# Κατανεμημένα Συστήματα Επικοινωνία Client/Server Υποδοχές

Χάρης Μανιφάβας
Τμήμα Εφ. Πληροφορικής & Πολυμέσων
ΤΕΙ Κρήτης

# Ορισμός των Υποδοχών (sockets)

- Τα sockets προσφέρουν μια γενικευμένη ικανότητα δικτύωσης
  - Είναι διαθέσιμα με τις περισσότερες τρέχουσες διανομές των συστημάτων Unix
  - Επιτρέπουν την επικοινωνία μεταξύ δύο διαφορετικών διεργασιών στην ίδια ή σε διαφορετικές μηχανές

- Σε έναν προγραμματιστή τα sockets φαίνονται και συμπεριφέρονται σαν έναν χαμηλού επιπέδου file descriptor
  - Αυτό συμβαίνει επειδή εντολές όπως read() και write() χρησιμοποιούνται για τα sockets με τον ίδιο τρόπο που χρησιμοποιούνται με τα αρχεία

# Ορισμός των Υποδοχών (sockets)

- In computer networking, an **Internet socket** or **network socket** is an endpoint of a bidirectional inter-process communication flow across an Internet Protocol-based computer network, such as the Internet

- The term *Internet sockets* is also used as a name for an application programming interface (API) for the TCP/IP protocol stack, usually provided by the operating system
  - Internet sockets constitute a mechanism for delivering incoming data packets to the appropriate application process or thread, based on a combination of local and remote IP addresses and port numbers
  - Each socket is mapped by the operational system to a communicating application process or thread

# Ορισμός των Υποδοχών (sockets)

- In computing, **network programming**, essentially identical to *socket programming* or *client–server programming*, involves writing computer programs that communicate with other programs across a computer network
  - The program or process initiating the communication is called a client process, and the program waiting for the communication to be initiated is the server process
  - The client and server processes together form a distributed system
  - The communication between the client and server process may either be connection-oriented (such as an established TCP virtual circuit or session), or connectionless (based on UDP datagrams)
  - A program that can act both as a client and a server is based on peer-to-peer communication
- Sockets are usually implemented by an API library such as Berkeley sockets, first introduced in 1983
  - Most implementations are based on Berkeley sockets, for example Winsock introduced 1991
  - Other socket API implementations exist, such as the STREAMS-based Transport Layer Interface (TLI)

# Ορισμός των Υποδοχών (sockets)

- The **Berkeley sockets** application programming interface (API)
  - Comprises a library for developing applications in the C programming language that perform inter-process communication, most commonly for communications across a computer network
  - Berkeley sockets (also known as the BSD socket API) originated with the 4.2BSD Unix operating system (released in 1983) as an API
  - Only in 1989, however, could UC Berkeley release versions of its operating system and networking library free from the licensing constraints of AT&T's copyright-protected Unix
  - The Berkeley socket API forms the *de facto* standard abstraction for network sockets
  - Most other programming languages use an interface similar to the C API
  - The API is also used for Unix domain sockets, which are an interface to interprocess communication (IPC) channels within a single computer
  - All modern operating systems now have some implementation of the Berkeley socket interface, as it became the standard interface for connecting to the Internet

# Ορισμός των Υποδοχών (sockets)

- Για τις περισσότερες διαδικασίες που χρησιμοποιούν sockets, οι ρόλοι του πελάτη (client) και του κεντρικού εξυπηρετητή (server) πρέπει να οριστούν
  - Ένας server είναι μια διεργασία που κάνει κάποια λειτουργία κατόπιν αιτήσεως από έναν client
  - Όπως θα φανεί παρακάτω, οι ρόλοι δεν είναι συμμετρικοί και δεν μπορούν να αντιστραφούν εύκολα χωρίς κάποια προσπάθεια

# Χρήση

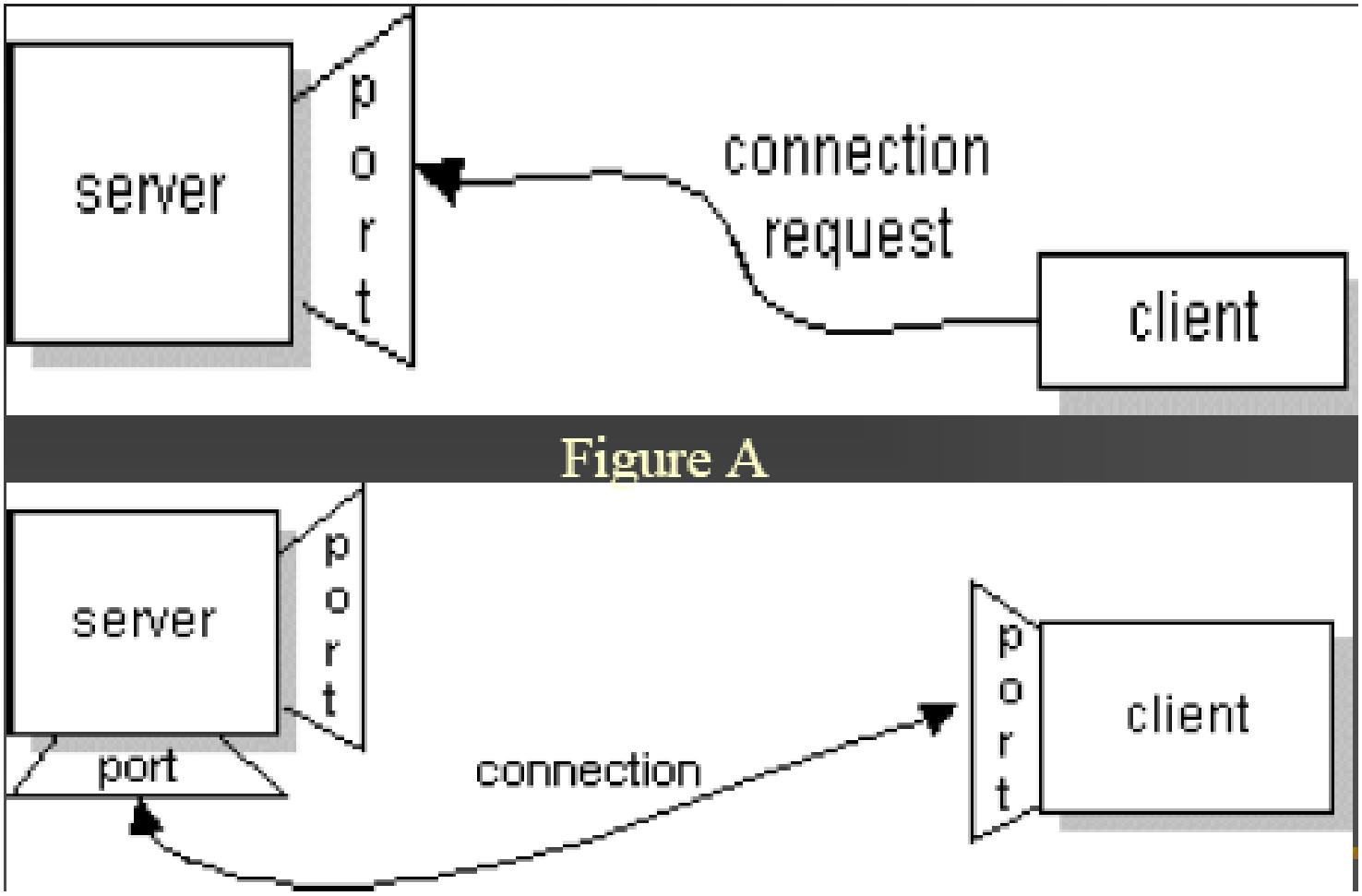
- Χρήση των sockets με έναν connectionless ή datagram τρόπο μεταξύ των client και server στην ίδια μηχανή
  - Σε αυτήν την περίπτωση, ο client δεν εγκαθιστά ρητά μια σύνδεση με τον server
  - Ο client, φυσικά, πρέπει να ξέρει τη διεύθυνση του server
  - Ο server, στη συνέχεια, περιμένει απλά ένα μήνυμα να καταφθάσει. Η διεύθυνση του client είναι μια από τις παραμέτρους του μηνύματος και χρησιμοποιείται από τον server για την απάντησή του

# Χρήση

- Χρήση των sockets με έναν συνδεδεμένο τρόπο μεταξύ των client και server στην ίδια μηχανή
  - Σε αυτήν την περίπτωση, οι ρόλοι του πελάτη και του κεντρικού υπολογιστή ενισχύονται περαιτέρω από τον τρόπο με τον οποίο το socket καθιερώνεται και χρησιμοποιείται
  - Αυτό το μοντέλο αναφέρεται συχνά ως connection-oriented client-server μοντέλο

- Χρήση των sockets με έναν συνδεδεμένο τρόπο μεταξύ των client και server σε διαφορετικές μηχανές
  - Αυτό είναι η επέκταση στο δίκτυο του δεύτερου σταδίου που μόλις αναφέρθηκε

# Χρήση


server
port
connection
request
client
Figure A
server
port
port
connection
port
client

# Internet Sockets

- Τα δύο βασικά είδη Internet sockets
  - Stream Sockets (SOCK_STREAM)
  - Datagram Sockets (SOCK_DGRAM)

- Stream Sockets
  - Είναι αξιόπιστα (error free)
  - Διπλής κατεύθυνσης
  - Telnet, HTTP χρησιμοποιούν stream sockets
  - Υψηλής ποιότητας μετάδοση (χρήση TCP)

- Datagram Sockets
  - Connectionless sockets
  - Πακέτα μπορεί να χαθούν
  - Χαμηλή ποιότητα μετάδοσης (χρήση UDP)

# Internet Sockets

- Ένα stream είναι μια ακολουθία από χαρακτήρες που "ρέουν" προς ή από μια διεργασία
- Ένα input stream συνδέεται σε κάποια πηγή εισόδου (input source) της διεργασίας, π.χ., πληκτρολόγιο ή socket
- Ένα output stream συνδέεται σε κάποια πηγή εξόδου (output source), π.χ., οθόνη ή socket

# Internet Sockets

- Ο Client πρέπει να επικοινωνήσει με τον server
  - Η διεργασία του server πρέπει να "τρέχει" ήδη
  - Ο server πρέπει να έχει δημιουργήσει ένα socket (πόρτα) για την "υποδοχή" της επικοινωνίας από την πλευρά του client

- Ο Client επικοινωνεί με τον server (TCP)
  - Με τη δημιουργία ενός τοπικού TCP socket
  - Προσδιορίζοντας μια IP διεύθυνση & αριθμό port της διεργασίας του server
  - Ο client TCP εγκαθιστά σύνδεση με το server TCP

  - Όταν κάποιος client επιδιώκει την επικοινωνία, το server TCP δημιουργεί νέο socket για την επικοινωνία της διεργασίας του server με τον client
    - Επιτρέπει στο server να "μιλάει" με πολλαπλούς clients
    - Ξεχωριστά port numbers για τη διάκριση μεταξύ clients

# Internet Sockets

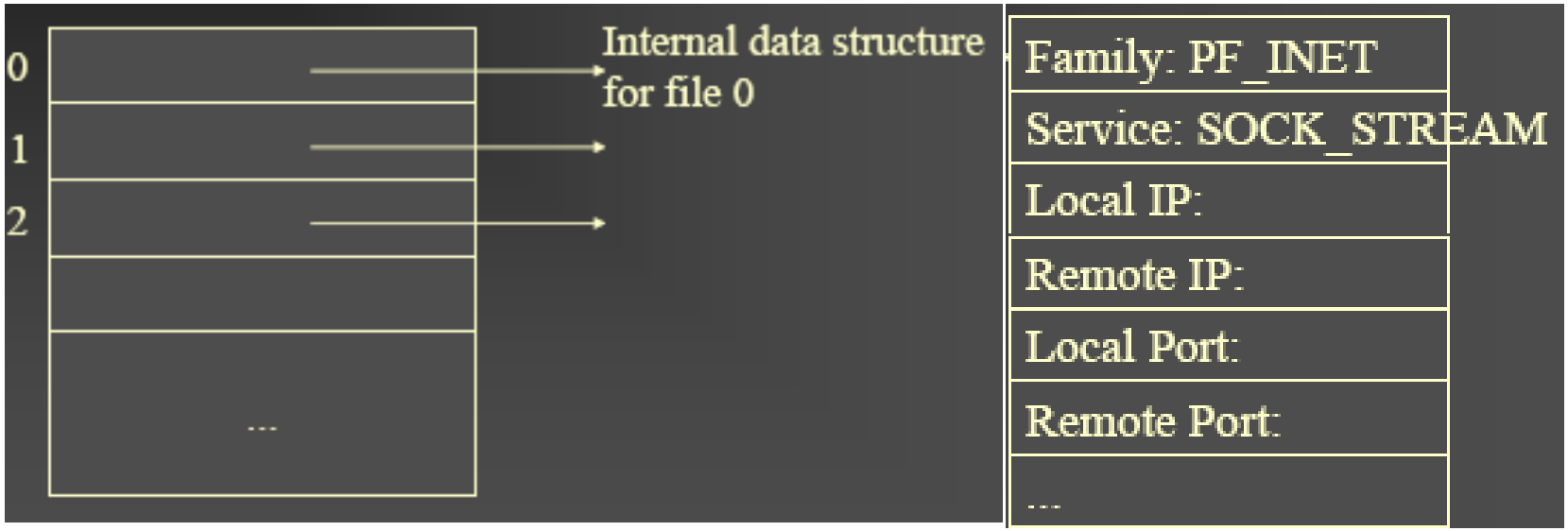
- Ο Client επικοινωνεί με τον server (UDP)
  - Δεν υπάρχει χειραψία (handshaking)
  - Δεν εγκαθίσταται “σύνδεση” μεταξύ client και server
  - Ο αποστολέας επισυνάπτει ρητά την IP διεύθυνση και το port προορισμού σε κάθε πακέτο
  - Ο server πρέπει να εξάγει την IP διεύθυνση και το port του αποστολέα από κάθε πακέτο που λαμβάνει
  - Τα μεταδιδόμενα δεδομένα μπορεί να παραληφθούν εκτός σειράς ή να χαθούν

# UNIX I/O

- Βασικές συναρτήσεις Εισόδου/Εξόδου
  - open
  - close
  - read (*see also* recv *and* recvfrom)
  - write (*see also* send *and* sendto)
  - lseek

```
int desc;
...
desc = open("file", O_RDWR, 0);
read(desc, buffer, 128);
…
close(desc);
```

# UNIX I/O


0
1
2
...
Internal data structure
for file 0
Family: PF_INET
Service: SOCK_STREAM
Local IP:
Remote IP:
Local Port:
Remote Port:
...

# Socket API Functions

- **socket()**
  - Creates a new socket of a certain socket type, identified by an integer number, and allocates system resources to it
- **bind()**
  - Associates a socket with a socket address structure, i.e. a specified local port number and IP address.
- **listen()**
  - Used on the server side, and causes a bound TCP socket to enter listening state
- **connect()**
  - Used on the client side, and assigns a free local port number to a socket
  - In case of a TCP socket, it causes an attempt to establish a new TCP connection
- **accept()**
  - Used on the server side
  - It accepts a received incoming attempt to create a new TCP connection from the remote client, and creates a new socket associated with the socket address pair of this connection
- **send() and recv(), or write() and read(), or recvfrom() and sendto()**
  - Used for sending and receiving data to/from a remote socket
- **close()**
  - Causes the system to release resources allocated to a socket
  - In case of TCP, the connection is terminated

# Socket API Functions

- The Berkeley socket interface is defined in several header files
  - The names and content of these files differ slightly between implementations
  - In general, they include:
    - **<sys/socket.h>**
      - Core BSD socket functions and data structures
    - **<netinet/in.h>**
      - AF_INET and AF_INET6 address families and their corresponding protocol families PF_INET and PF_INET6
      - Widely used on the Internet, these include IP addresses and TCP and UDP port numbers
    - **<sys/un.h>**
      - PF_UNIX/PF_LOCAL address family
      - Used for local communication between programs running on the same computer
      - Not used on networks
    - **<arpa/inet.h>**
      - Functions for manipulating numeric IP addresses
    - **<netdb.h>**
      - Functions for translating protocol names and host names into numeric addresses
      - Searches local data as well as DNS.

# Κλήση Συστήματος – Socket()

- int socket(int af, int type, int protocol)
  - **family or domain**
    - Το πρώτο όρισμα ορίζει την οικογένεια () στην οποία ανήκει το socket
    - Οι δύο οικογένειες που χρησιμοποιούνται αρκετά συχνά είναι:
    - AF_UNIX για την επικοινωνία μεταξύ διεργασιών στην ίδια μηχανή (χρήση αρχείων)
    - AF_INET για την επικοινωνία μεταξύ διεργασιών που βρίσκονται σε διαφορετικές μηχανές χρησιμοποιώντας τα τυποποιημένα πρωτόκολλα (IP/UDP/TCP)
    - AF_INET6 για IPv6
  - **type**
    - Το δεύτερο όρισμα είναι ο τύπος του socket και έχει να κάνει με το «στυλ» της επικοινωνίας
    - Οι δύο ο συνηθέστερα χρησιμοποιημένες τιμές περιλαμβάνουν:
    - SOCK_STREAM: Μία ροή δεδομένων χωρίς όρια μεγέθους
      - Η παράδοση σε ένα δικτυωμένο περιβάλλον είναι εγγυημένη
      - Εάν η παράδοση είναι αδύνατη, ο αποστολέας λαμβάνει έναν μήνυμα λάθους
    - SOCK_DGRAM: Μια ροή δεδομένων όπου κάθε πακέτο έχει συγκεκριμένο μέγεθος
      - Η παράδοση σε ένα δικτυωμένο περιβάλλον δεν είναι εγγυημένη

# Κλήση Συστήματος – Socket()

- int socket(int af, int type, int protocol)
  - **protocol**
    - Συνήθως protocol = 0
    - Επιτρέπει στο σύστημα να επιλέξει το πρώτο πρωτόκολλο που είναι διαθέσιμο αφού πρώτα έχει οριστεί η οικογένεια και ο τύπος του socket
      - IPPROTO_TCP: για το πρωτόκολλο TCP
      - IPPROTO_UDP: για το πρωτόκολλο UDP2
      - These protocols are specified in <netinet/in.h>
  - Η συνάρτηση επιστρέφει -1 σε περίπτωση λάθους, διαφορετικά τον περιγραφέα της υποδοχής
  - Μια άλλη διεργασία που θέλει να επικοινωνήσει με τη προηγούμενη δεν έχει τρόπο να ονομάσει την υποδοχή, μέχρις ότου η υποδοχή να λάβει διεύθυνση
    - Στο πεδίο UNIX η διεύθυνση συνήθως αποτελείται από ένα ή δύο ονόματα διαδρομής
    - Στο πεδίο Internet η διεύθυνση αποτελείται από ένα συνδυασμό IP διεύθυνσης και αριθμού θύρας (port)

# Κλήση Συστήματος – Bind()

- int bind(int sockfd, struct sockaddr *my_addr, int addrlen)
  - Η πρακτική χρήση της bind, είναι να *δένει* το socket που δημιουργήθηκε, σε μια συγκεκριμένη port
  - **sockfd**
    - Είναι το socket file descriptor (το socket που δημιουργήθηκε)
  - **my_addr**
    - Είναι ένας δείκτης στην δομή sockaddr, η οποία περιέχει πληροφορίες για ένα όνομα διαδρομής ή την local IP address, την port στην οποία θα γίνει bind το socket)
  - **addrlen**
    - Είναι το μέγεθος του sockaddr (χρήση του sizeof(struct sockaddr))
  - Η συνάρτηση επιστρέφει -1 σε περίπτωση λάθους, διαφορετικά 0 σε περίπτωση επιτυχίας

# Παράδειγμα 1: Socket() – Bind()

- Παράδειγμα 1
  - Δημιουργία υποδοχής με αριθμητική IP διεύθυνση
  - Υπάρχουν αρκετές μακροεντολές που περιλαμβάνουν προκαθορισμένες διευθύνσεις, όπως INADDR_LOOPBACK, INADDR_BROADCAST κλπ
  - Η μακροεντολή INADDR_ANY επιπτρέπει στο σύστημα να αποδώσει αυτό οποιαδήποτε εισερχόμενη IP διεύθυνση ώστε να επιτευχθεί σύνδεση με απομακρυσμένο υπολογιστή που αναμένει αίτημα σύνδεσης στη συγκεκριμένη θύρα
    - When receiving, a socket bound to this address receives packets from *all* interfaces
    - When sending, a socket bound with INADDR_ANY binds to the default IP address, which is that of the lowest-numbered interface

# Παράδειγμα 1: Socket() – Bind()

- Παράδειγμα 1
  - Δημιουργία υποδοχής με αριθμητική IP διεύθυνση
  - Οι συναρτήσεις htons(), htonl() χρησιμοποιούνται για να 'μεταφράσουν', αν είναι απαραίτητο, τη σειρά αποθήκευσης των bytes (byte order) ακεραίων (short και long) μεταξύ του οικείου υπολογιστή και του δικτύου (host to network)
    - Αντίστοιχα υπάρχουν και οι συναρτήσεις ntohs(), ntohl() (network to host)
    - Αυτό γίνεται για να εξασφαλιστεί επικοινωνία μεταξύ 'big-endian' και 'long-endian' συστημάτων
    - The **htons** function can be used to convert an IP port number in host byte order to the IP port number in network byte order

# Παράδειγμα 1: Socket() – Bind()

- The adjectives *big-endian* and *little-endian* refer to which bytes are most significant in multi-byte data types and describe the order in which a sequence of bytes is stored in a computer's memory
  - In a big-endian system, the most significant value in the sequence is stored at the lowest storage address (i.e., first)
  - In a little-endian system, the least significant value in the sequence is stored first
  - For example, consider the number 1025 (2 to the tenth power plus one) stored in a 4-byte integer:

00000000 00000000 00000100 00000001

| Address | Big-Endian representation of 1025 | Little-Endian representation of 1025 |
|---|---|---|
| 00 | 00000000 | 00000001 |
| 01 | 00000000 | 00000100 |
| 02 | 00000100 | 00000000 |
| 03 | 00000001 | 00000000 |

- Who is who
  - Many mainframe computers, particularly IBM mainframes, use a big-endian architecture
  - The PowerPC system is *bi-endian* because it can understand both systems
  - Most modern computers, including PCs, use the little-endian system (LSB first)
  - The network byte order, as used on the Internet big-endian (MSB first)

# Παράδειγμα 1: Socket() – Bind()

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>

int make_socket (unsigned short int port)
{
        int sock;
        struct sockaddr_in name;

        /* Create the socket. */
        sock = socket (PF_INET, SOCK_STREAM, 0);
        if (sock < 0)
        {
                perror ("socket");
                exit (EXIT_FAILURE);
        }
        /* Give the socket a name. */
        name.sin_family = AF_INET;
        name.sin_port = htons (port);
        name.sin_addr.s_addr = htonl (INADDR_ANY);
        if (bind (sock, (struct sockaddr *) &name, sizeof (name)) < 0)
        {
                perror ("bind");
                exit (EXIT_FAILURE);
        }
        return sock;
        }
```

```
/*
 * Socket address, internet style.
 */
struct sockaddr_in {
        u_char  sin_len;
        u_char  sin_family;
        u_short sin_port;
        struct  in_addr sin_addr;
        char    sin_zero[8];
};
```

This structure contains the Internet address in network byte order.

```
struct in_addr {
   u_long s_addr;
};
```

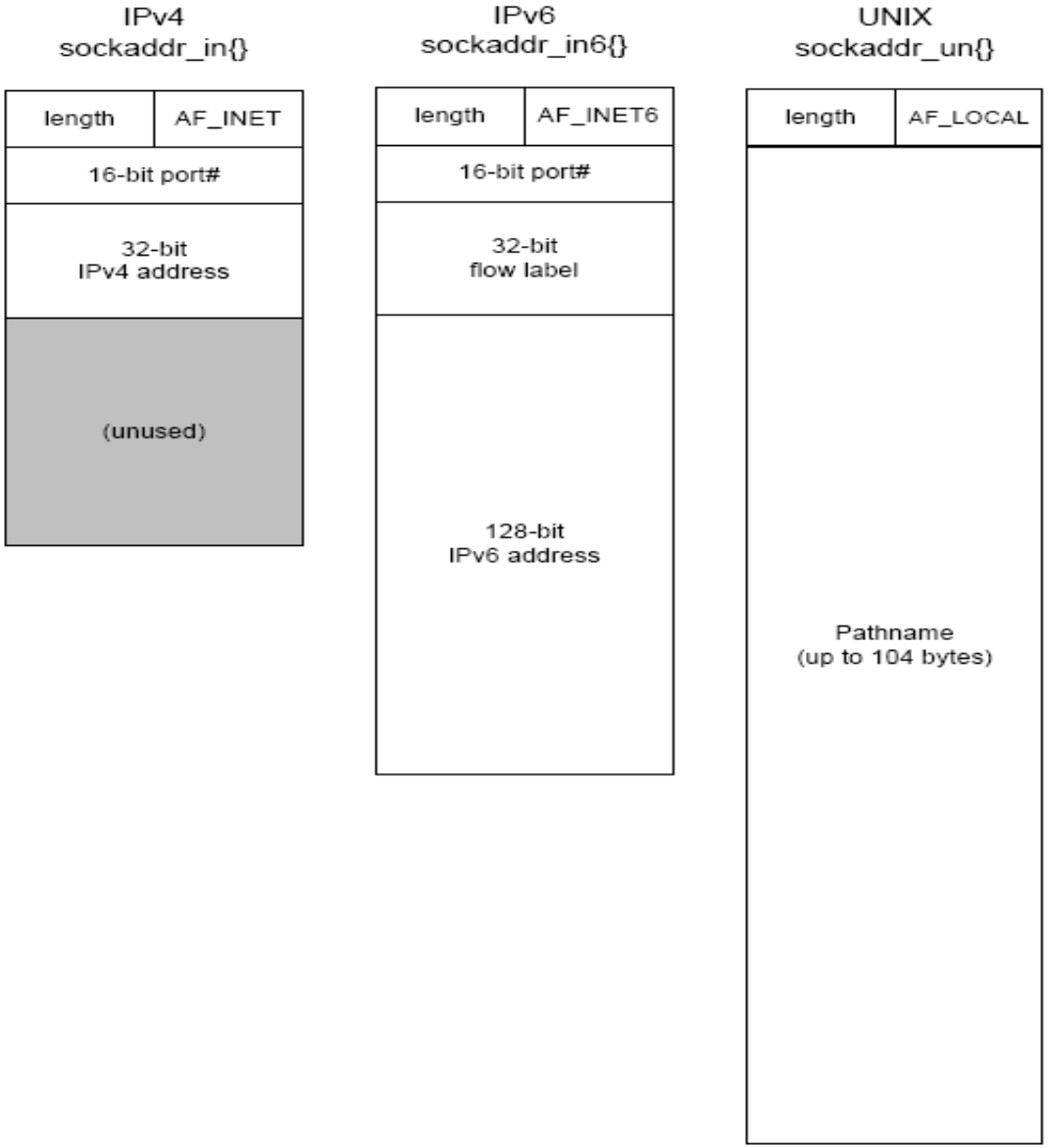
# Παράδειγμα 1: Socket() – Bind()

**2.2 IPv4 socket address structures**

```
struct sockaddr_in {
        sa_fimily_t         sin_family;     /*Address Family – sys/socket.h*/
        uint16_t            sin_port;       /*Port Number –
                                              unsigned 16 bit integer (sys/types.h)*/
        struct in_addr      sin_addr;       /*Internet Address*/
        unsigned char       sin_zero[8];    /*Pad Bytes*/
};

struct in_addr {
        uint32_t            s_addr;         /*Internet Address*/
};
```

Η *sin_family* παίρνει τιμή *AF_INET*
Η *sin_port* καθορίζει τη θύρα του socket (σε network byte order)
Η *s_addr* περιέχει την IP διεύθυνση (σε network byte order)

# Παράδειγμα 1: Socket() – Bind()



# Παράδειγμα 2: Socket() – Bind()

- Παράδειγμα 2
  - Δημιουργία υποδοχής με χρήση συναρτήσεων και δηλώσεων που συλλέγουν πληροφορίες για τον υπολογιστή μέ βάση το όνομα hostname
    - Το όνομα hostname χρησιμοποιείται ως κλειδή αναζήτησης πληροφοριών σχετικά με τον υπολογιστή σε μια 'βάση δεδομένων υπολογιστών' (host database)
    - Η απλούστερη host database είναι το αρχείο /etc/hosts που διατηρεί κάθε σύστημα UNIX/Linux
  - Η συνάρτηση gethostbyname() αναζητά πληροφορίες για τον υπολογιστή με το όνομα hostname στη βάση δεδομένων και τις αποθηκεύει στη δομή hostinfo
    - Από τη δομή hostinfo αντιγράφουμε στη δομή sockaddr την IP διεύθυνση του υπολογιστή
    - The **gethostbyname()** function returns a structure of type *hostent* for the given host *name*. Here *name* is either a host name, or an IPv4 address in standard dot notation, or an IPv6 address in colon (and possibly dot) notation. If *name* is an IPv4 or IPv6 address, no lookup is performed and **gethostbyname**() simply copies *name* into the *h_name* field and its *struct in_addr* equivalent into the *h_addr_list[0]* field of the returned *hostent* structure
  - Αντίστοιχη διαδικασία μπορεί να υλοποιηθεί και για την τιμή port, η οποία θα μπορούσε να εξαχθεί από μια βάση δεδομένων υπηρεσιών (service database) έτσι ώστε η θύρα λαμβάνει τη τιμή που αντιστοιχεί στο όνομα της αντιστοιχης υπηρεσίας
    - Η συνηθισμένη βάση δεδομένων είναι το αρχείο /etc/services

# Παράδειγμα 2: Socket() – Bind()

This structure contains host information.

```
struct hostent {
   char    *h_name;        /* official name of host                */
   char    **h_aliases;    /* alias list                           */
   int     h_addrtype;     /* host address type                    */
   int     h_length;       /* length of address                    */
   char    **h_addr_list;  /* list of addresses from name server   */
 #define h_addr h_addr_list[0]  /* address, for backward           */
                                /* compatiblity                    */
};
```

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

void init_sockaddr (struct sockaddr_in *name,
                    const char *hostname,
                    unsigned short int port)
{
    struct hostent *hostinfo;

    name->sin_family = AF_INET;
    name->sin_port = htons (port);

    hostinfo = gethostbyname (hostname);

    if (hostinfo == NULL)
    {
        fprintf (stderr, "Unknown host %s.\n", hostname);
        exit (EXIT_FAILURE);
    }

    name->sin_addr = *(struct in_addr *) hostinfo->h_addr;
}
```

# Σύνδεση Υποδοχών Ροής

- Συνήθως η σύνδεση μέσω υποδοχών δεν είναι συμμετρική
  - Η μια διεργασία λειτουργεί ως διακομιστής και η άλλη ως πελάτης
  - Ο διακομιστής συνδέει την υποδοχή του σε προσυμφωνημένο όνομα διαδρομής ή θύρα
  - Στη συνέχεια αναμένει αιτήσεις επικοινωνίας άλλων διεργασιών μέσω της υποδοχής
  - Στη περίπτωση υποδοχών ροής (SOCK_STREAM), ο διακομιστής καλεί τη συνάρτηση **listen**()
  - Ο πελάτης συνδέεται στην υποδοχή του διακομιστή με τη συνάρτηση **connect**() που έχει σύνταξη και λειτουργία αντίστοιχη με τη συνάρτηση bind()
  - Στη περίπτωση που ο τύπος της υποδοχής είναι SOCK_STREAM ο διακομιστής απαντά στο αίτημα connect() με τη συνάρτηση **accept**() ώστε να ολοκληρωθεί η επικοινωνία

# Κλήση Συστήματος – Listen()

- int listen(int s, int backlog)
  - Η listen() καθιερώνει το socket ως το παθητικό άκρο μίας σύνδεσης
  - Δεν αναστέλλει την εκτέλεση της διεργασίας
  - Κανένα μήνυμα δεν μπορεί να σταλεί μέσω αυτού του socket
  - Τα εισερχόμενα μηνύματα μπορούν να παραληφθούν
  - **s**
    - Είναι ο file descriptor που συνδέεται με το socket που δημιουργείται χρησιμοποιώντας την socket()
  - **backlog**
    - είναι το μέγεθος της ουράς αναμονής των αιτήσεων που περιμένουν ενώ ο εξυπηρετητής είναι απασχολημένος με την εξυπηρέτηση μίας αίτησης
    - Η συνήθης τιμή είναι 5
    - Once a connection is accepted, it is dequeued
  - 0 επιστρέφεται στην επιτυχία, -1 στο λάθος με το errno να δείχνει το πρόβλημα

# Κλήση Συστήματος – Connect()

- int connect(int sockfd, struct sockaddr *serv_addr, int addrlen)
  - Η connect, εγκαθιδρύει μια σύνδεση με ένα απομακρυσμένο host
  - **sockfd**
    - Είναι το *τοπικό* socket file descriptor
  - **serv_addr**
    - Είναι η *διεύθυνση* του απομακρυσμένου host που θέλουμε να συνδεθούμε
  - **Addrlen**
    - Το μέγεθος της διεύθυνσης
  - connect() returns an integer representing the error code: 0 represents success, while -1 represents an error

# Κλήση Συστήματος – Accept()

- int accept(int sockfd, struct sockaddr *name, int *namelen)
  - Η κλήση της accept() εγκαθιστά μια σύνδεση πελάτη-εξυπηρετητή από την πλευρά του εξυπηρετητή
  - Ο πελάτης ζητά τη χρησιμοποίηση της σύνδεσης με την κλήση της connect()
  - Ο εξυπηρετητής πρέπει να έχει δημιουργήσει το socket χρησιμοποιώντας την socket(), να έχει δώσει ένα όνομα στο socket με την bind(), και να την ενεργοποιήσει περιμένοντας νέες αιτήσεις με την listen()
  - **sockfd**
    - Είναι ο file descriptor για το socket ο οποίος επιστρέφεται από την socket()
  - **name**
    - Είναι ένας δείκτης σε μια δομή του τύπου sockaddr
  - Η κλήση της accept() επιστρέφει -1 σε περίπτωση λάθους και ένα **νέο** περιγραφέα υποδοχής, που αναφέρεται μόνο στη συγκεκριμένη σύνδεση
    - Ένας διακομιστής μπορεί να έχει ταυτόχρονα πολλαπλές συνδέσεις SOCK_STREAM σε μια υποδοχή
    - All further communication with the remote host now occurs via this new socket
    - Datagram sockets do not require processing by accept() since the receiver may immediately respond to the request using the listening socket

# Κλήση Συστήματος – Recvfrom()

- Ανάλογα με το είδος του socket που χρησιμοποιείται, υπάρχουν διαφορετικές εντολές για αποστολή και λήψη δεδομένων

  - Για stream sockets
    - int send(int sockfd, const void *msg, int len, int flags);
    - int recv(int sockfd, void *buf, int len, unsigned int flags);

  - Για datagram sockets
    - int sendto(int sockfd, const void *msg, int len, unsigned int flags, const struct sockaddr *to, int tolen);
    - int recvfrom(int sockfd, void *buf, int len, unsigned int flags, struct sockaddr *from, int *fromlen);

# Κλήση Συστήματος – Sendto()

- int sendto(int s, char *msg, int len, int flags, struct sockaddr *to, int tolen)
  - Αυτή η λειτουργία επιτρέπει να αποσταλεί ένα μήνυμα msg μήκους len σε ένα socket με file descriptor s το οποίο περιγράφεται με το to και tolen, όπου tolen είναι το πραγματικό μήκος το to
  - Τα flags προς το παρόν θα είναι 0
  - Ο αριθμός χαρακτήρων που στέλνονται είναι ο αριθμός που επιστρέφει η συνάρτηση
  - Όταν υπάρξει κάποιο πρόβλημα, -1 επιστρέφεται και το errno περιγράφει το λάθος

# Κλήση Συστήματος – Recvfrom()

- int recvfrom(int s, char *msg, int len, int flags, struct sockaddr *from, int *fromlen)
  - Αυτή η συνάρτηση επιτρέπει σε ένα μήνυμα msg με μέγιστο μήκος len να διαβαστεί από ένα socket με file descriptor s το οποίο περιγράφεται με το from και fromlen, όπου fromlen είναι το πραγματικό μήκος το from
  - Ο αριθμός χαρακτήρων που διαβάζονται πραγματικά από την υποδοχή είναι και ο αριθμός που επιστρεφεται από την συνάρτηση
  - Όταν υπάρξει κάποιο λάθος, -1 επιστρέφεται και το errno περιγράφει το λάθος
  - Τα flags μπορεί να είναι 0, ή μπορεί να είναι MSG_PEEK για να εξετάσει ένα μήνυμα χωρίς πραγματικά να το λάβει από τη ουρά αναμονής
  - Εάν κανένα μήνυμα δεν είναι διαθέσιμο για να διαβαστεί, η διαδικασία θα μπλοκάρει περιμένοντας ένα, εκτός και αν το socket έχει οριστεί με nonblocking συμπεριφορά
  - Η συνάρτηση read() μπορεί επίσης να χρησιμοποιηθεί για να διαβάσουμε δεδομένα από ένα socket

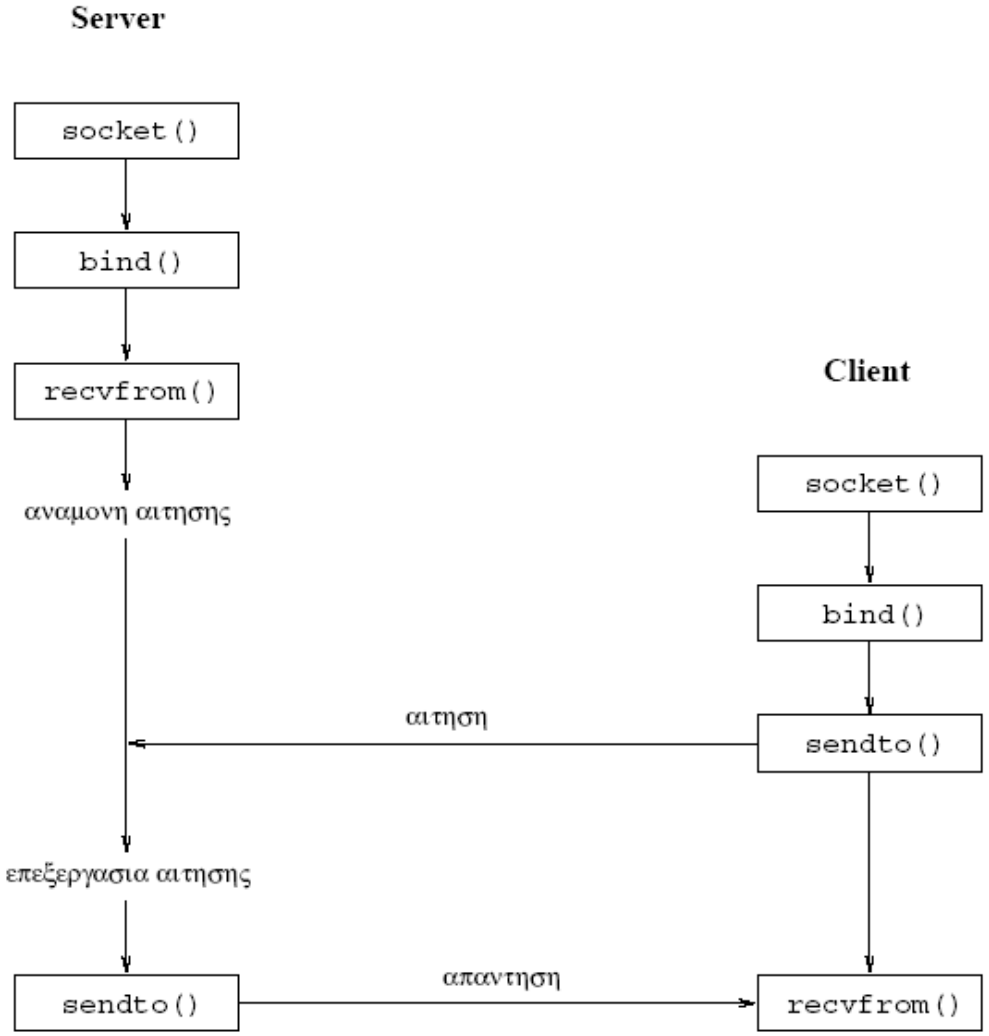
# Κλήση Συστήματος – Close()

- int close(sockfd)
  - The operating system does not release the resources allocated to a socket until a close() call occurs on the socket descriptor
  - This is especially important if the connect() call fails and may be retried
  - Each successful call to socket() must have a matching call to close() in all possible execution paths
  - The header file <unistd.h> defines the close function
  - When the close() system call is initiated by an application, only the interface to the socket is destroyed, not the socket itself
  - It is the kernel's responsibility to destroy the socket internally
  - Sometimes, a socket may enter a TIME_WAIT state, on the server side, for up to 4 minutes

# UDP Επικοινωνία client/server

- Τα πρωτόκολλα επικοινωνίας χωρίς σύνδεση (Connectionless Protocols) επιτρέπουν την αποστολή δεδομένων χωρίς να είναι απαραίτητη η σύνδεση πριν την αποστολή των δεδομένων
  - Το πρωτόκολλο UDP (User Datagram Protocol- Πρωτόκολλο Αυτοδύναμο Πακέτου Χρήστη) είναι ένα πρωτόκολλο επικοινωνίας χωρίς σύνδεση
  - Σε ένα connectionless πρωτόκολλο και τα δύο sockets πρέπει να γνωστοποιήσουν την ύπαρξή τους στο σύστημα χρησιμοποιώντας την “bind”
  - Όταν καλείται η “bind” συνδέεται με ένα νέο port – δεν μπορεί να συνδεθεί με ένα που ήδη χρησιμοποιείται
  - Εάν θέσουμε το port = 0 τότε το σύστημα δίνει ένα αχρησιμοποίητο μέχρι εκείνη τη στιγμή port
  - Για να γράψουμε ή να διαβάσουμε δεδομένα χρησιμοποιούμε τις συναρτήσεις recvform/sendto και όχι τις read/write

# UDP Επικοινωνία client/server



# UDP Server.c

```
/* udpServer.c */
#include <sys/socket.h>

#define LOCAL_SERVER_PORT 1500
#define MAX_MSG 100

int main(int argc, char *argv[]) {
 int sd, rc, n, cliLen;
 struct sockaddr_in cliAddr, servAddr;
 char msg[MAX_MSG];

 /* socket creation */
 sd=socket(AF_INET, SOCK_DGRAM, 0);

 /* bind local server port */
 servAddr.sin_family = AF_INET;
 servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
 servAddr.sin_port = htons(LOCAL_SERVER_PORT);
 rc = bind (sd, (struct sockaddr *) &servAddr,sizeof(servAddr));
 if(rc<0) {
  printf("%s: cannot bind port number %d \n",
         argv[0], LOCAL_SERVER_PORT);
  exit(1);
 }
```

# UDP Server.c

```
printf("%s: waiting for data on port UDP %u\n", argv[0],LOCAL_SERVER_PORT);

/* server infinite loop */
while(1) {

  /* init buffer */
  memset(msg,0x0,MAX_MSG);

  /* receive message */
  cliLen = sizeof(cliAddr);
  n = recvfrom(sd, msg, MAX_MSG, 0, (struct sockaddr *) &cliAddr, &cliLen);
  if(n<0) {
    printf("%s: cannot receive data \n",argv[0]);
    continue;
  }
  /* print received message */
  printf("%s: from %s:UDP%u : %s \n", argv[0],inet_ntoa(cliAddr.sin_addr),
          ntohs(cliAddr.sin_port),msg);
 }/* end of server infinite loop */
return 0;
}
```

# UDP Client.c

```
/* udpClient.c */
#include <sys/socket.h>

#define REMOTE_SERVER_PORT 1500
#define MAX_MSG 100

int main(int argc, char *argv[]) {
 int sd, rc, i;
 struct sockaddr_in cliAddr, remoteServAddr;
 struct hostent *h;

 /* check command line args */
 if(argc<3) {
  printf("usage : %s <server> <data1> ... <dataN> \n", argv[0]);
  exit(1);
 }
 /* get server IP address (no check if input is IP address or DNS name */
 h = gethostbyname(argv[1]);
 if(h==NULL) {
  printf("%s: unknown host '%s' \n", argv[0], argv[1]);
  exit(1);
 }
 printf("%s: sending data to '%s' (IP : %s) \n", argv[0], h->h_name,
        inet_ntoa(*(struct in_addr *)h->h_addr_list[0]));
```

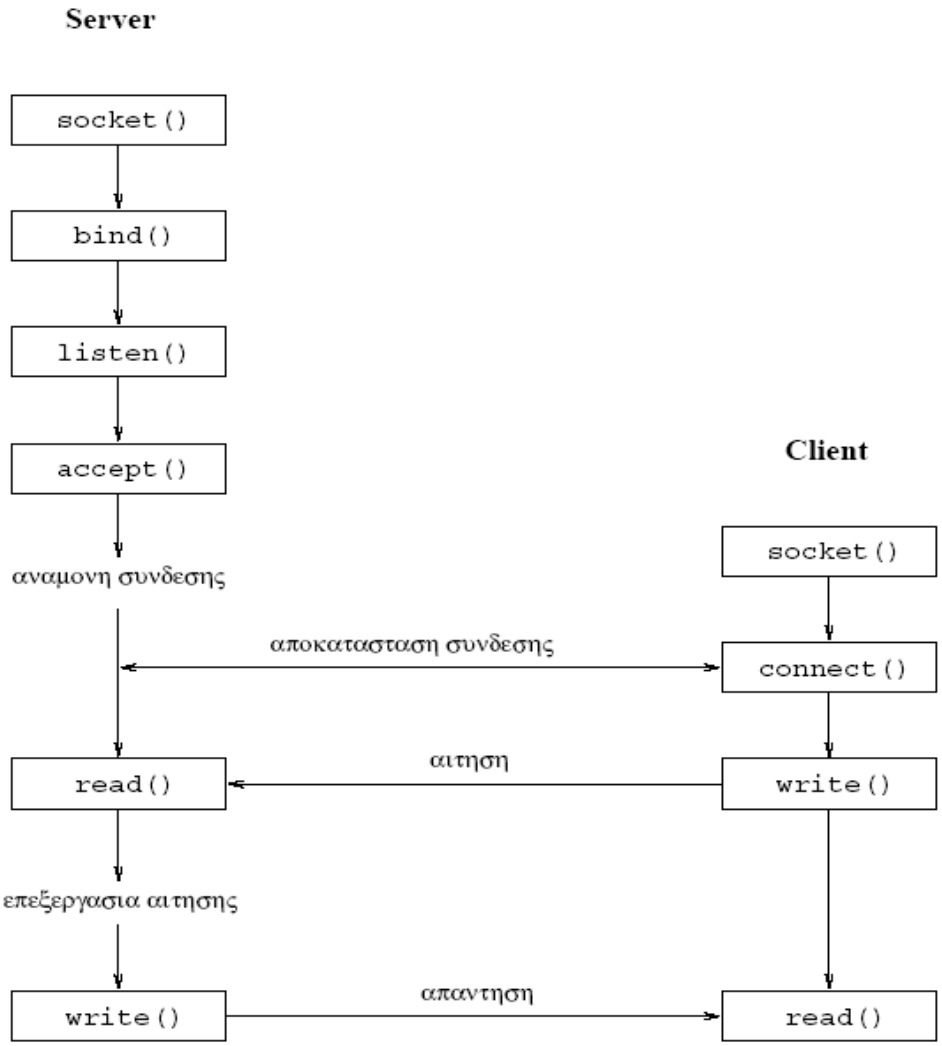
# UDP Client.c

```
remoteServAddr.sin_family = h->h_addrtype;
memcpy((char *) &remoteServAddr.sin_addr.s_addr,
        h->h_addr_list[0], h->h_length);
remoteServAddr.sin_port = htons(REMOTE_SERVER_PORT);

/* socket creation */
sd = socket(AF_INET,SOCK_DGRAM,0);

/* bind any port */
cliAddr.sin_family = AF_INET;
cliAddr.sin_addr.s_addr = htonl(INADDR_ANY);
cliAddr.sin_port = htons(0);
rc = bind(sd, (struct sockaddr *) &cliAddr, sizeof(cliAddr));

/* send data */
for(i=2;i<argc;i++) {
  rc = sendto(sd, argv[i], strlen(argv[i])+1, 0,
              (struct sockaddr *) &remoteServAddr, sizeof(remoteServAddr));
}
return 1;
}
```

# TCP Επικοινωνία client/server

- Ο server κάνει το socket του γνωστό στο σύστημα χρησιμοποιώντας την “bind”
- Αυτό βοηθάει στην εύρεση του socket από κάποιο άλλο socket
- Ο server ακούει (“listen”) σε αυτό το socket για να δεχτεί (“accept”) εισερχόμενα μηνύματα
- Ο πελάτης (client) δημιουργεί μία δικτυακή σύνδεση στο αντίστοιχο socket και οι δύο διαδικασίες (client- server) αρχίζουν την ανταλλαγή μηνυμάτων
- Η ανταλλαγή μηνυμάτων είναι χωρίς όριο και γίνεται και προς τις δύο κατευθύνσεις

# TCP Επικοινωνία client/server



# Συναρτήσεις Μετατροπής Διευθύνσεων

- inet_ntoa(), inet_aton(), inet_addr
  - All of these functions convert IP addresses from a struct in_addr to a string in dots-and-numbers format (e.g. "192.168.5.10") and vice-versa
    - struct in_addr is part of your struct sockaddr_in, most likely
    - If you have an IP address passed on the command line this is the easiest way to get a struct in_addr to connect() to
  - The function inet_ntoa() converts a network address in a struct in_addr to a dots-and-numbers format string
    - The "n" in "ntoa" stands for network, and the "a" stands for ASCII for historical reasons
  - The function inet_aton() is the opposite, converting from a dots-and-numbers string into a in_addr_t (which is the type of the field s_addr in your struct in_addr)
  - Finally, the function inet_addr() is an older function that does basically the same thing as inet_aton()
    - It's theoretically deprecated, but you'll see it a lot and the police won't come get you if you use it
  - *These functions are deprecated because they don't handle IPv6!*
    - *Use inet_ntop() or inet_pton() instead!*
    - *They are included here because they can still be found in the wild*

# Συναρτήσεις Μετατροπής Διευθύνσεων

**unsigned long inet_addr(const char *string);**

| | |
|---|---|
| Τιμή που επιστρέφει κατά την επιτυχή εκτέλεση | 32-bit διεύθυνση internet σε network byte order |
| Τιμή που επιστρέφει σε περίπτωση λάθους | INADDR_NONE |
| Παράμετροι | Διεύθυνση internet στην μορφή xxx.xxx.xxx.xxx (string) |

Η συνάρτηση *inet_addr* μετατρέπει ένα string της μορφής xxx.xxx.xxx.xxx σε μία 32-bit internet διεύθυνση.

# Συναρτήσεις Μετατροπής Διευθύνσεων

**int inet_aton(const char *string, struct in_addr *addr);**

| | |
|---|---|
| Τιμή που επιστρέφει κατά την επιτυχή εκτέλεση | 1 |
| Τιμή που επιστρέφει σε περίπτωση λάθους | 0 |
| Παράμετροι | 1. Διεύθυνση internet στην μορφή xxx.xxx.xxx.xxx (string)<br>2. Το στοιχείο της δομής (structure) στο οποίο θα αποθηκευτεί η διεύθυνση IP σε μορφή network byte order. |

Η συνάρτηση *inet_aton* μετατρέπει ένα string της μορφής xxx.xxx.xxx.xxx σε μία 32-bit internet διεύθυνση.

**char *inet_ntoa(struct in_addr addr);**

| | |
|---|---|
| Τιμή που επιστρέφει κατά την επιτυχή εκτέλεση | 1 |
| Τιμή που επιστρέφει σε περίπτωση λάθους | 0 |
| Παράμετροι | 1. Structure με την διεύθυνση internet στην μορφή xxx.xxx.xxx.xxx (string) |

Η συνάρτηση *inet_ntoa* μετατρέπει μία 32-bit τιμή της μορφής network byte order σε ένα string της μορφής xxx.xxx.xxx.xxx.

# Συναρτήσεις Ανάκτησης Ονομάτων

**int gethostname(char *name, size_t len);**

| | |
|---|---|
| Τιμή που επιστρέφει κατά την επιτυχή εκτέλεση | 0 |
| Τιμή που επιστρέφει σε περίπτωση λάθους | -1 |
| Παράμετροι | 1. Ο buffer name που θα περιέχει το hostname.<br>2. Το μέγιστο μήκος του buffer name (Το μήκος του hostname μαζί με το null byte.) |

Η συνάρτηση *gethostname* επιστρέφει το hostname του Η/Υ στον οποίο εργαζόμαστε.

**int getdomainname(char *name,size_t len);**

| | |
|---|---|
| Τιμή που επιστρέφει κατά την επιτυχή εκτέλεση | 0 |
| Τιμή που επιστρέφει σε περίπτωση λάθους | -1 |
| Παράμετροι | 3. Ο buffer name που θα περιέχει το domain name<br>4. Το μέγιστο μήκος του buffer name (Το μήκος του domain name μαζί με το null byte.) |

Η συνάρτηση *getdomainname* επιστρέφει το NIS domain name του Η/Υ στον οποίο εργαζόμαστε.

# Συναρτήσεις Ανάκτησης Ονομάτων

- The gethostbyname() and gethostbyaddr() functions
  - These functions are not strictly a component of the BSD socket API, but are often used in conjunction with the API functions
  - For instance
    - If you have "www.example.com", you can use **gethostbyname()** to get its IP address and store it in a struct in_addr
    - Conversely, if you have a struct in_addr or a struct in6_addr, you can use **gethostbyaddr()** to get the hostname back
  - Furthermore, these functions are now considered legacy interfaces for querying the domain name system
    - New functions that are completely protocol-agnostic have been defined
    - These new function are getaddrinfo() and getnameinfo(), and are based on a new *addrinfo* data structure
    - In particular, **gethostbyname()** doesn't work well with IPv6
    - **gethostbyaddr()** *is* IPv6 compatible, but you should use the newer shinier **getnameinfo()** instead

# Συναρτήσεις Ανάκτησης Ονομάτων

- The gethostbyname() and gethostbyaddr() functions
  - Used to resolve host names and addresses in the domain name system or the local host's other resolver mechanisms (e.g., /etc/hosts lookup)
  - They return a pointer to an object of type *struct hostent*, which describes an Internet Protocol host
  - The functions take the following arguments:
    - ***name*** specifies the name of the host. For example: www.wikipedia.org
    - ***addr*** specifies a pointer to a *struct in_addr* containing the address of the host.
    - ***len*** specifies the length, in bytes, of *addr*.
    - ***type*** specifies the address family type (e.g., AF_INET) of the host address.
  - The functions return a NULL pointer in case of error, in which case the external integer *h_errno* may be checked to see whether this is a temporary failure or an invalid or unknown host
  - Otherwise a valid *struct hostent* * is returned

# Συναρτήσεις Ανάκτησης Ονομάτων

**struct hostent *gethostbyname(const char *name);**

| | |
|---|---|
| Τιμή που επιστρέφει κατά την επιτυχή εκτέλεση | Δείκτης (pointer) στην δομή hostent. |
| Τιμή που επιστρέφει σε περίπτωση λάθους | Null pointer |
| Παράμετροι | Ένα string που αντιπροσωπεύει το όνομα του Η/Υ |

Η συνάρτηση *gethostbyname* επιστρέφει μία δομή (structure) *hostent* σχετικά με τον Η/Υ με όνομα *name*. Η μορφή της δομής παρουσιάζεται παρακάτω

**struct hostent *gethostbyaddr(const char *addr, int len, int type);**

| | |
|---|---|
| Τιμή που επιστρέφει κατά την επιτυχή εκτέλεση | Δείκτης (pointer) στην δομή hostent. |
| Τιμή που επιστρέφει σε περίπτωση λάθους | Null pointer |
| Παράμετροι | 1. Διεύθυνση IP του host. Για διευθύνσεις τύπου AF_INET είναι ο δείκτης στο μέλος sin_addr.<br>2. Το μήκος της διεύθυνσης (Για AF_INET είναι 4 byte ενώ για AF_INET6 είναι 6 byte)<br>3. Ο τύπος της διεύθυνσης |

Η συνάρτηση *gethostbyaddr* επιστρέφει μία δομή (structure) *hostent* σχετικά με τον Η/Υ με IP *addr.

# Συναρτήσεις Ανάκτησης Ονομάτων

- The **getaddrinfo()** and **getnameinfo()** functions
  - Part of the POSIX standard application programming interface (API) for converting domain name system (DNS) hostnames and IP addresses between their human-readable text representations and structured binary formats for the operating system's networking API
    - They are inverse functions of each other
  - This set of functions is fully network protocol agnostic and supports both IPv4 and IPv6
    - It is the recommended interface for name resolution in building protocol independent applications and for transitioning legacy IPv4 code to the IPv6 Internet
  - getaddrinfo()
    - Converts human-readable text strings representing hostnames or IP addresses into a dynamically allocated linked list of struct addrinfo structures
  - getnameinfo()
    - Converts the internal binary representation of an IP address in the form of a struct sockaddr pointer into text strings consisting of the hostname or, if the address cannot be resolved into a name, a textual IP address representation, as well as the service port name or number

# Συναρτήσεις Ανάκτησης Ονομάτων

- The **getaddrinfo()** and **getnameinfo()** functions
  - The C data structure used to represent addresses and hostnames within the networking API is the following:

```
struct addrinfo {
    int     ai_flags;
    int     ai_family;
    int     ai_socktype;
    int     ai_protocol;
    size_t  ai_addrlen;
    struct  sockaddr *ai_addr;
    char    *ai_canonname;      /* canonical name */
    struct  addrinfo *ai_next; /* this struct can form a linked list */
};
```