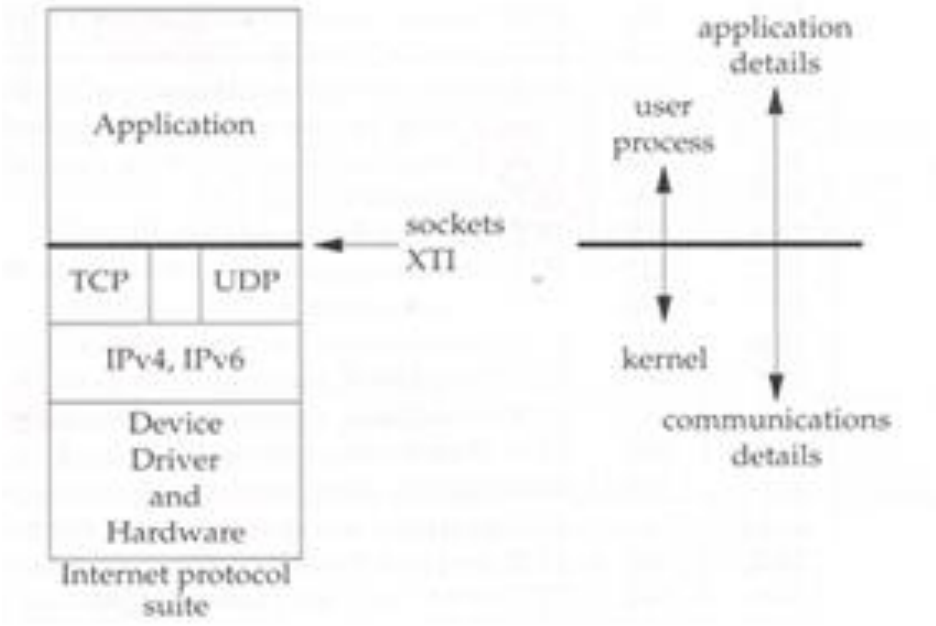
Μοντέλο client-server



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
  - 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

Δικτυακός Προγραμματισμός



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

Δικτυακός Προγραμματισμός

- Το Socket API είναι ένα Application Programming Interface (API), συνήθως, παρέχεται από το λειτουργικό σύστημα, που επιτρέπει σε προγράμματα εφαρμογής να ελέγχουν και να χρησιμοποιούν Network Sockets.
- Σκοπός του Socket API είναι η γενική επικοινωνία μεταξύ διεργασιών (που τρέχουν στον ίδιο ή σε διαφορετικούς υπολογιστές)
- Ένα network socket είναι το ένα άκρο μιας επικοινωνίας μεταξύ δύο διεργασιών πάνω από ένα δίκτυο υπολογιστών
- Socket APIs συνήθως βασίζονται στο POSIX sockets στανταρντ τα οποία με την σειρά τους είναι εξέλιξη του Berkeley sockets API.
  - Τα Berkeley sockets API εμφανίστηκε στο BSD4.1 UNIX το 1981
  - Τα BSD sockets API είναι γραμμένα σε C
  - Οι περισσότερες γλώσσες προγραμματισμού παρέχουν παρόμοια API τα οποία βασίζονται (και χρησιμοποιούν) στο αντίστοιχο C API.

Sockets, IP Addresses και Ports

- A socket address is the combination of an IP address and a port number, much like one end of a telephone connection is the combination of a phone number and a particular extension.
- Based on this address, internet sockets deliver incoming data packets to the appropriate application process or thread.
- 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:
  - 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:**

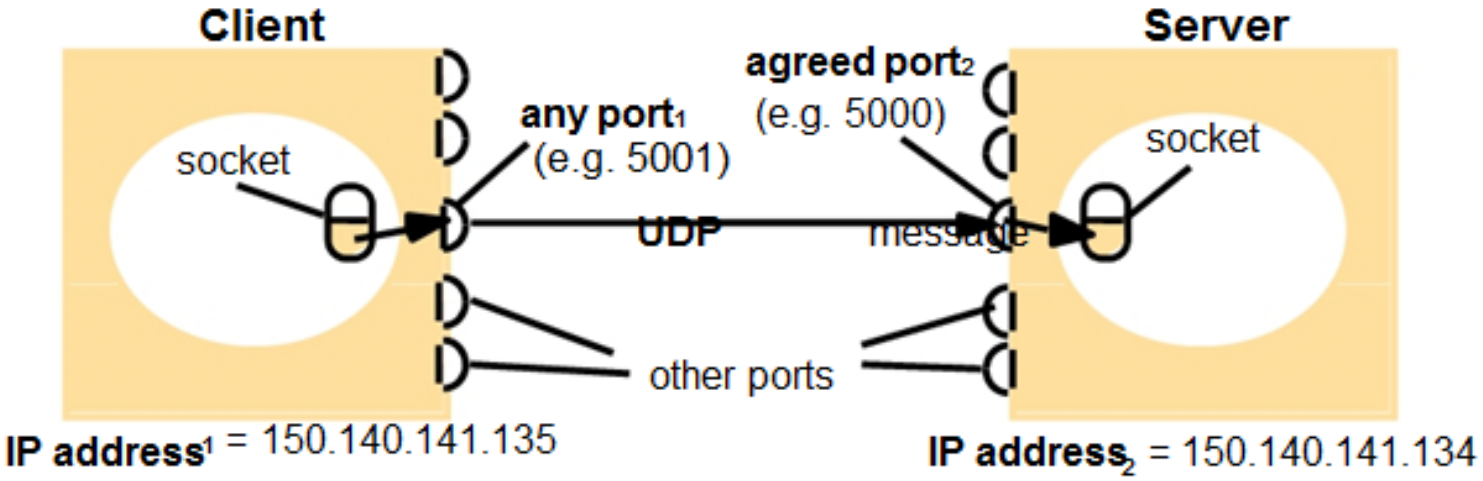
  { Protocol / IP address1 / Port1 / IP address2 / Port2 }

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

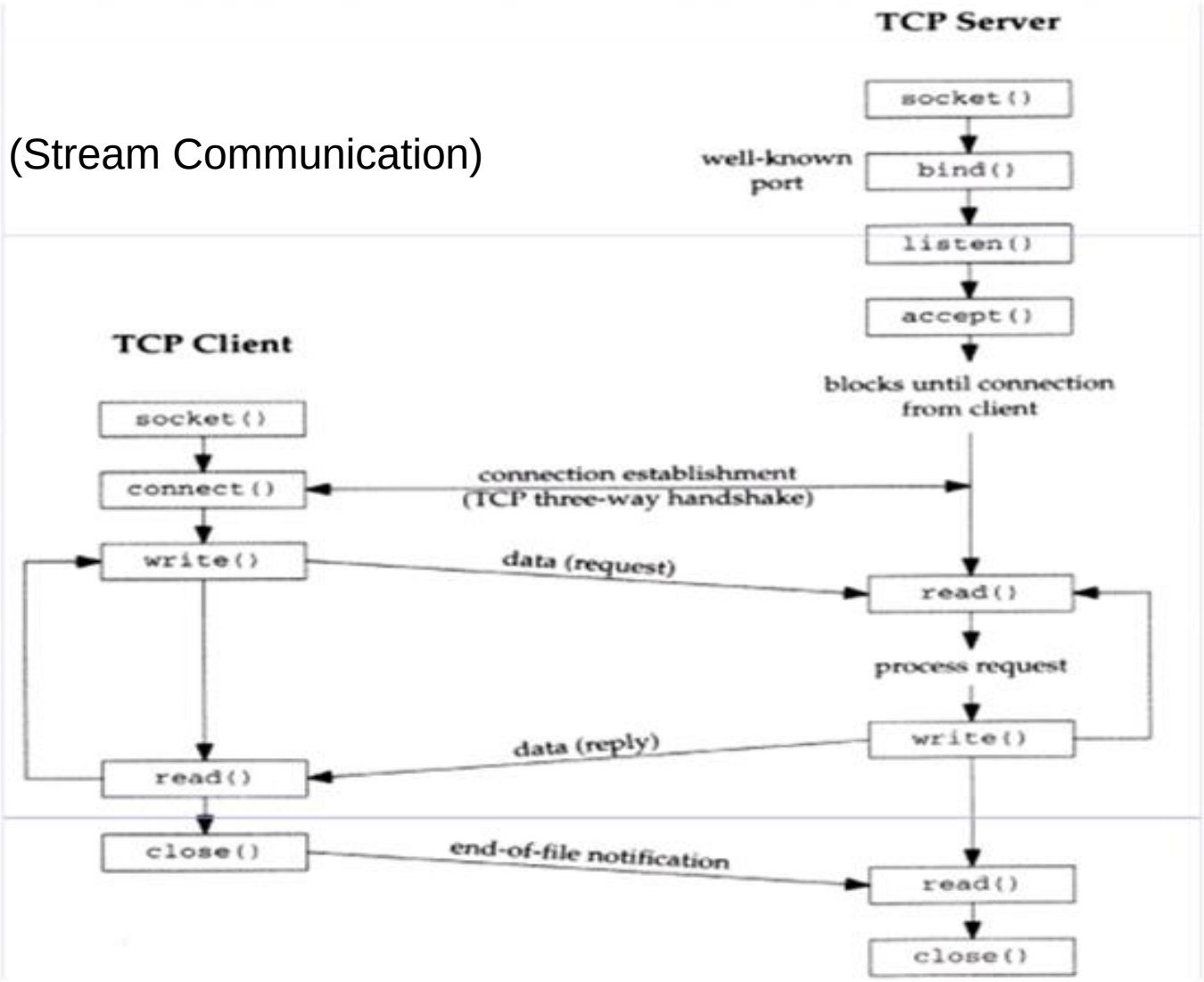


- **Η επικοινωνία από άκρη-σε-άκρη (end-to-end connection) ταυτοποιείται μονοσήμαντα με την πεντάδα:**

{ 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()`)

Steps in Establishing a Socket on Server

- Create a socket with the ***socket()*** system call
- Bind the socket to an address using the ***bind()*** system call. For a server socket, an address consists of a port number on the host machine
- Listen for connections with ***listen()*** system call
- Accept a connection with the ***accept()*** system call. This call typically blocks until a client connects with the server
- **write()** and **read()** data

Πρωτογενείς κλήσεις για τα sockets

- Οι οδηγίες (#include) που χρησιμοποιούνται για αυτές τις κλήσεις είναι οι :
  - ***#include <sys/types.h>***

    This header file contains definitions of a number of data types used in system calls. These types are used in the next two include files.

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

    The header file socket.h includes a number of definitions of structures needed for sockets.

  - ***#include <netinet/in.h>***

    The header file in.h contains constants and structures needed for internet domain addresses.

Η κλήση “socket”

Αρχικοποίηση ενός socket

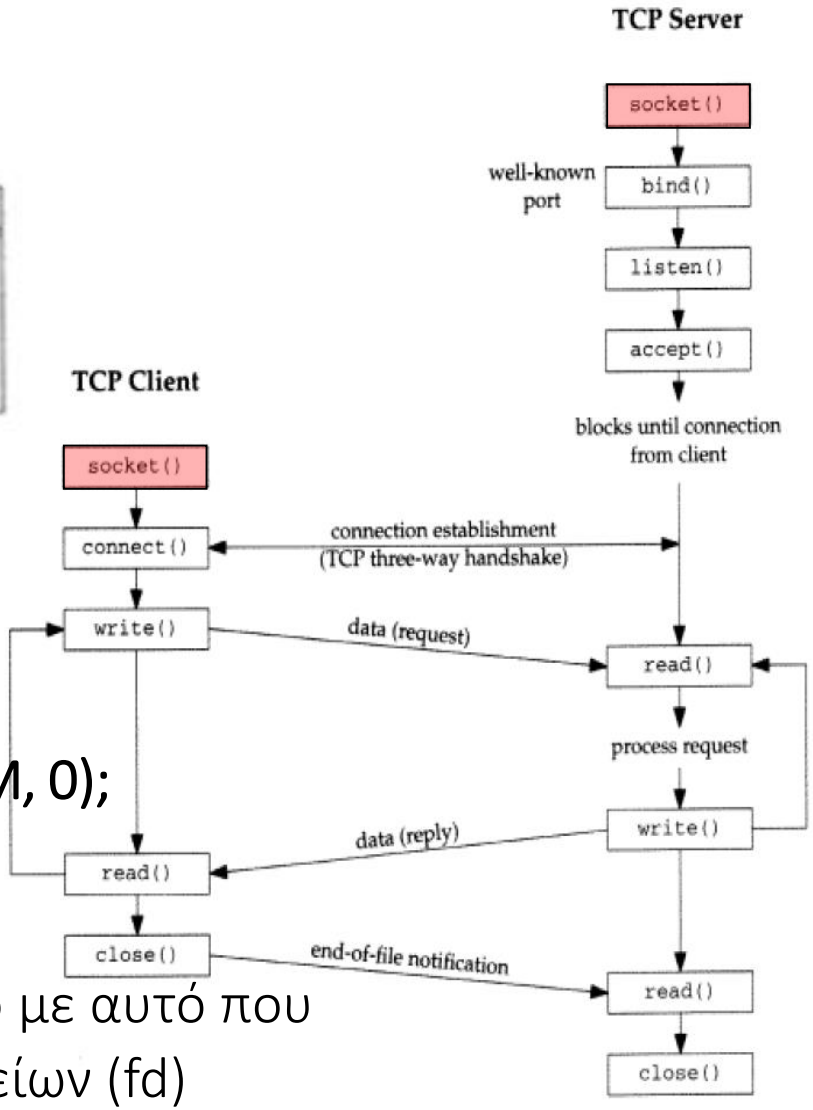
int socket (int family, int type, int protocol)

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

type	Description
SOCK_STREAM	stream socket
SOCK_DGRAM	datagram socket
SOCK_RAW	raw socket

π.χ.

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



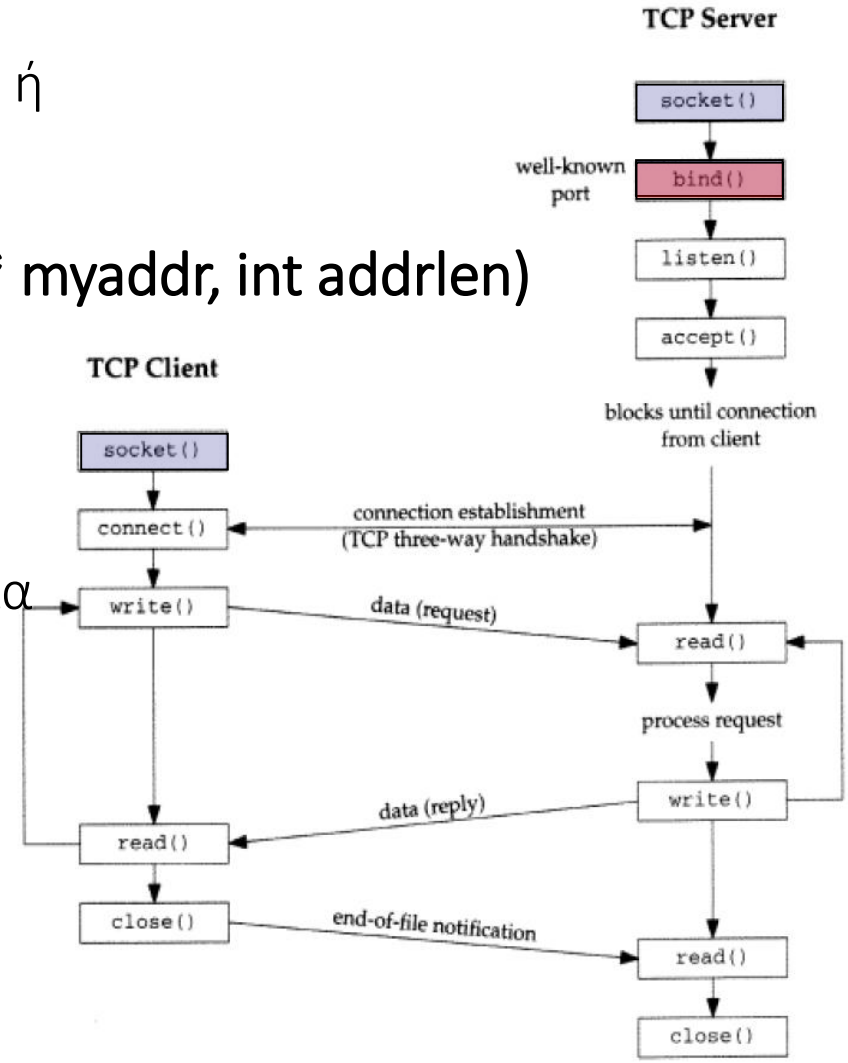
Επιστρέφει ένα ακέραιο (**sockfd**) ανάλογο με αυτό που επιστρέφουν οι ρουτίνες διαχείρισης αρχείων (fd)

Η κλήση “bind” - Ονομασία του Socket

- Αρχικά το socket δεν έχει IP διεύθυνση ή port

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 διεργασία



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`

```
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 */
};
```

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

sin_port: καθορίζει το port number

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

π.χ.

struct sockaddr_in serv_addr;

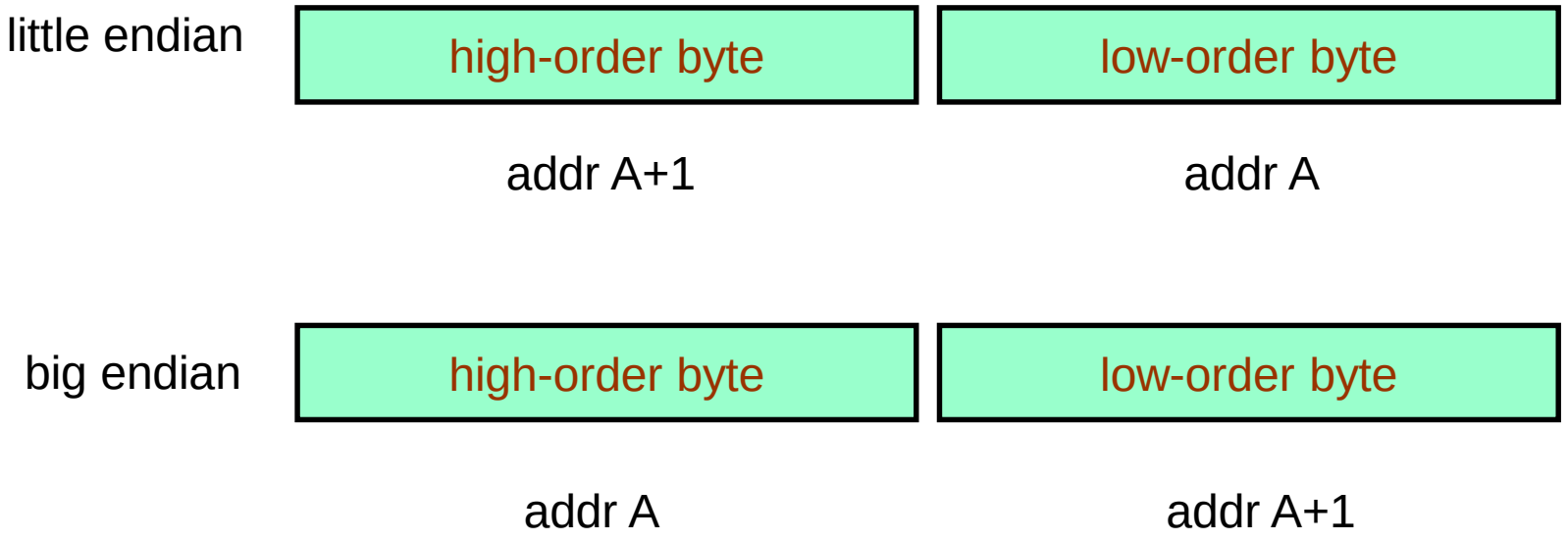
serv_addr.sin_family = AF_INET;

serv_addr.sin_port = htons(portno);

serv_addr..sin_addr.s_addr = htonl(INADDR_ANY);

Byte Ordering

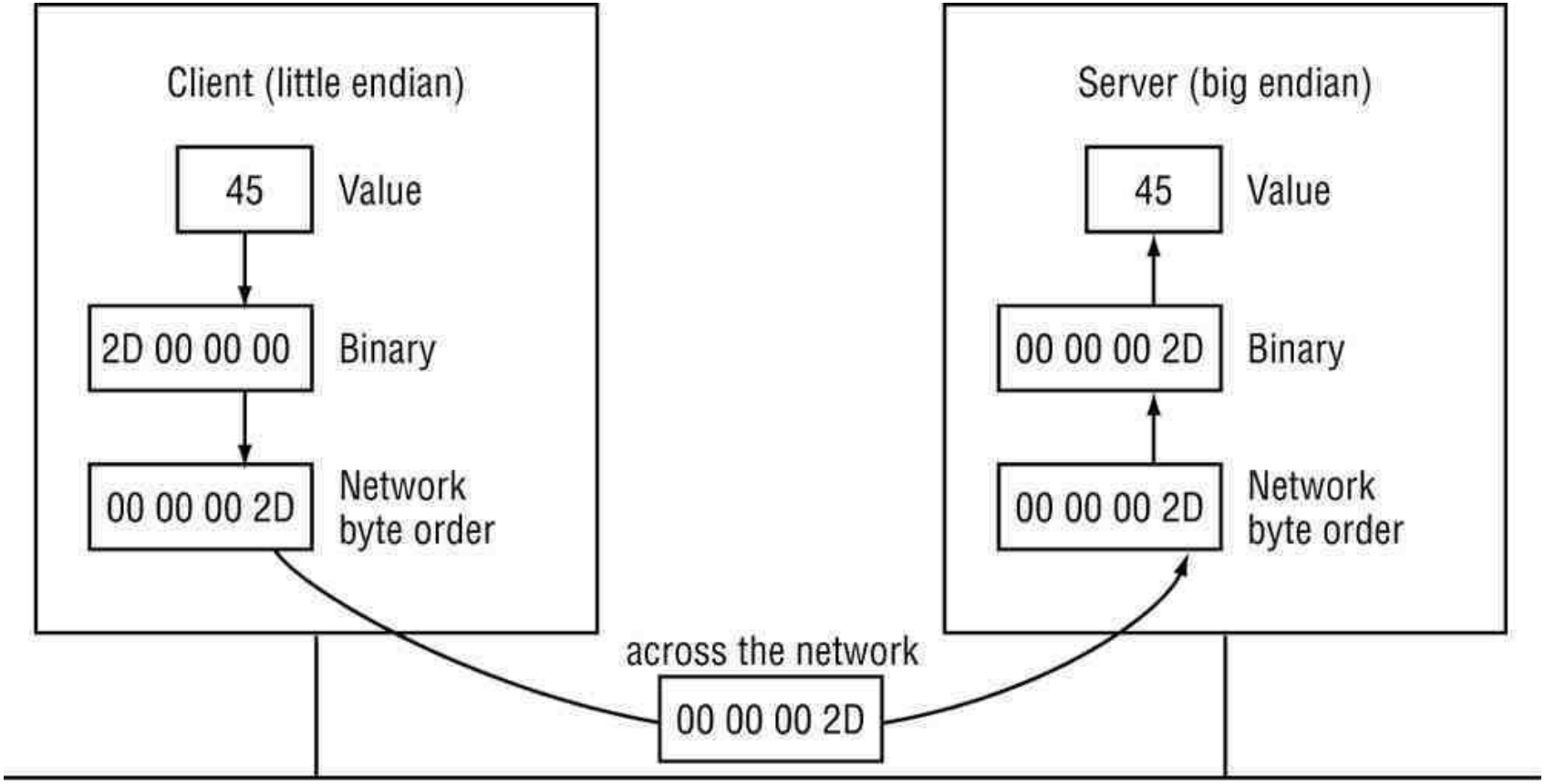
- different byte ordering in various platform
  - little endian : Intel 80x86 , DEC VAX , DEC PDP-11
  - big endian : IBM 370 , Motorola 68000 , Pyramid



Byte Ordering

- The problem of using different binary datatype representations in various systems.
- A solution: send binary datatypes in a *generic* method.
- The *network byte order* representation of binary datatypes was created as intermediate storage for binary data to be transmitted across the network.
- The idea is for each network program to convert its own local binary data into network byte order before transmitting it.
- On the receiving side, the system must convert the incoming data from network byte order into its own internal byte order.
- This ensures that the binary data will be converted to the proper representation for the destination host

Byte Ordering



Byte Ordering Routines

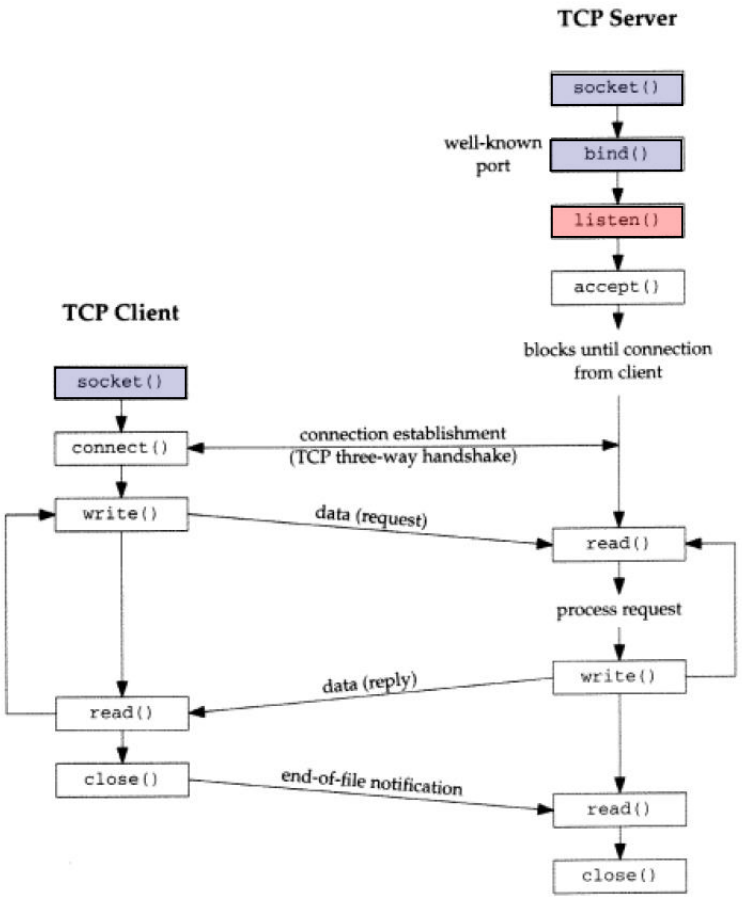
Network Byte Order is big endian

```
u_long htonl (u_long hostlong);       //convert host-to-network,long int
u_short htons(u_short hostshort);   //convert host-to network,short int
u_long ntohl (u_long netlong);        // convert network-to-host,long int
u_short ntohs (u_short netshort);   // convert network-to-host, short
                                    // integer
```

Η κλήση “listen” (Προαιρετική)

- Σύνταξη:
  `int listen (int sockfd, int backlog)`

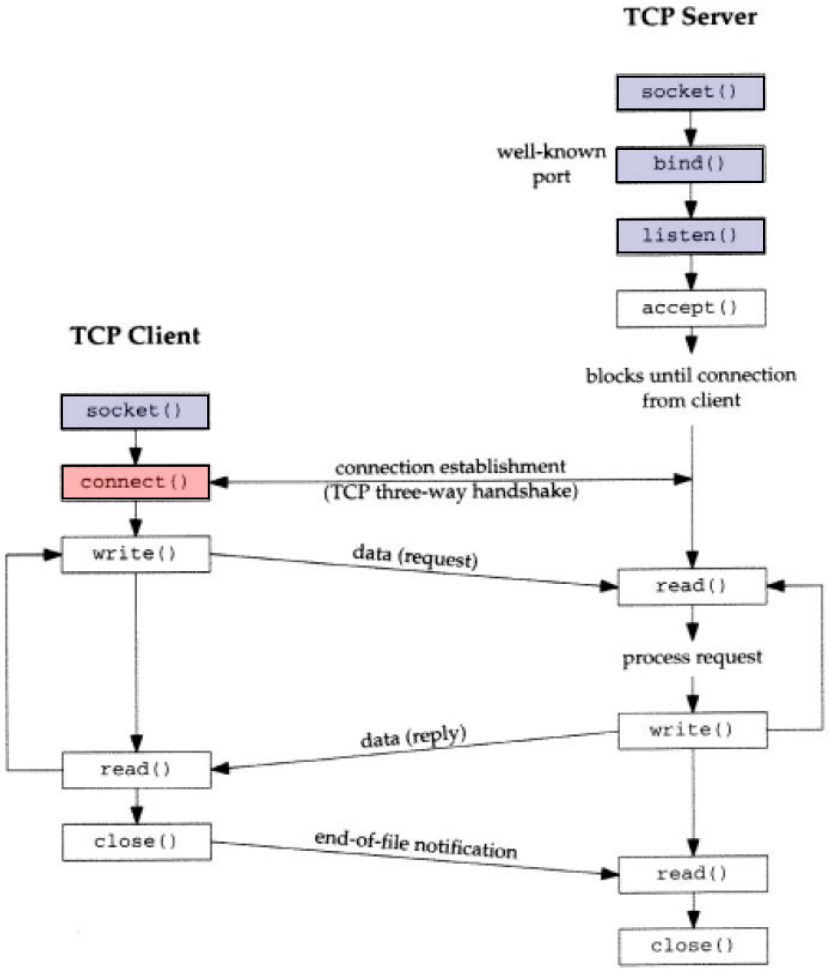
- Χρησιμοποιείται από ένα connection-oriented εξυπηρετητή για να ενημερωθεί το σύστημα ότι ο (εξυπηρετητής) είναι έτοιμος να λάβει μηνύματα στο socket με descriptor το **sockfd**

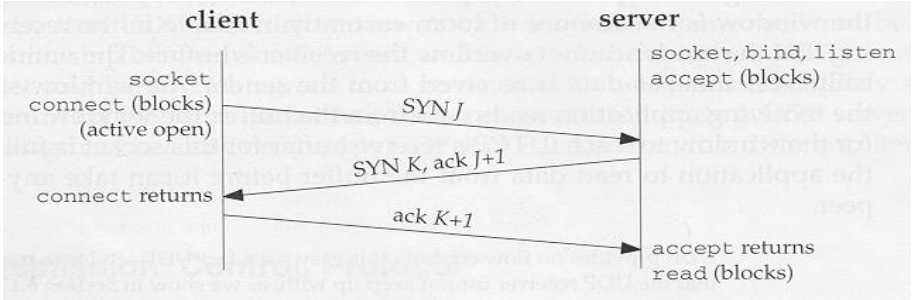


Σύνδεση (“connect”) client στο server

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

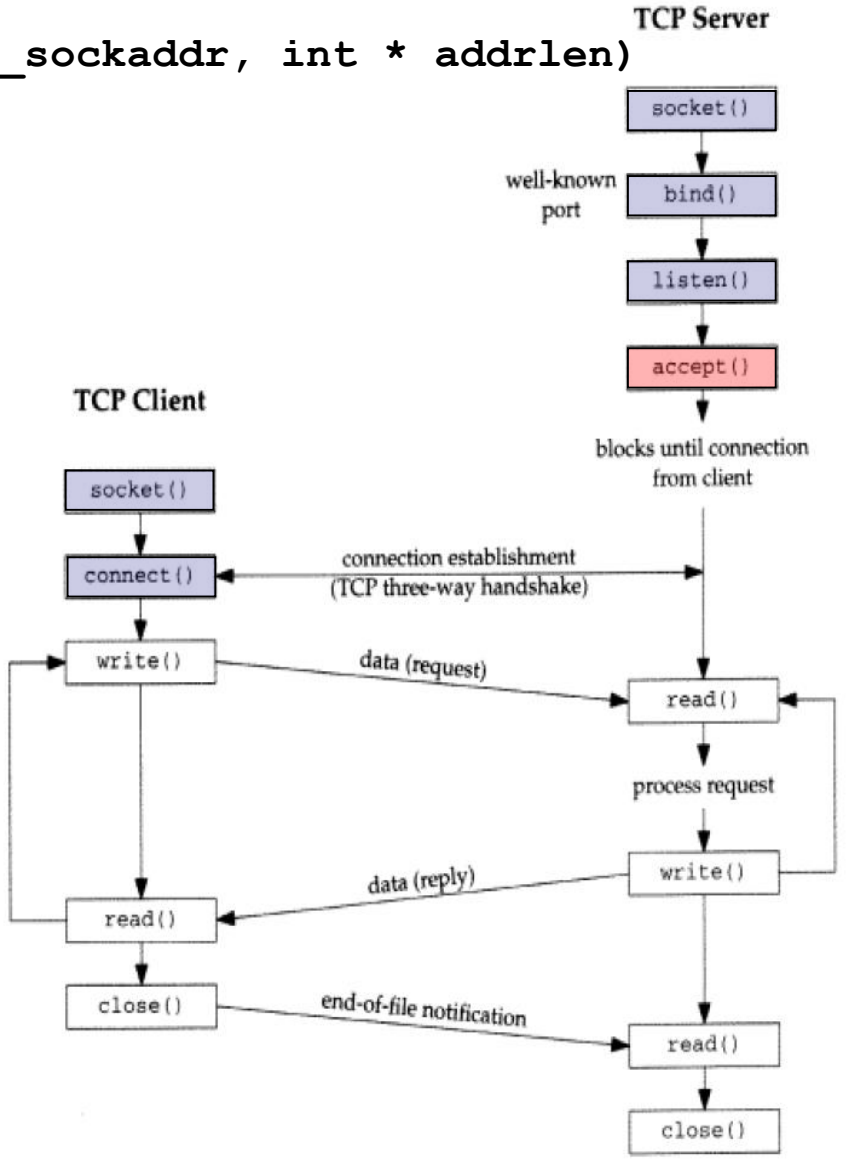




Αποδοχή (“accept”) σύνδεσης

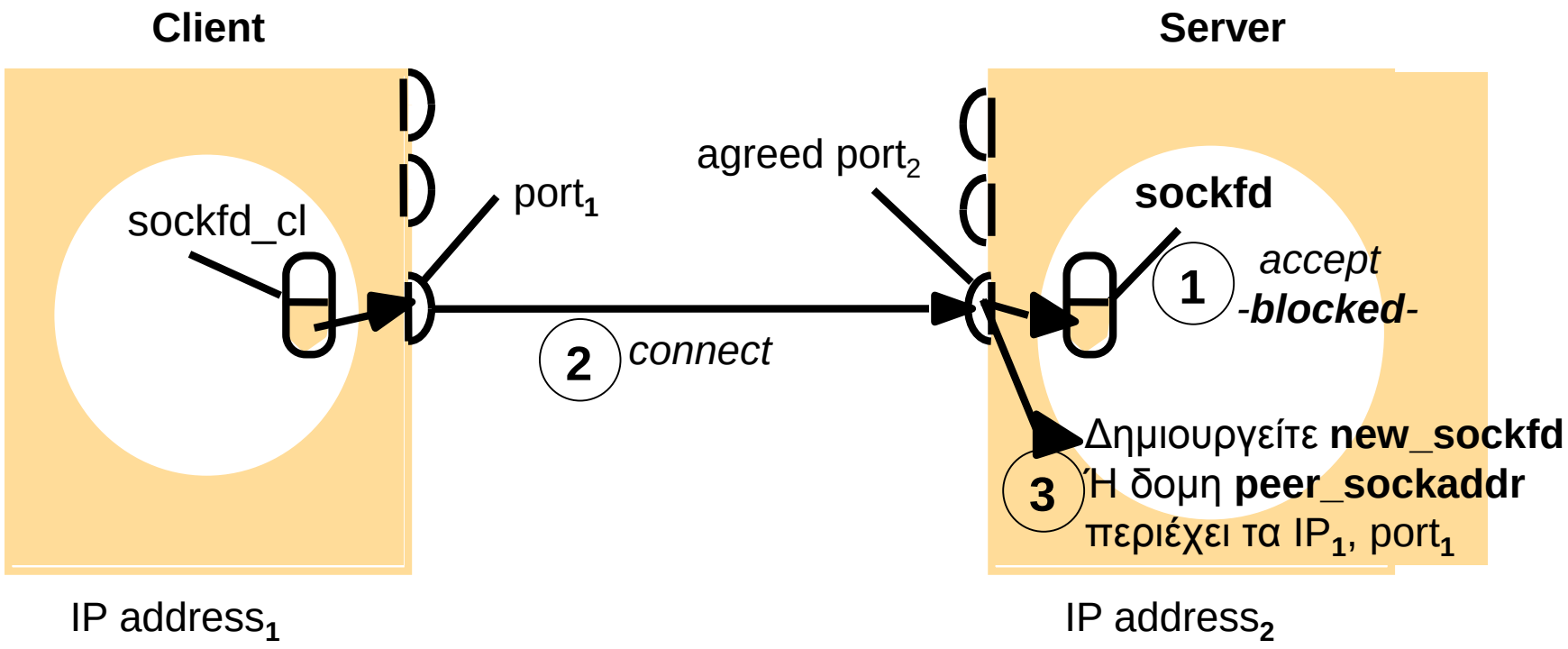
```
int accept (sockfd, struct sockaddr *peer_sockaddr, int * addrlen)
```

- 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”

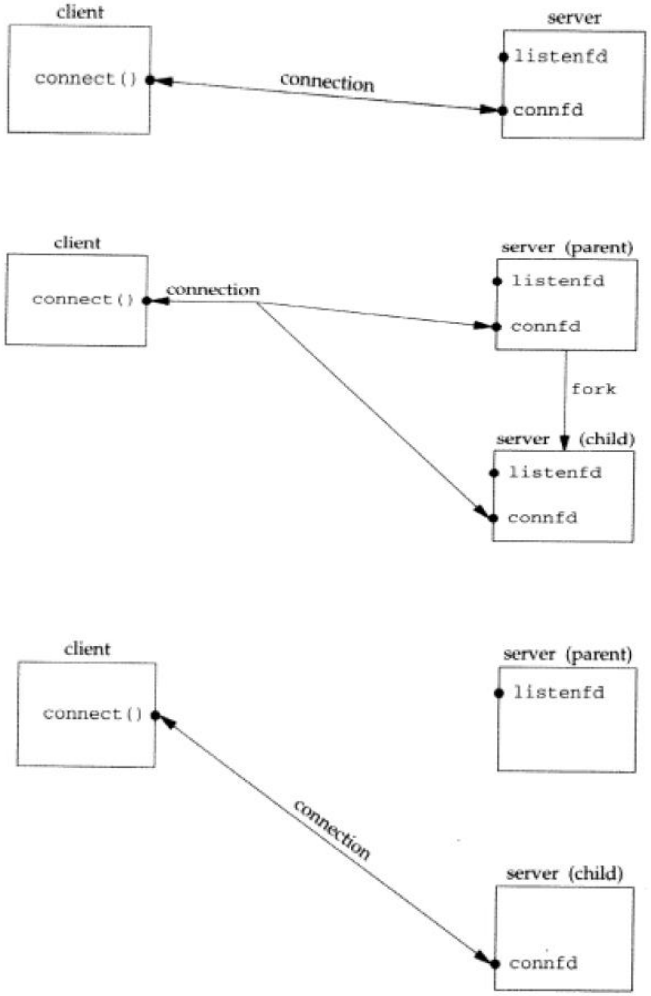
- Απο την στιγμή που η accept() επιστρέφει το νέο socket, το κανάλι επικοινωνίας με τον client έχει εγκατασταθεί
- Μέχρι να ξανακληθεί η accept(), ο server δεν μπορεί να δεχτεί άλλες κλήσεις για σύνδεση
- Οι κλήσεις που γίνονται όσο ο server βρίσκεται εκτός της accept() τοποθετούνται σε μια ουρά (default 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())

Η κλήση “fork”

- Σύνταξη:

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

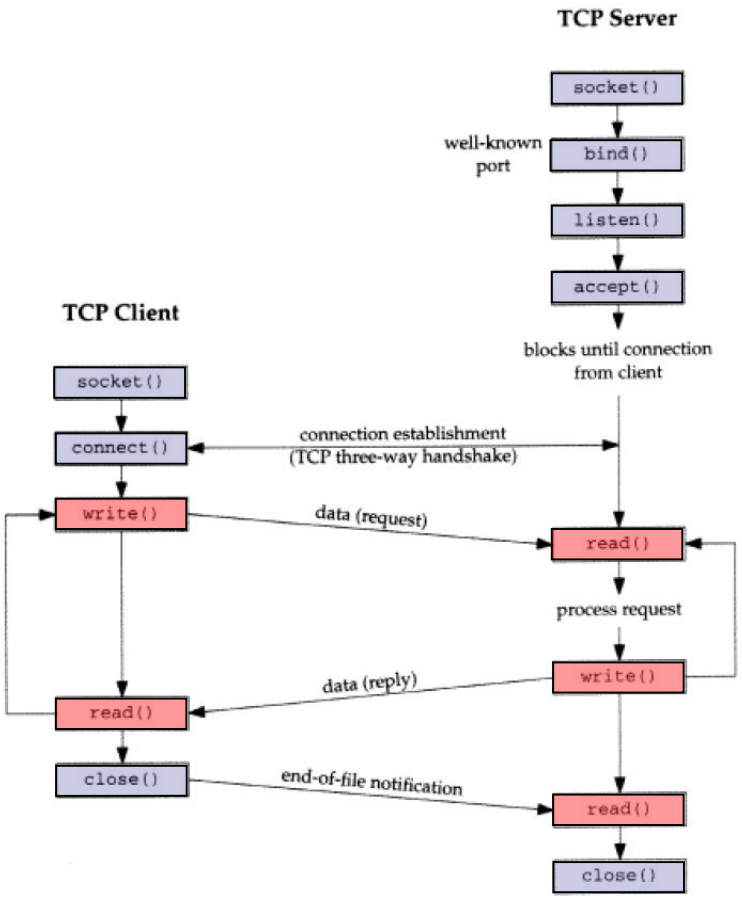
- Επιστρέφει 0 στο «παιδί»
- Το «παιδί» μπορεί να βρει τον «πατέρα» του καλώντας την getppid()
- Ο «πατέρας» μπορεί να έχει πολλά παιδιά



Επικοινωνία, Είσοδος και Έξοδος

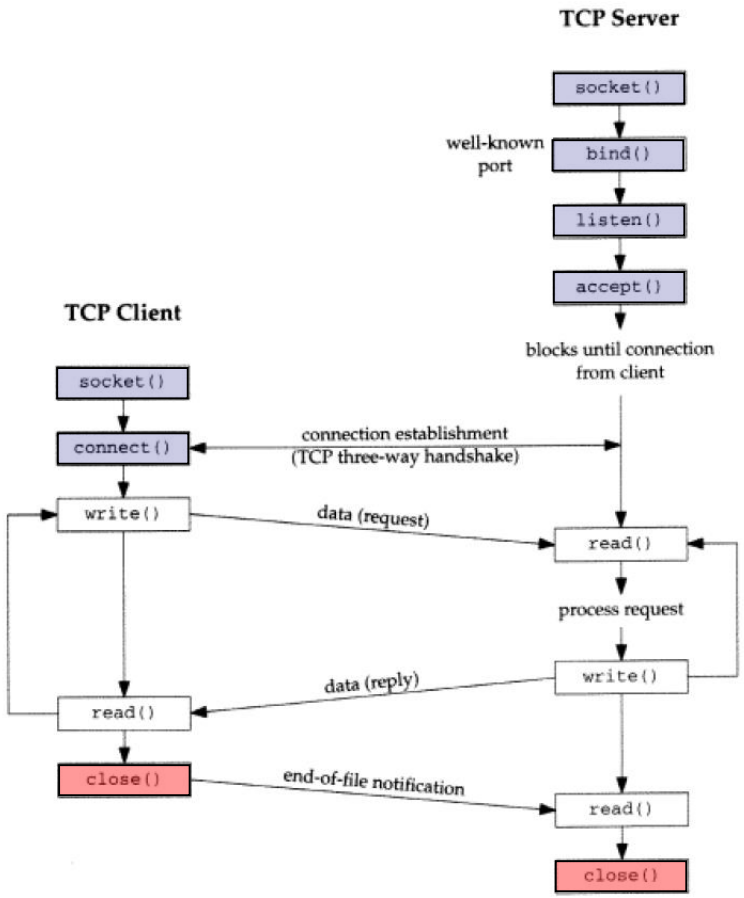
Ο socket descriptor έχει την ίδια λειτουργικότητα με έναν file descriptor

- 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»



Η κλήση “close”

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



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

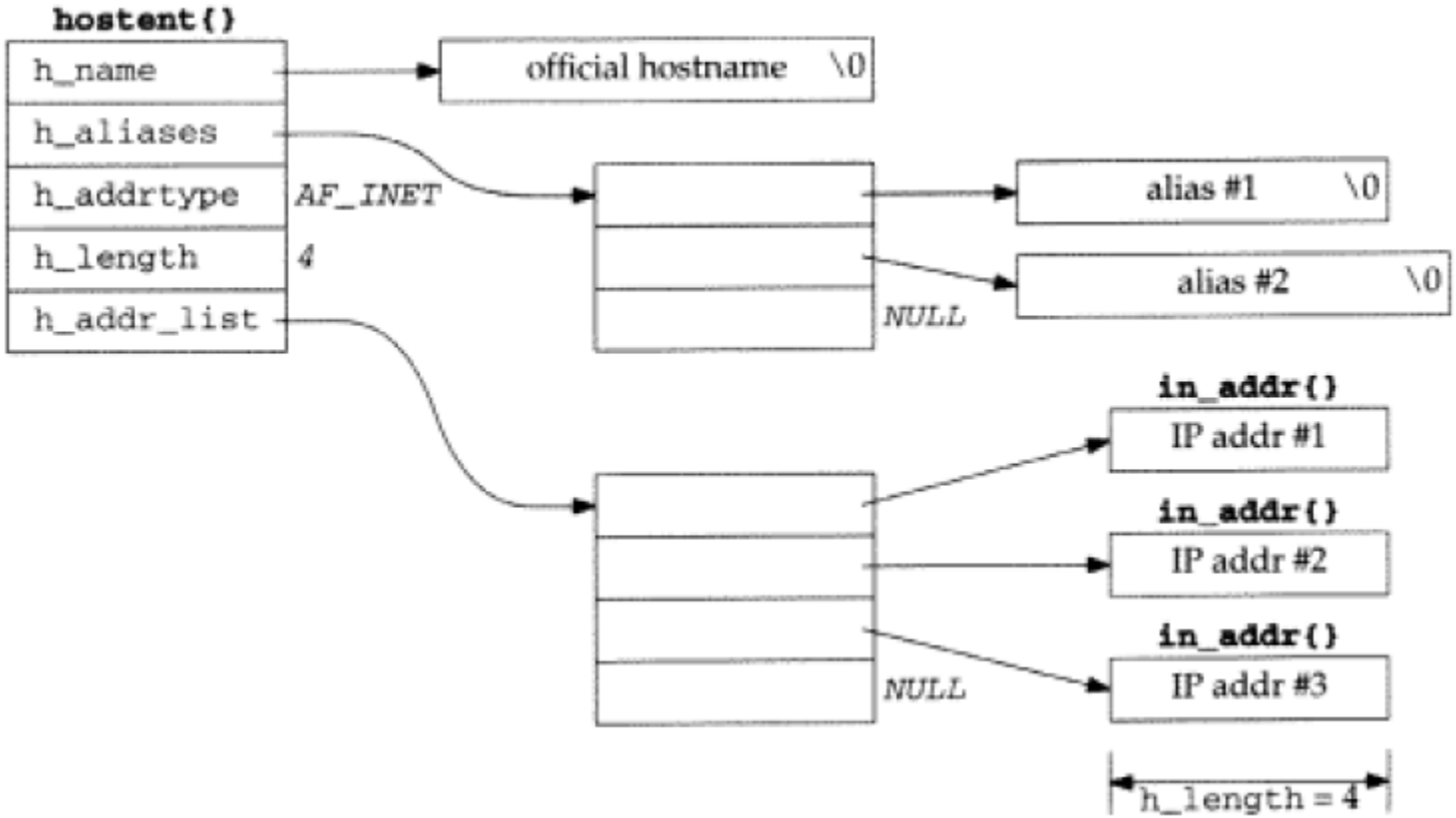
```
#include <netdb.h>

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);
```

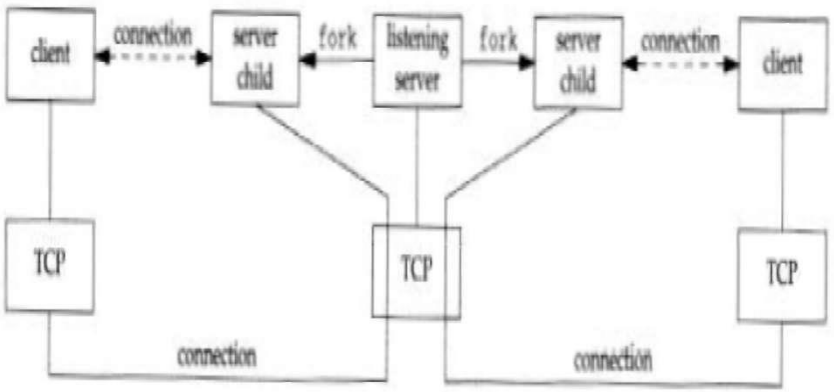
Returns: nonnull pointer if OK, NULL on error

```
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 */
};
```

UDP vs TCP

- Το UDP είναι ένα connectionless, μη αξιόπιστο, datagram transport protocol
  - Δεν μπορεί να χειριστεί διπλά πακέταή πακέτα σε λάθος σειρά
  - Δεν παρέχει flow control και congestion avoidance μηχανισμούς
- Έχει ελάχιστο overhead
- Εφαρμογές που χρησιμοποιούν το UDP:
  - DNS, NFS, SNMP, TFTP, real-time multiplayer games, voice conferencing, broadcasting

UDP vs TCP (client/server)

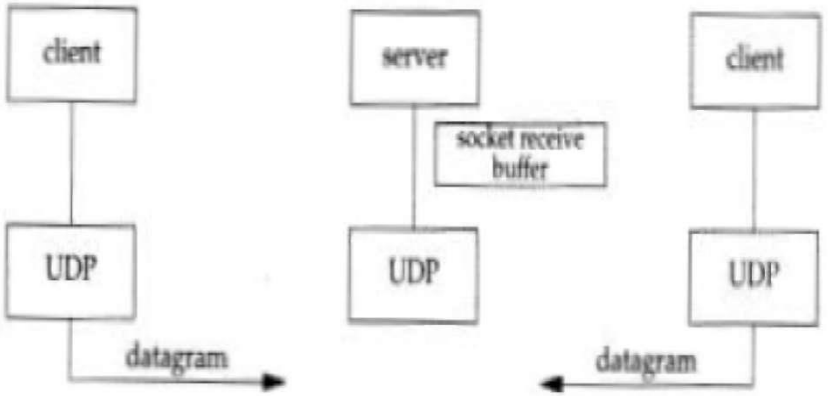


TCP

Ο client επικοινωνεί με έναν «αφοσιωμένο» αντίγραφο του server, ενώο «αρχικός» server μπορεί να δέχεται νέες κλήσεις σύνδεσης

UDP

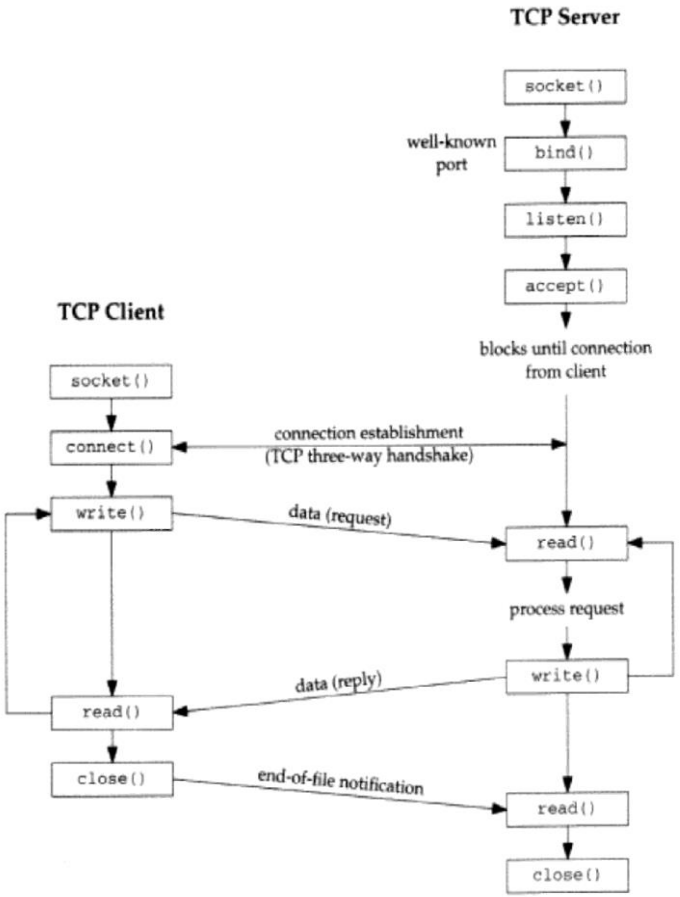
Ο client στέλνει πακέτα (datagram) στον server. Τα πακέτα αποθηκεύονται σε μια ουρά και επεξεργάζονται απότον server



UDP vs TCP (client/server)

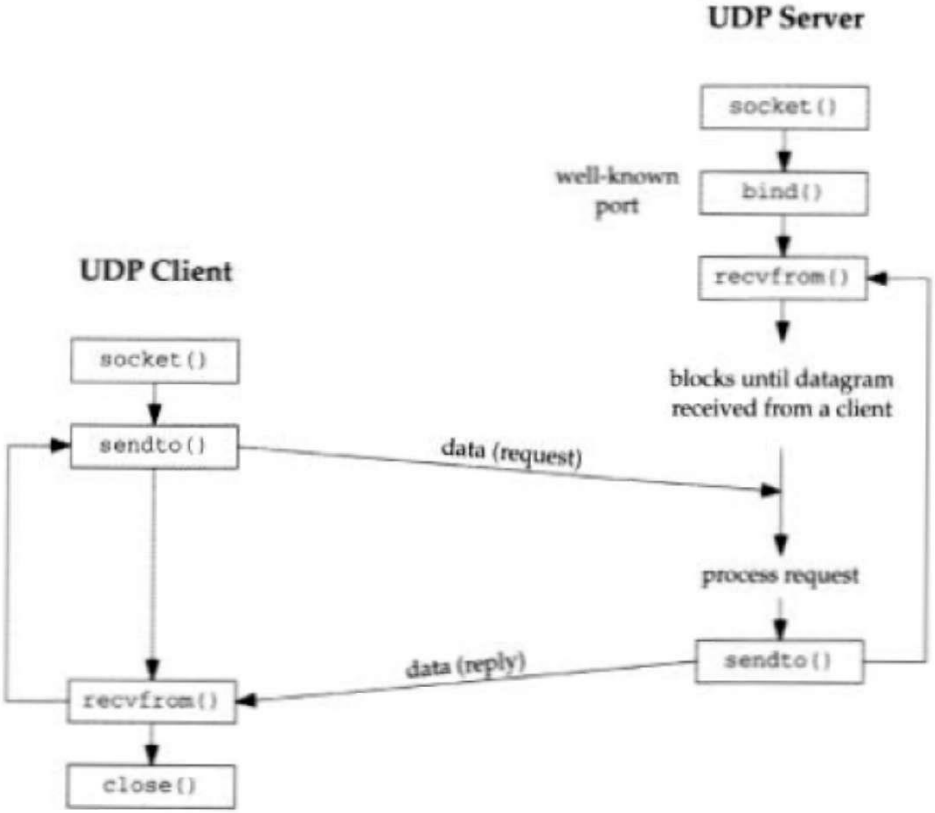
TCP Client/Server interaction

(Stream Communication)



UDP Client/Server interaction

(Datagram Communication)



UDP Sockets - Connectionless

- Server
  - create endpoint (socket())
  - bind address (bind())
  - transfer data (sendto() recvform())
- Client
  - create endpoint (socket())
  - transfer data (sendto() recvform())

Η κλήση "recvfrom"

- Σύνταξη:
  - **ssize_t recvfrom (int sockfd, void *buff, size_t nbytes, int flags, struct sockaddr *from, socklen_t *addrlen);**
- Περιμένει μέχρι να φτάσουν δεδομένα (blocking)
- Επιστρέφει το size των δεδομένων (datagram) που έλαβε
- Στο void *buff αποθηκεύει τα δεδομένα (datagram) με μέγιστο μέγεθος nbytes
- Στην δομή sockaddr *from επιστρέφει την διεύθυνσητουαποστολέα
- Δεν υπάρχει καμία σύνδεση (connection) με τον αποστολέα

Η κλήση " sendto "

- Σύνταξη:
  - **ssize_t sendto (int sockfd, void *buff, size_t nbytes, int flags, const struct sockaddr *to, socklen_t *addrlen);**
- Στέλνει τα δεδομένα (datagram) που βρίσκονται στη παράμετρο void *buff με μέγεθος nbytes στον προορισμό (δηλώνεται μέσω της δομής const struct sockaddr * to)
- Δεν υπάρχει καμία σύνδεση (connection) με τον παραλήπτη

Sockets σε άλλες γλώσσες

- Java, PhP, .NET, Perl ....
- Ακολουθούν πιστά το μοντέλο των Berkeley sockets της C
- Συνήθως ευκολότερο error handling (exceptions)
- Object-oriented
- Όμως παρέχουν και higher-level μηχανισμούς για network communication (RMI, Web services, CORBA, Protocol-specific libraries)
- Εφόσον χρειάζεται να προγραμματίσετε με sockets, σημαίνει ότι ήδη θα έχετε συνήθως επιλέξει C\C++!

Ερωτήσεις?

Χρησιμοποιήθηκαν εν μέρη εικόνες και κείμενο από:

- J. Kurose and K. Ross, Computer Networking: A Top-Down Approach
- CS 498MC: Systems and Networking Lab
- CSEE W4140: Networking Laboratory