Topic 5: Sockets*

K24: Systems Programming

Instructor: Mema Roussopoulou

Επικοινωνία διεργασιών σε διαφορετικούς υπολογιστές

- Έχουμε δει (ή θα δούμε) διαδιεργασιακή επικοινωνία στο ίδιο μηχάνημα
  - Μέσω αρχείων
  - Σήματα
  - Αγωγοί
  - Κοινόχρηστη μνήμη
  - ...
- Τι γίνεται αν ενδιάμεσα υπάρχει ένα δίκτυο;

Επικοινωνία διεργασιών σε διαφορετικούς υπολογιστές

- Συνήθως μοντέλο πελάτη-εξυπηρετητή (client-server)
- Ο εξυπηρετητής παρέχει κάποια υπηρεσία
- Ο εξυπηρετητής περιμένει να δεχτεί συνδέσεις από πελάτες
- Πολλοί πελάτες συνδέονται για να χρησιμοποιήσουν την υπηρεσία
- Πελάτης-εξυπηρετητής μιλάνε απευθείας

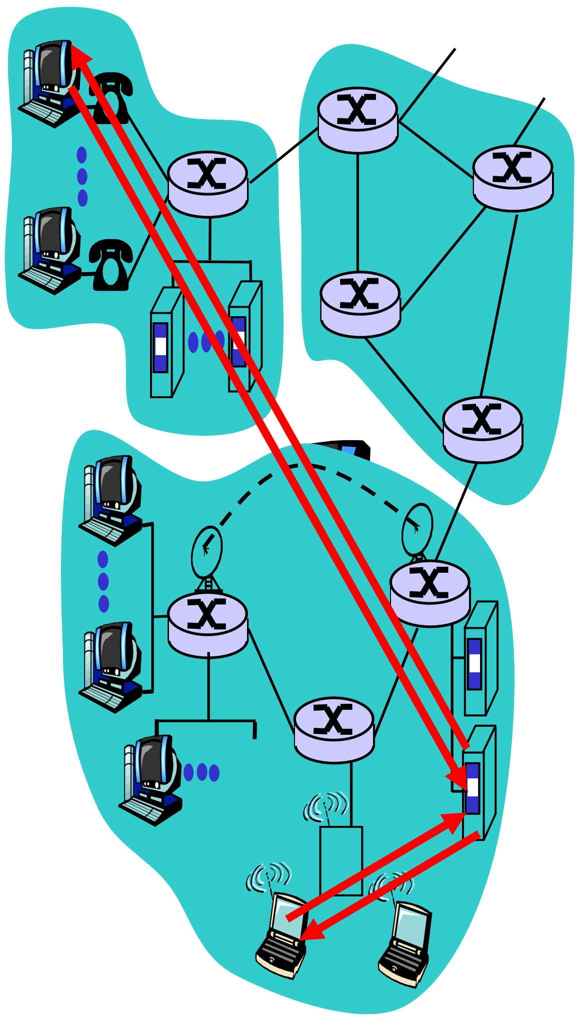
Μερικές εφαρμογές/ υπηρεσίες δικτύου

- E-mail
- Web
- Instant messaging
- Remote login
- P2P file sharing
- Multi-user network games
- Streaming stored video clips
- Internet telephone
- Real-time video conference
- Massive parallel computing
-
-
-

Μοντέλα Επικοινωνίας

- Client-server
- Peer-to-peer (P2P)
- Hybrid of client-server and P2P

Client-server architecture



server:

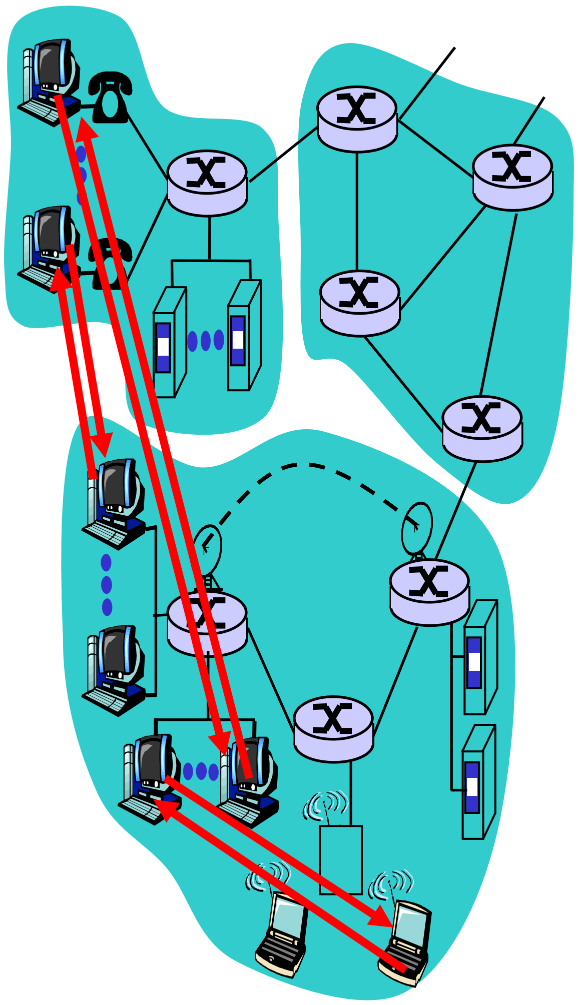
- always-on host
- permanent IP address
- server farms for scaling

clients:

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

Pure P2P architecture

- no always-on server
- arbitrary end systems directly communicate
- each node (peer) acts as both a server and a client
- peers are intermittently connected and change IP addresses
- example: Gnutella



Highly scalable but difficult to manage

Hybrid of client-server and P2P

Skype

- Internet telephony app
- Finding address of remote party: centralized server(s)
- Client-client connection is direct (not through server)

Instant messaging

- Chatting between two users is P2P
- Presence detection/location centralized:
  - User registers its IP address with central server when it comes online
  - User contacts central server to find IP addresses of buddies

Processes communicating

Process: program running within a host.

- within same host, two processes communicate using inter-process communication (defined by OS).
- processes in different hosts communicate by exchanging messages

Client process: process that initiates communication

Server process: process that waits to be contacted

- Note: applications with P2P architectures have client processes & server processes

Transport Services Offered by OS

- OS deals with host to host data transfer
  - Did the data get there?
  - What to do when the data gets there?
- Supplies the application layer with a socket API (connect, send, receive)
- When data arrives at a host from the net, OS determines which application process is to receive the data (via socket)
- Applications typically use one of
  - TCP
  - UDP

What transport service does an application need?

Data loss

- some apps (e.g., audio) can tolerate some loss
- other apps (e.g., file transfer, telnet) require 100% reliable data transfer

Timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

Bandwidth

- some apps (e.g., multimedia) require minimum amount of bandwidth to be "effective"
- other apps ("elastic apps") make use of whatever bandwidth they get

OS Transport Services (TCP and UDP)

TCP service:

- *connection-oriented:* setup required between client and server processes
- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum bandwidth guarantees

UDP service:

- unreliable data transfer between sending and receiving process
- does not provide: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee

Q: why bother? Why is there a UDP?

TCP versus UDP

Applications using TCP:

- HTTP (WWW), FTP (file transfer), Telnet (remote login), SMTP (email)

Applications using UDP:

- streaming media, teleconferencing, DNS, Internet telephony

Πως θα διαλέγατε ανάμεσα στο TCP & UDP?

Processes communicating

Process: program running within a host.

- within same host, two processes communicate using inter-process communication (defined by OS).
- processes in different hosts communicate by exchanging messages

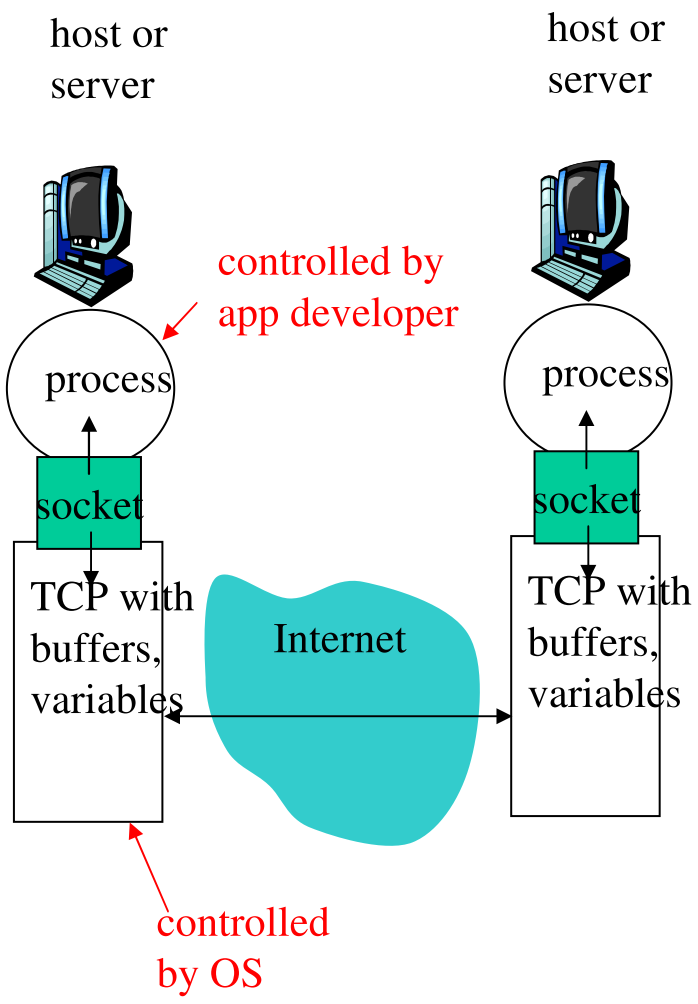
Client process: process that initiates communication

Server process: process that waits to be contacted

- Note: applications with P2P architectures have client processes & server processes

Sockets (Υποδοχές)

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door which brings message to socket at receiving process


host or server
host or server
controlled by app developer
process
process
socket
socket
TCP with buffers, variables
TCP with buffers, variables
Internet
controlled by OS

- API: (1) choice of transport service OS will use; (2) ability to fix a few parameters

Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- Q: does IP address of host on which process runs suffice for identifying the process?

Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- Q: does IP address of host on which process runs suffice for identifying the process?
  - Answer: NO, many processes can be running on same host
- *identifier* includes both IP address and port numbers associated with process on host.
- Example port numbers:
  - HTTP server: 80
  - Mail server: 25
- to send HTTP message to www.di.uoa.gr web server:
  - IP address: 147.52.17.2
  - Port number: 80

Application protocol defines

- Types of messages exchanged,
  - e.g., request, response
- Message syntax:
  - what fields in messages & how fields are delineated
- Message semantics
  - meaning of information in fields
- Rules for when and how processes send & respond to messages

Public-domain protocols:

- defined in RFCs
- allows for interoperability
- e.g., HTTP, SMTP

Proprietary protocols:

- e.g., Skype, KaZaA, etc.

Socket programming

Goal: learn how to build client/server application programs that communicate using sockets

Socket API

- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by applications
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram (UDP)
  - reliable, byte stream-oriented (TCP)

socket

a *host-local*, *application-created*, *OS-controlled* interface (a "door") into which application process can both send and receive messages to/from another application process

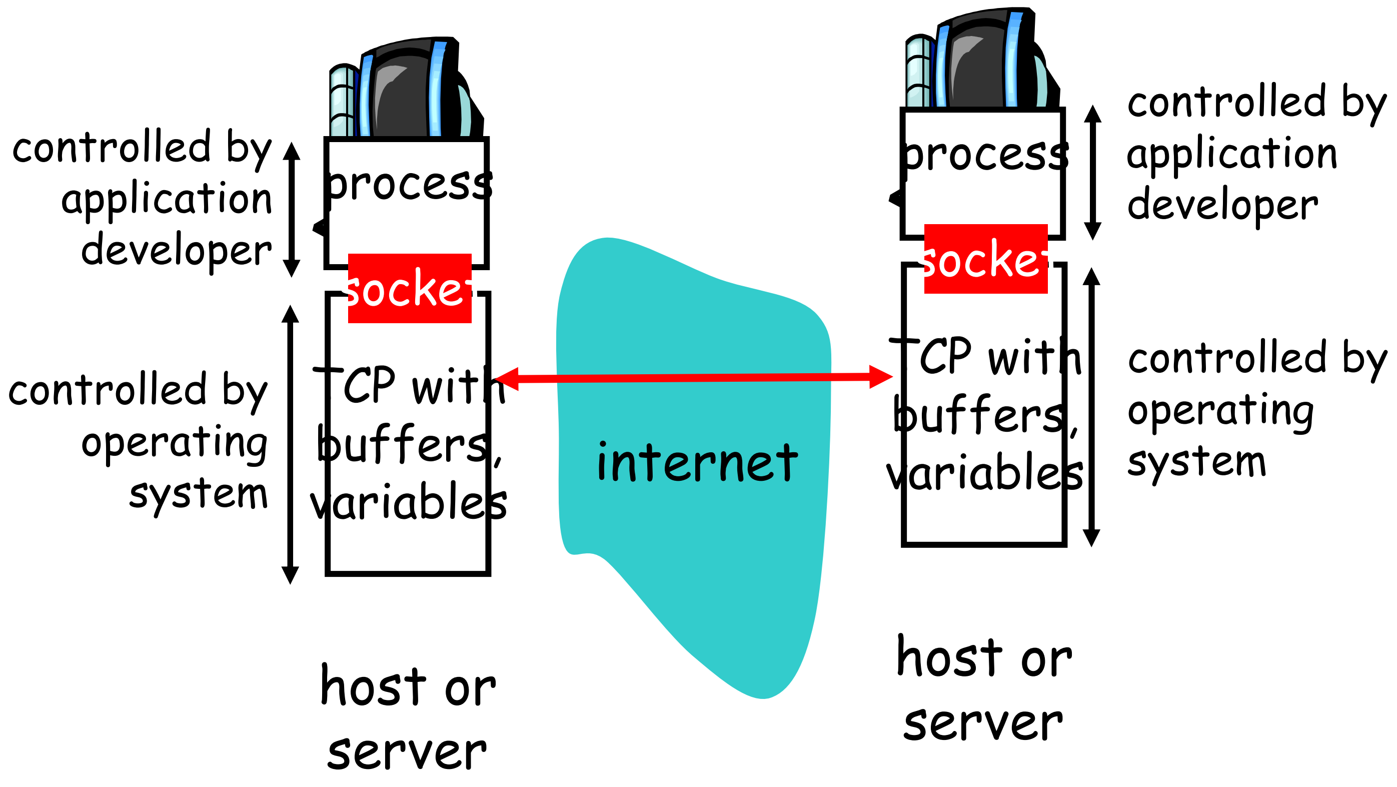
Σύγκριση TCP-UDP

	TCP	UDP
ΑΠΑΙΤΗΣΗ ΣΥΝΔΕΣΗ	ΝΑΙ	ΌΧΙ
ΑΞΙΟΠΙΣΤΙΑ	ΝΑΙ	ΌΧΙ
ΟΡΙΑ ΜΗΝΥΜΑΤΩΝ	ΌΧΙ	ΝΑΙ
ΔΙΑΔΟΧΙΚΟΤΗΤΑ ΜΗΝΥΜΑΤΩΝ	ΝΑΙ	ΌΧΙ
ΕΙΔΟΣ ΥΠΟΔΟΧΗΣ	SOCK_STREAM	SOCK_DGRAM

Socket-programming using TCP

Socket: a door between application process and end-end-transport protocol (UDP or TCP)

TCP service: reliable transfer of **bytes** from one process to another


controlled by application developer
process
socket
controlled by operating system
TCP with buffers, variables
host or server
internet
process
socket
TCP with buffers, variables
controlled by application developer
controlled by operating system
host or server

Socket programming with TCP

Client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

Client contacts server by:

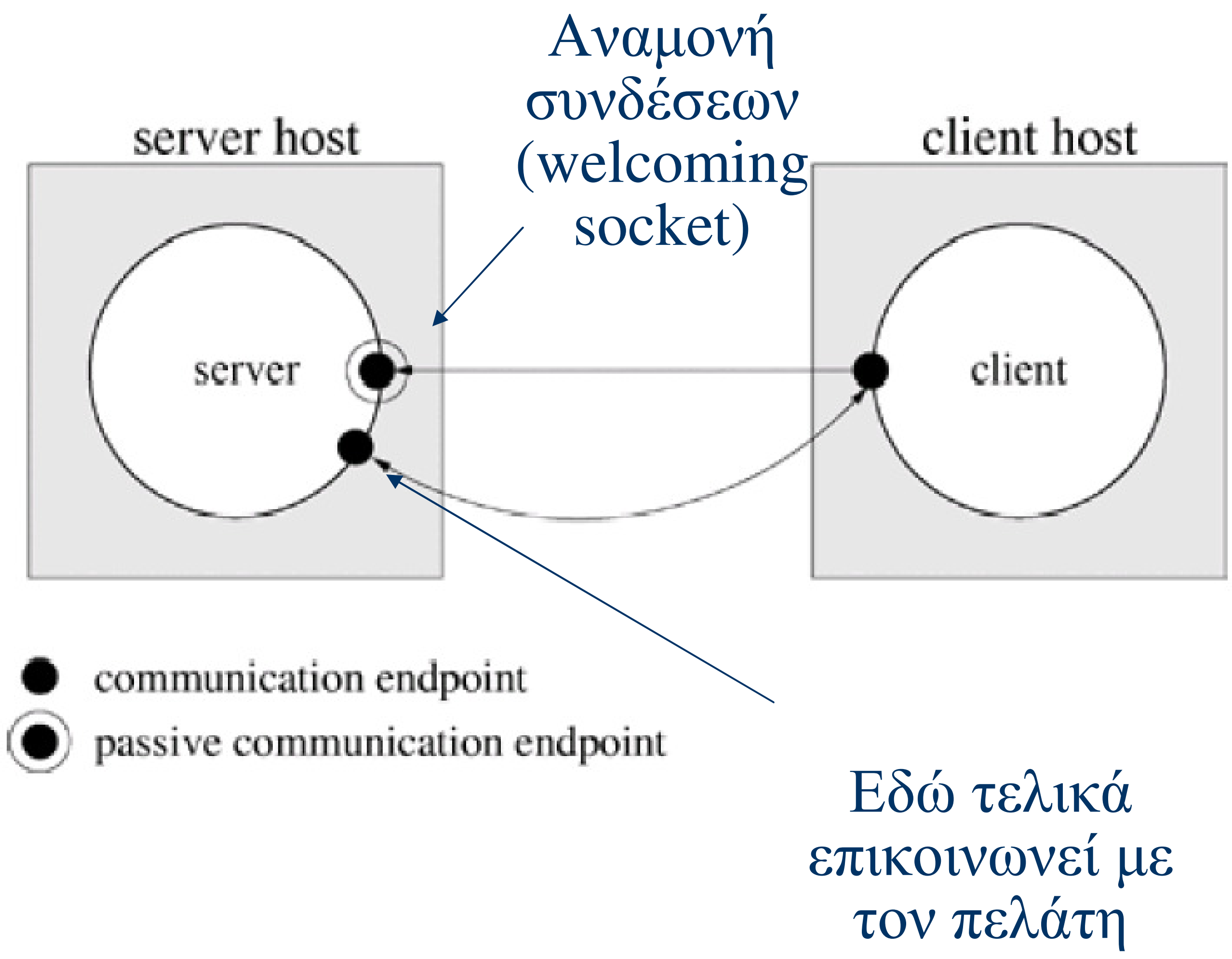
- creating client-local TCP socket
- specifying IP address, port number of server process
- When client creates socket: client TCP establishes connection to server TCP

- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients

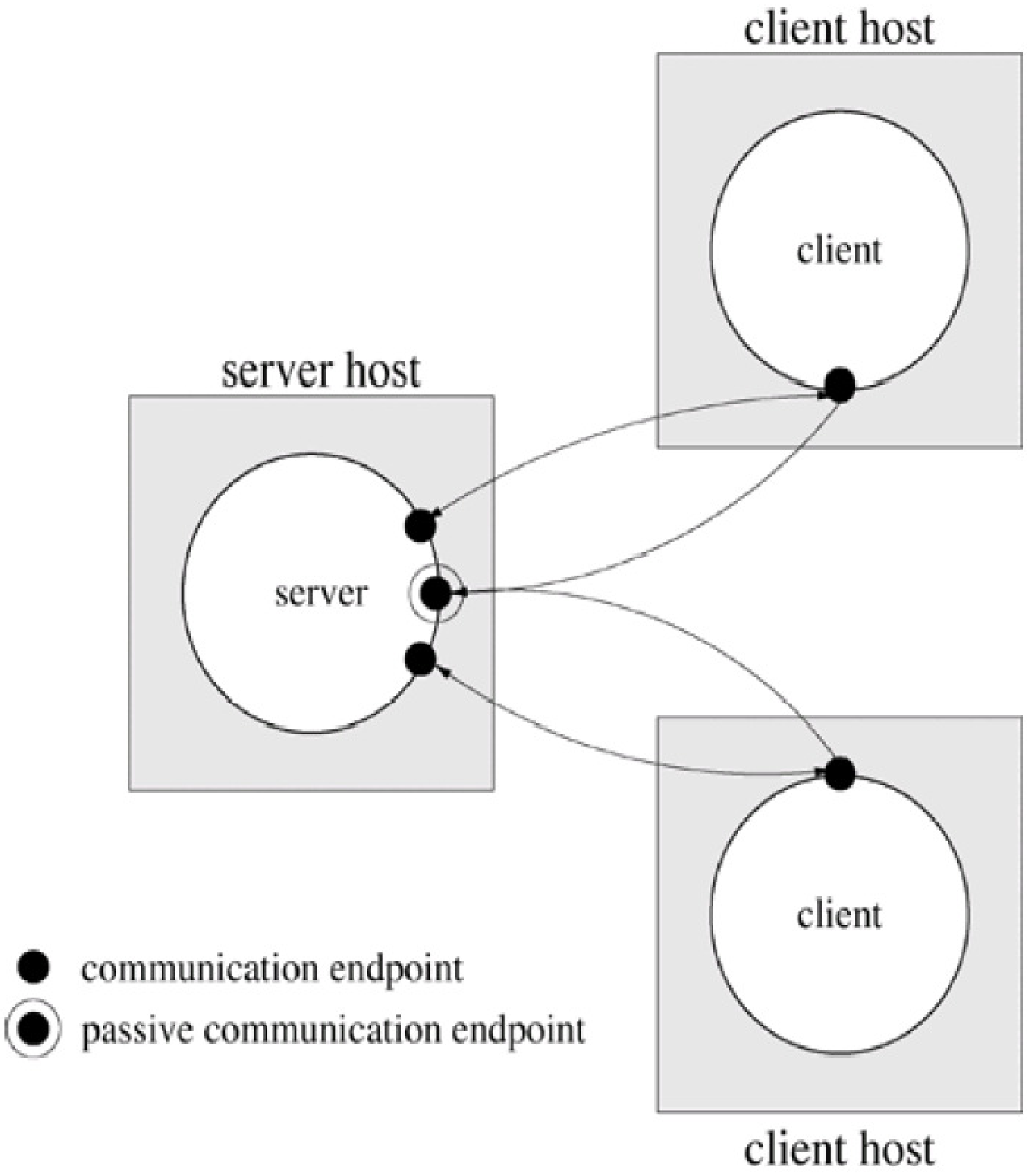
application viewpoint

TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

Πρωτόκολλο TCP


Αναμονή συνδέσεων (welcoming socket)
server host
client host
server
client
● communication endpoint
◉ passive communication endpoint
Εδώ τελικά επικοινωνεί με τον πελάτη

Πολλοί Πελάτες



Σειριακός Εξυπηρετητής

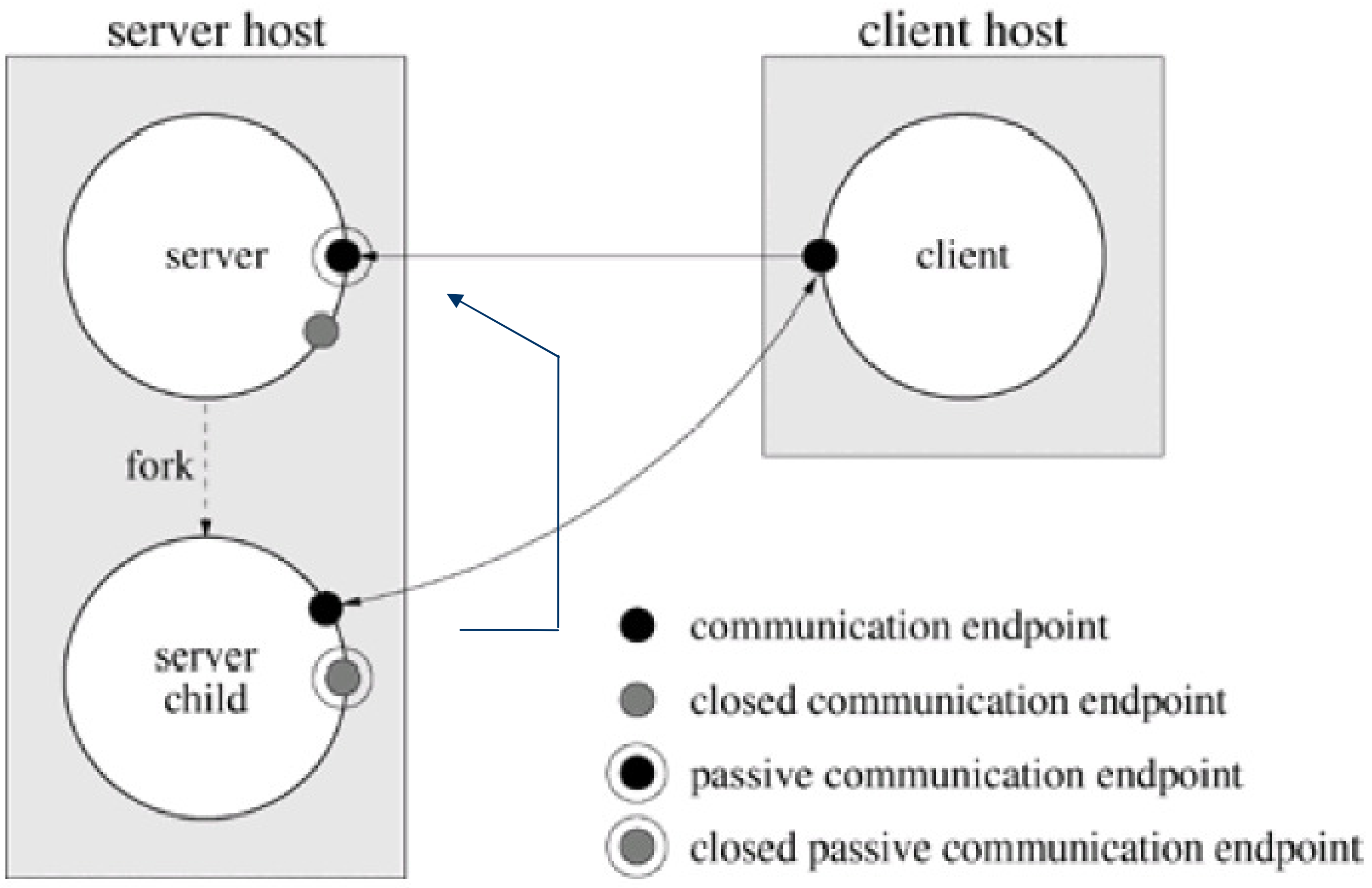
```
for (;;) {
  wait for a client request on the listening file descriptor
  create a private two-way communication channel to the clie
  while (no error on the private communication channel)
          read from the client
          process the request
          respond to the client
   close the file descriptor for the private communication channel
}
```

ΠΑΝΤΑ

ΜΕΙΟΝΕΚΤΗΜΑΤΑ

1) Ένα πελάτη τη φορά, μέχρι να τελειώσει ο πελάτης…
2) Αιτήσεις άλλων πελατών περιμένουν και μπορεί να αποτύχουν

Πιο Σύνηθες


server host
client host
server
client
fork
server child
communication endpoint
closed communication endpoint
passive communication endpoint
closed passive communication endpoint

Ο εξυπηρετητής γεννάει ένα παιδί για κάθε πελάτη

ΜΕΙΟΝΕΚΤΗΜΑ

Πάρα πολλά παιδιά αν πολλοί πελάτες

Λειτουργία

Λειτουργία Πατέρα

```
for (;;) {
  wait for a client request on the listening file descriptor
  create a private two-way communication channel to the client
  fork a child to handle the client
  close the file descriptor for the private communication channel
  clean up zombie children
}
```

ΠΑΝΤΑ

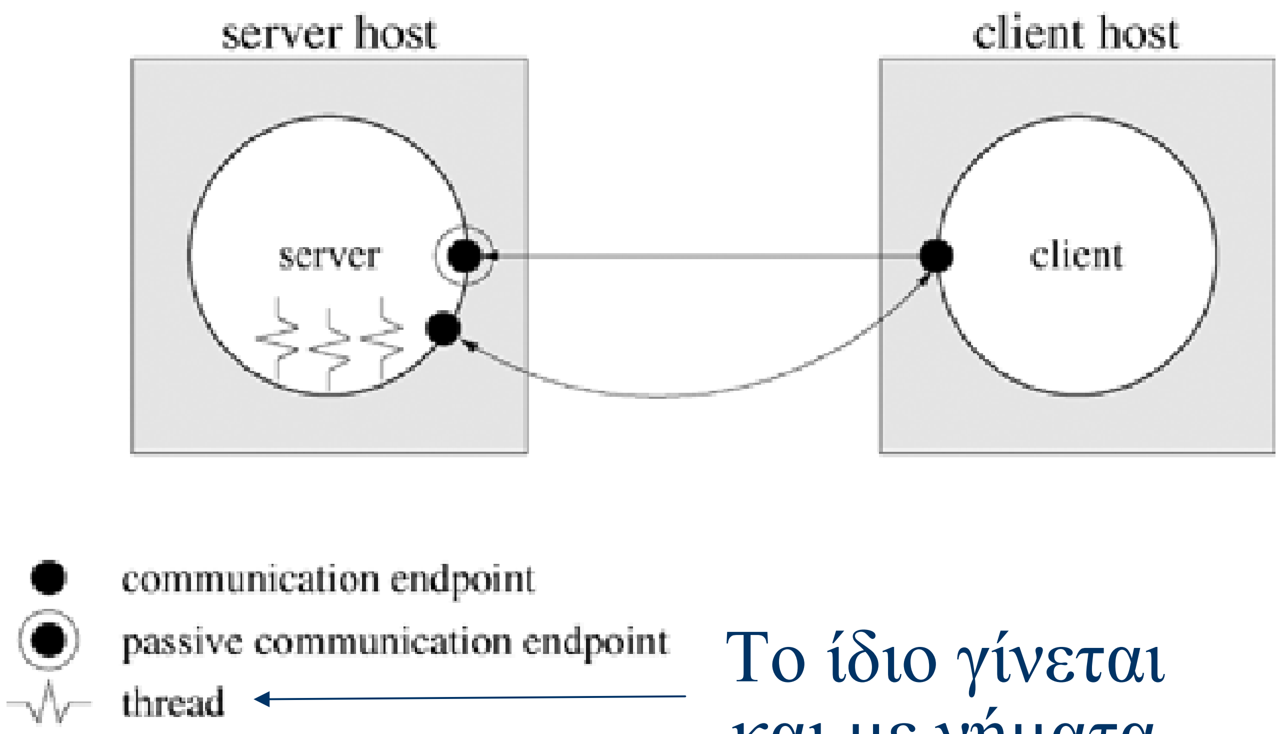
Λειτουργία Παιδιού

```
close the listening file descriptor
handle the client
close the communication for the private channel
exit
```

Ερωτήσεις

- Γιατί ο εξυπηρετητής να κλείνει τους περιγραφείς;
  - Για να μη ξεμείνει από περιγραφείς
  - Για να μπορεί ο πελάτης να διαγνώσει EOF με τη read (όπως στους σωλήνες, αν δεν υπάρχουν άλλα δεδομένα και η άλλη πλευρά κλείσει το άκρο)

Εναλλακτικό Μοντέλο



Το ίδιο γίνεται και με νήματα

Αντί για μια καινούργια διεργασία ανά πελάτη, δημιουργεί ένα καινούργιο νήμα.

ΜΕΙΟΝΕΚΤΗΜΑ

Συγχρονισμός νημάτων για διάβασμα δεδομένων πελατών (πχ από σωλήνα)

Χρήσιμες Γνώσεις

•Ένας ακέραιος με δεκαεξαδική αναπαράσταση 0xD04C σε little-endian αρχιτεκτονική θα ερμηνευτεί ως 0x4CD0 σε big-endian host.
(π.χ., SUN (big-endian) vs Intel (little-endian))

•Η αποστολή δυαδικών αριθμών πρέπει να γίνεται με κάποια προσυμφωνημένη δικτυακή διάταξη

•Συναρτήσεις βιβλιοθήκης htons, htonl, ntohs, και ntoh

```
unsigned short htons(unsigned short hostshort)
unsigned long htonl(unsigned long hostlong)
unsigned short ntohs(unsigned short netshort)
unsigned long ntohl(unsigned long netlong)
```

•Μετατροπή ακολουθίας bytes από διάταξη «μηχανής» σε διάταξη «δικτύου» και αντίστροφα για short και long ακεραίους

•Απαιτήσεις

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

Χρήσιμες Γνώσεις (2)

•Συναρτήσεις βιβλιοθήκης gethostbyname και gethostbyaddr

```
struct hostent *gethostbyname(char *name)
struct hostent *gethostbyaddr(char *addr,
                              int len, int type)
```

•Η gethostbyname βρίσκει τη διεύθυνση IP που αντιστοιχεί στο όνομα name.

•Επιστρέφει έναν δείκτη σε δομή struct hostent όπου στο πεδίο h_addr της δομής βρίσκεται η Internet διεύθυνση και στο πεδίο h_length το μέγεθος της

•Η gethostbyaddr μετατρέπει μια διεύθυνση IP addr, με μέγεθος len, και τύπου type (πάντα **AF_INET**), σε όνομα.

•Επιστρέφει έναν δείκτη σε δομή struct hostent όπου στο πεδίο h_name βρίσκεται το όνομα του υπολογιστή

•Επιστρέφουν NULL σε περίπτωση σφάλματος (ορίζει το σφάλμα το h_errno)

•Απαίτηση: #include <netdb.h>

Χρήσιμες Γνώσεις (3)

•Όλες οι κλήσεις συστήματος που ακολουθούν και χρησιμοποιούνται για διαχείριση υποδοχών απαιτούν

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

και επιστρέφουν -1 σε περίπτωση αποτυχίας

•Κλήση συστήματος socket

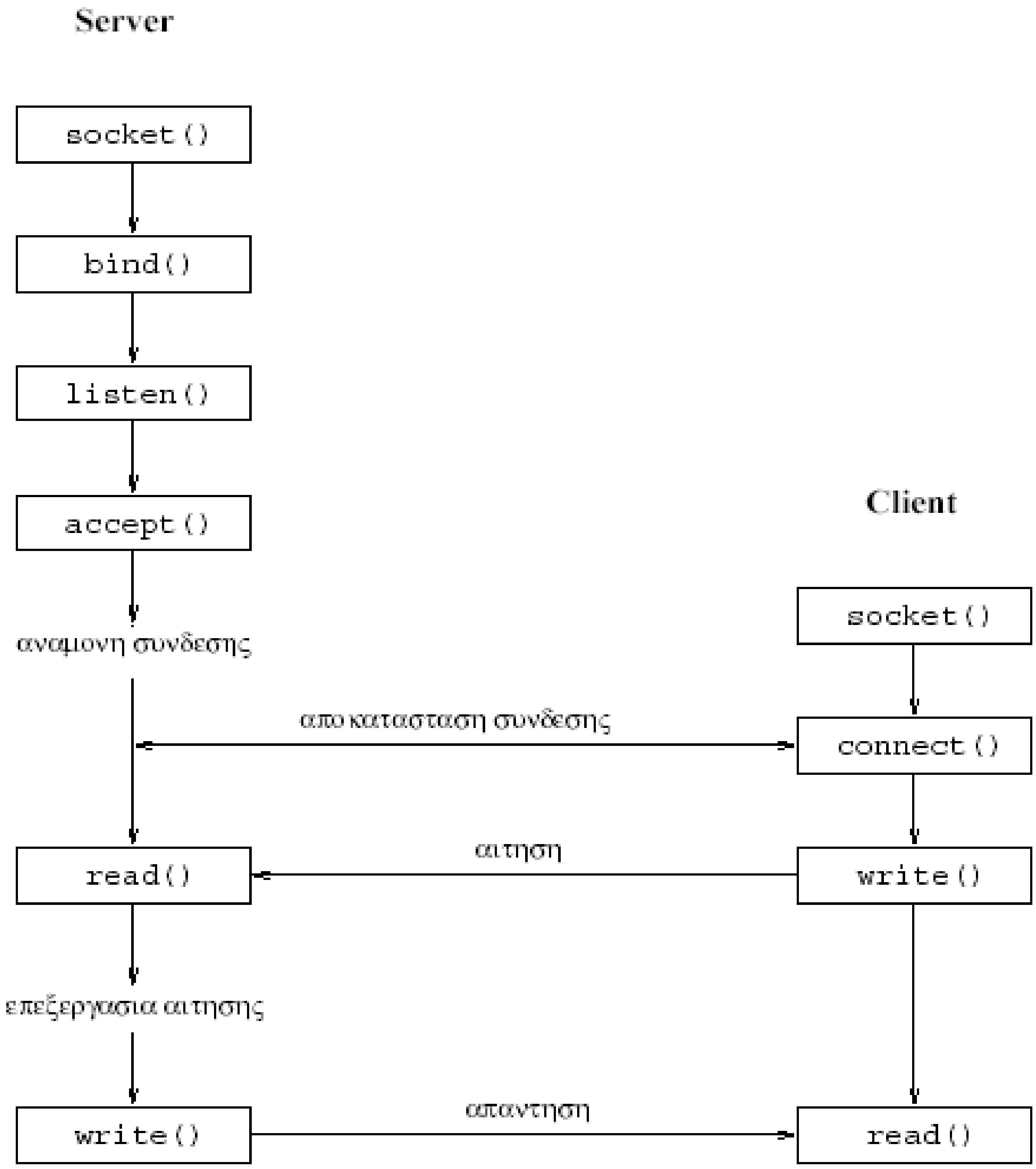
```
int socket(int domain, int type, int protocol)
```

Δημιουργεί μια υποδοχή και επιστρέφει ένα περιγραφει αρχείου που αντιστοιχεί σ'αυτην

Το domain πρέπει να είναι AF_INET (παλιά PF_INET)
Το type πρέπει να είναι SOCK_STREAM ή
SOCK_DGRAM
Σαν protocol, πάντα δίνουμε το default (0)

```
int sock;
if ((sock = socket(AF_INET, SOCK_STREAM,0)) == -1)
    perror("Failed to create socket");
```

TCP Επικοινωνία



Δομές για ορισμό διευθύνσεων υποδοχών

```
#include <netinet/in.h>
struct sockaddr_in {
    sa_family_t sin_family;       /* AF_INET */
    in_port  sin_port;            /* port number  */
    struct in_addr sin_addr;      /* IPv4 address */
};

struct in_addr {
    in_addr_t s_addr;        /* IPv4 unsigned 32 bit int */
};
```

Συνάρτηση bind

- `int bind(int fd, struct sockaddr *address, unsigned int addresslen)`
- Συνδέει την υποδοχή που αντιστοιχεί στον περιγραφέα αρχείου `fd` με ένα όνομα/διεύθυνση `*address`
- Πεδίο Internet
  * Ορίζεται ένα `struct sockaddr_in name` και δίνονται τα AF_INET στο πεδίο `name.sin_family`, `htonl(INADDR_ANY)` στο πεδίο `name.sin_addr.s_addr` και `htons(port)` στο πεδίο `name.sin_port`, όπου `port` είναι ο αριθμός θύρας που χρησιμοποιείται και αφού η διεύθυνση του `name` προσαρμοσθεί σε `(struct sockaddr *)` δίνεται στο `address`
  * Απαίτηση: `#include <netinet/in.h>`

```
struct sockaddr_in server;


server.sin_family = AF_INET;
server.sin_addr.s_addr = htonl(INADDR_ANY);
server.sin_port = htons((short)8652);
if (bind(sock, (struct sockaddr *)&server, sizeof(server)) == -1)
   perror("Failed to bind the socket to port");
```

Συναρτήσεις connect, listen, accept

≈ 5

- Κλήση συστήματος listen
  - `int listen(int fd, int queuelength)`
  - Ορίζει μία ουρά μήκους queuelength σε έναν server στην οποία μπορούν να συσσωρεύονται αιτήσεις από clients για σύνδεση στην υποδοχή που αντιστοιχεί στον περιγραφέα αρχείου fd
- Κλήση συστήματος accept
  - `int accept(int fd, struct sockaddr *address,`
    `unsigned int *addresslen)`
  - Αποδέχεται μία αίτηση σύνδεσης που έχει υποβληθεί σε έναν server στην υποδοχή με περιγραφέα αρχείου fd
  - Πληροφορίες για τη διεύθυνση του client που συνδέθηκε επιστρέφονται μέσω της δομής *address το μέγεθος της οποίας επιστρέφεται στο *addresslen
  - Επιστρέφει ένα νέο περιγραφέα αρχείου ο οποίος πρέπει να χρησιμοποιηθεί από τον server για επικοινωνία με τον client

Συνδέσου εδώ

- Κλήση συστήματος connect
  - `int connect(int fd, struct sockaddr *address,`
    `unsigned int addresslen)`
  - Υποβολή αίτησης σύνδεσης από έναν client μέσω υποδοχής που αντιστοιχεί στον περιγραφέα αρχείου fd με τον server του οποίου η διεύθυνση έχει κατασκευασθεί στη δομή *address το μέγεθος της οποίας έχει τεθεί στο addresslen

Συναρτήσεις bzero, bcopy

- Συναρτήσεις βιβλιοθήκης `bzero` και `bcopy`
  - `void bzero(char *buf, int count)`
    - Θέτει 0 σε count bytes αρχίζοντας από τη διεύθυνση buf
  - `void bcopy(char *buf1, char *buf2, int count)`
    - Αντιγράφει count bytes αρχίζοντας από τη διεύθυνση buf1 στη διεύθυνση buf2 ← Αντίθετα από memcpy
  - Απαίτηση: `#include <string.h>`
- Για μεταγλώττιση στο Solaris προγραμμάτων C με κλήσεις συστήματος για υποδοχές, πρέπει να προστίθεται στην εντολή μεταγλώττισης και το "`-lsocket -lnsl`"

Πρακτικές λύσεις

•Αν ο εξυπηρετητής πάει να γράψει σε socket που έχει κλείσει ο πελάτης, τότε λαμβανει σήμα SIGPIPE

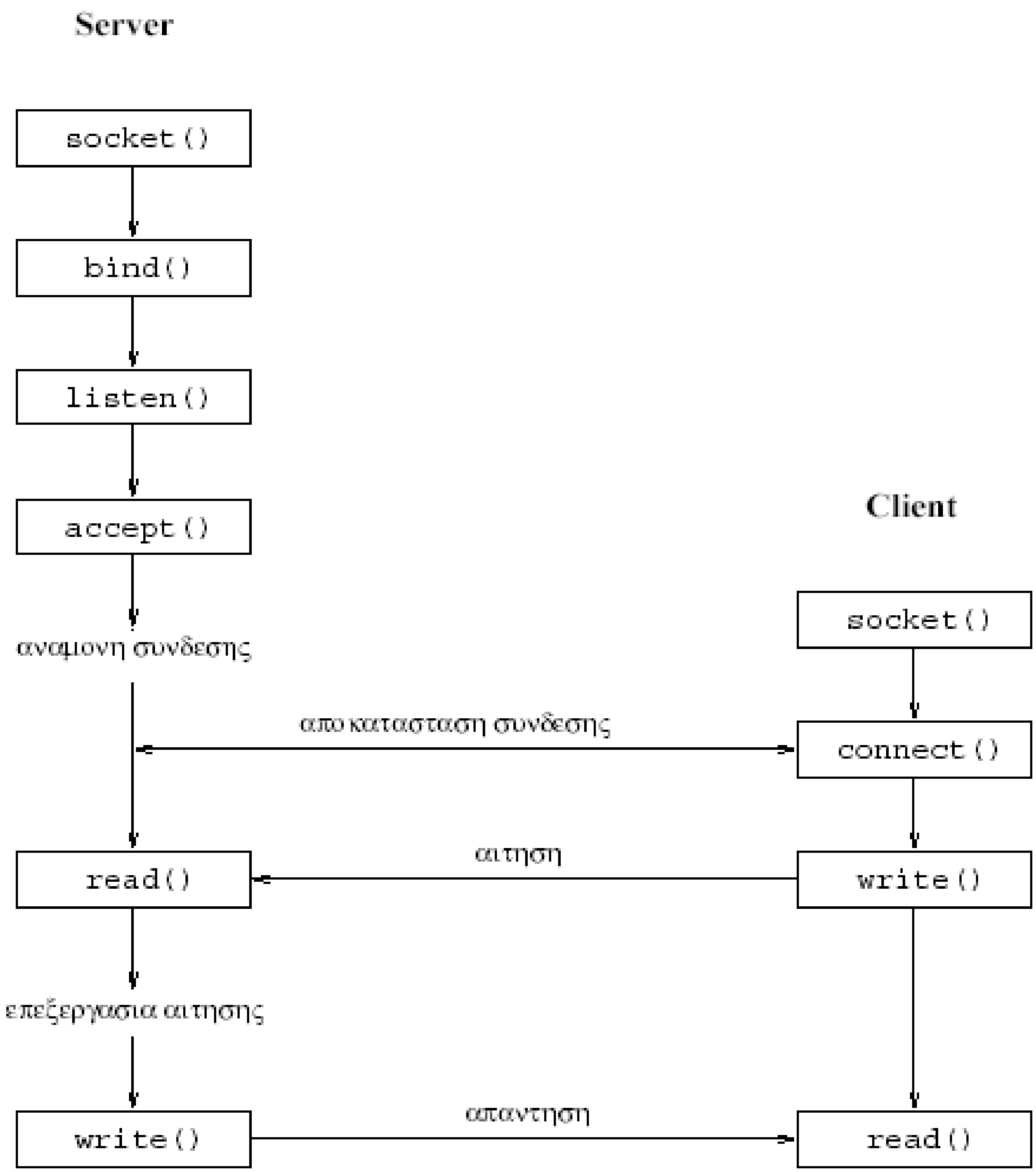
•Να βάζετε στην αρχή χειριστή του σήματος SIGPIPE (διοτι προεπιλεγμενη ενεργεια ειναι τερματισμος)

•Οταν ο server τερματιζει, ο αριθμος θυρας οπου ακουει παραμενει δεσμευμενο για ενα διαστημα (state TIME_WAIT)

•Σε reboot, server βλεπει σφαλμα στη Bind: Address Already in Use”

•Για γρήγορη επαναχρησιμοποίηση των ports ενός socket, κάντε χρήση της συνάρτησης setsockopt (με ενεργοποίηση της επιλογής SO_REUSEADDR – δείτε το βιβλίο για λεπτομέρειες)

TCP Επικοινωνία



Next…

- TCP server receives a string and replies with string capitalized

```
/*inet_str_server.c: Internet stream sockets server */
#include <stdio.h>
#include <sys/wait.h>          /* sockets */
#include <sys/types.h>         /* sockets */
#include <sys/socket.h>        /* sockets */
#include <netinet/in.h>        /* internet sockets */
#include <netdb.h>                /* gethostbyaddr */
#include <unistd.h>               /* fork */
#include <stdlib.h>               /* exit */
#include <ctype.h>                /* toupper */
#include <signal.h>         /* signal */
void child_server(int newsock);
void perror_exit(char *message);
void sigchld_handler (int sig);

void main(int argc, char *argv[]) {
   int            port, sock, newsock;
   struct sockaddr_in server, client;
   socklen_t clientlen;
   struct sockaddr *serverptr=(struct sockaddr *)&server;
   struct sockaddr *clientptr=(struct sockaddr *)&client;
   struct hostent *rem;
   if (argc != 2) {
      printf("Please give port number\n");exit(1);}
   port = atoi(argv[1]);
   /* Reap dead children asynchronously */
   signal(SIGCHLD, sigchld_handler);
```

```
/* Create socket */
  if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
     perror_exit("socket");
  server.sin_family = AF_INET;  /* Internet domain */
  server.sin_addr.s_addr = htonl(INADDR_ANY);
  server.sin_port = htons(port);     /* The given port */
  /* Bind socket to address */
  if (bind(sock, serverptr, sizeof(server)) < 0)
     perror_exit("bind");
  /* Listen for connections */
  if (listen(sock, 5) < 0) perror_exit("listen");
  printf("Listening for connections to port %d\n", port);
  while (1) {
     /* accept connection */
          if ((newsock = accept(sock, clientptr, &clientlen))
           < 0) perror_exit("accept");
          /* Find client's address */
          if ((rem = gethostbyaddr((char *) &client.sin_addr.s_a
         sizeof(client.sin_addr.s_addr), client.sin_family))
          == NULL) {
             herror("gethostbyaddr"); exit(1);}
          printf("Accepted connection from %s\n", rem->h_na
          switch (fork()) {    /* Create child for serving client */
          case -1:     /* Error */
             perror("fork"); break;
          case 0:       /* Child process */
             close(sock); child_server(newsock);
             exit(0);
          }
close(newsock); /* parent closes socket to client */
    }
}

void child_server(int newsock) {
    char buf[1];
    while(read(newsock, buf, 1) > 0) {  /* Receive 1 char */
              putchar(buf[0]);          /* Print received char */
              /* Capitalize character */
              buf[0] = toupper(buf[0]);
              /* Reply */
              if (write(newsock, buf, 1) < 0)
                  perror_exit("write");
    }
    printf("Closing connection.\n");
    close(newsock);              /* Close socket */
}

/* Wait for all dead child processes */
void sigchld_handler(int sig) {
              while (waitpid(-1, NULL, WNOHANG) > 0);
}

void perror_exit(char *message) {
    perror(message);
    exit(EXIT_FAILURE);
}
```

And now the client…

```
/* inet_str_client.c: Internet stream sockets client */
#include <stdio.h>
#include <sys/types.h>          /* sockets */
#include <sys/socket.h>         /* sockets */
#include <netinet/in.h>         /* internet sockets */
#include <unistd.h>         /* read, write, close */
#include <netdb.h>                  /* gethostbyaddr */
#include <stdlib.h>                 /* exit */
#include <string.h>                 /* strlen */

void perror_exit(char *message);

void main(int argc, char *argv[]) {
    int           port, sock, i;
    char            buf[256];
    struct sockaddr_in server;
    struct sockaddr *serverptr = (struct sockaddr*)&server;
    struct hostent *rem;
    if (argc != 3) {
            printf("Please give host name and port number\n");
            exit(1);}
            /* Create socket */
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
            perror_exit("socket");
            /* Find server address */
    if ((rem = gethostbyname(argv[1])) == NULL) {
              herror("gethostbyname"); exit(1);
    }
port = atoi(argv[2]);    /*Convert port number to integer*/
   server.sin_family = AF_INET;       /* Internet domain */
   memcpy(&server.sin_addr, rem->h_addr, rem->h_length);
   server.sin_port = htons(port);        /* Server port */
   /* Initiate connection */
   if (connect(sock, serverptr, sizeof(server)) < 0)
             perror_exit("connect");
   printf("Connecting to %s port %d\n", argv[1], port);
   do {
           printf("Give input string: ");
           fgets(buf, sizeof(buf), stdin);        /* Read from stdin*
           for(i=0; buf[i] != '\0'; i++) { /* For every char */
              /* Send i-th character */
           if (write(sock, buf + i, 1) < 0)
             perror_exit("write");
         /* receive i-th character transformed */
           if (read(sock, buf + i, 1) < 0)
              perror_exit("read");
           }
           printf("Received string: %s", buf);
   } while (strcmp(buf, "END\n") != 0); /* Finish on "end" */
   close(sock);              /* Close socket and exit */
}

void perror_exit(char *message)
{
   perror(message);
   exit(EXIT_FAILURE);
}
```

Server on linux02:

```
mema@bowser> ./server 9002
Listening for connections to port 9002
Accepted connection from linux03.di.uoa.gr
Hello world
EnD
Closing connection .
```

Client on linux03:

```
mema@bowser> ./client linux02.di.uoa.gr 9002
Connecting to linux02.di.uoa.gr port 9002
Give input string : Hello world
Received string : HELLO WORLD
Give input string : EnD
Received string : END
mema@bowser>
```

Χρήσιμες Γνώσεις

- INADDR_ANY (0.0.0.0)
  - Αντιστοιχεί στο άκου σε «όλες τις διευθύνσεις»
- port = 0
  - Αφήνει το λειτουργικό να επιλέξει port.
- int setsockopt()
  - Θέτει ειδικές ιδιότητες του socket
  - Θα δούμε χρήση παρακάτω
- int shutdown(int socket_fd, int how)
  - Κλείνει τη μια κατεύθυνση ενός αμφίδρομου socket
  - how = SHUT_RD, SHUT_WR, or SHUT_RDWR
  - Example: HTTP client sends request, then shutdown(fd, SHUT_WR) to tell server “I am done sending the whole request, but can still receive.” Server detects EOF by receive of 0 bytes and can assume it has complete request.
  - At end, we close the socket with close() call

Χρήσιμες Γνώσεις

- Που είναι συνδεδεμένο το socket του server μου;
  - int getsockname(int sock_fd, struct sockaddr *sa, int *sa_len)
  - διεύθυνση όπου έχει γίνει το socket bind.
  - On return, sa points to struct sockaddr where fields are filled in

- How do I get my server to find out the client’s hostname/address?
  - Use int getpeername(int sock_fd, struct sockaddr *sa, int *sa_len)
  - διεύθυνση στο άλλο άκρο της σύνδεσης.
  - On return, sa points to struct sockaddr with fields filled in

Εξέταση Ενεργών Sockets

- Εντολή netstat
- Πχ: netstat –n –l –p
  - Δείξε με νούμερα (-n)
  - τα LISTEN-ing sockets (-l)
  - και τις διεργασίες στις οποίες ανήκουν (-p)
- Ακόμα...
  - -t / -u TCP/UDP
  - -a socket σε όλα τα states
- TCP states
  - LISTEN, ESTABLISHED, TIME_WAIT…

Parsing and Printing Addresses

Old:

inet_ntoa	**Convert struct in_addr to printable form ‘a.b.c.d’**
inet_aton	**Convert IP address string in ‘.’ notation to 32bit network address**

New (these work with IPv6 as well as IPv4):

inet_ntop	**Convert address from network format to printable presentation format**
inet_pton	**Convert presentation format address to network format**

Review

- A traditional way to write network servers:
  - the main server process (or thread) blocks on accept(), waiting for a connection.
  - once a connection comes in, the server forks new process (thread), the child process (thread) handles the connection
  - the main server process is able to accept new incoming requests.

Alternative server structure with select ()

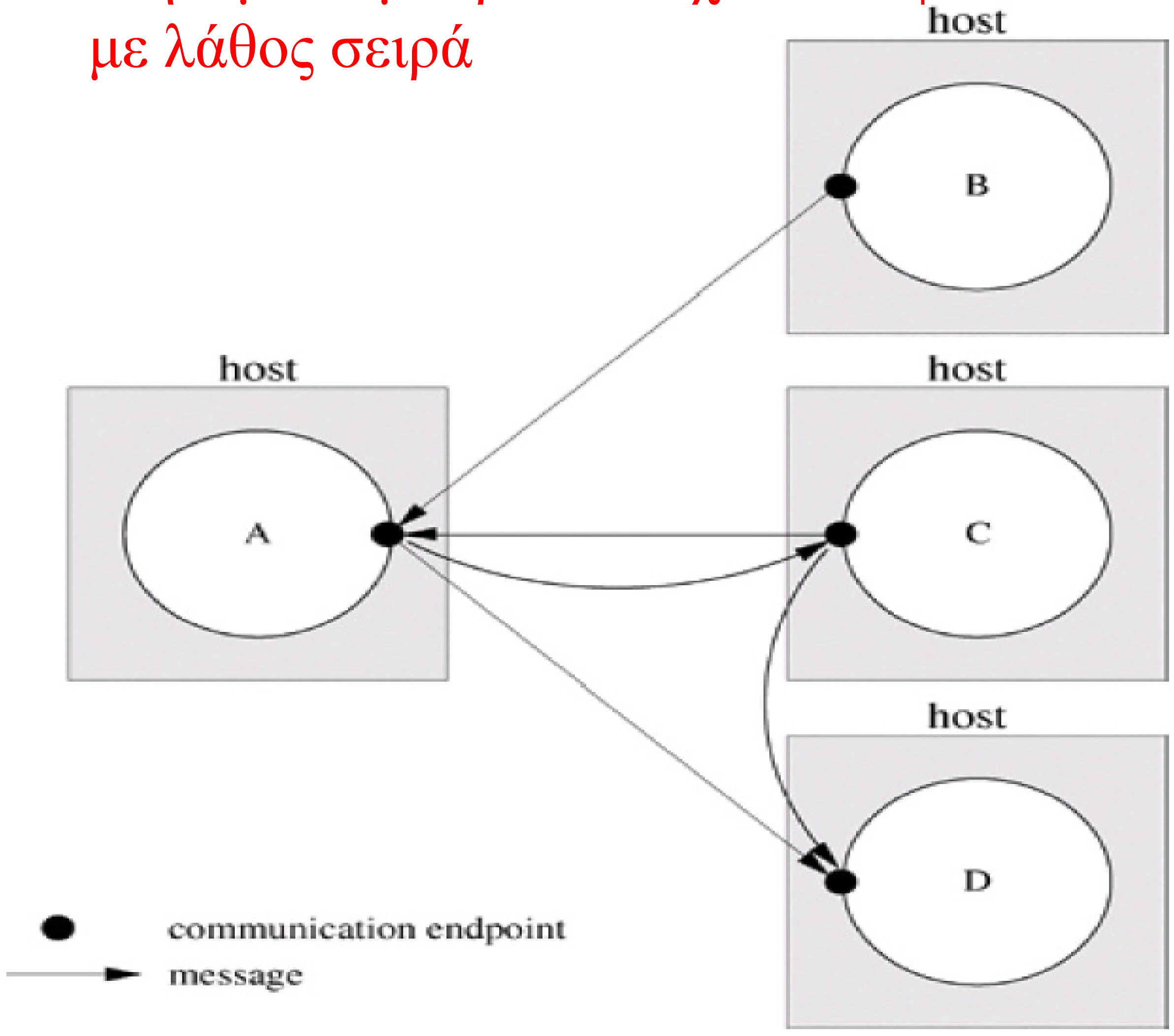
- A single process multiplexes all sockets via *select()* call
  - Adv: no need for separate child process/thread per request
- Select() blocks until an "event" occurs on a file descriptor (socket)
  - Data comes in off the net for a socket
  - File descriptor ready for writing
  - Describe events (file descriptors) of interest by filling a *fd_set* structure using macros

Alternative server structure with select () *

- Fill up a fd_set structure with the fds you want to know when data comes in on
- Fill up a fd_set structure with the fds you want to know when you can write on
- Call select(), block until event occurs
- When select returns, check if any fds was the reason process woke up and service relevant fd
- Repeat forever
- * See code sample at: http://www.lowtek.com/sockets/select.html

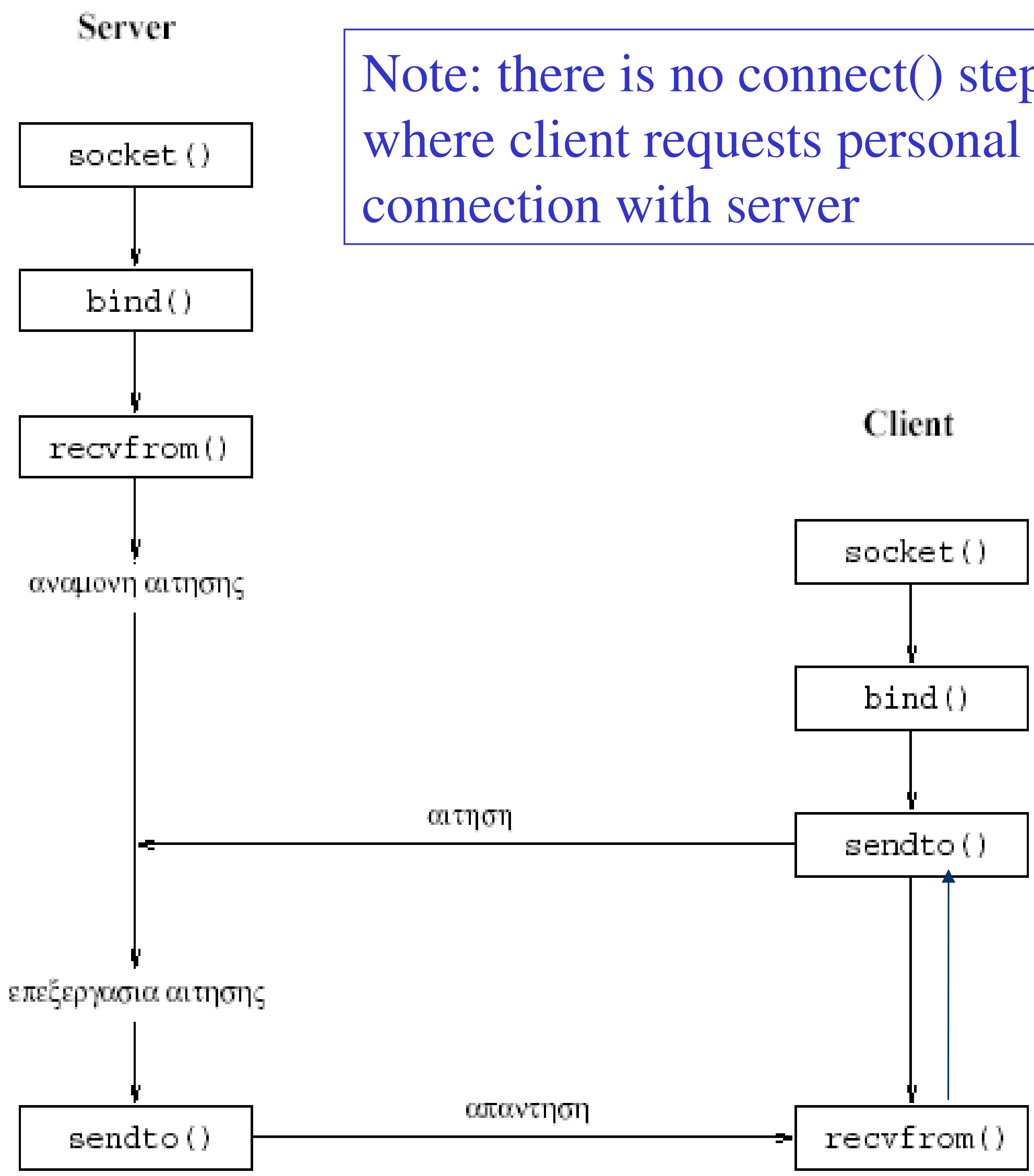
Πρωτόκολλο UDP

- User Datagram Protocol
- Επικοινωνία χωρίς σταθερή σύνδεση
- Ανταλλαγή απλών μηνυμάτων μεταξύ εξυπηρετητή-πελατών
- Μηνύματα μπορούν να χαθούν, φτάσουν με λάθος σειρά



UDP Επικοινωνία

Note: there is no connect() step where client requests personal connection with server



Συναρτήσεις sendto, recvfrom

```
– int recvfrom(int fd, char *buf, int count,
        int flags, struct sockaddr *address,
        unsigned int *addresslen)
– int sendto(int fd, char *buf, int count,
        int flags, struct sockaddr *address,
        unsigned int addresslen)
```

– Χρησιμοποιούνται αντί των read και write για την παραλαβή και την αποστολή μηνυμάτων μέσω UDP

– Τα buf και count έχουν την ίδια σημασία όπως στις read και write

– Στο flags συνήθως δίνεται η τιμή 0, εκτός αν πρόκειται να γίνει χειρισμός κάποιων ειδικών περιπτώσεων

– Στη δομή *address επιστρέφεται η διεύθυνση της τηλεγραφικής υποδοχής που χρησιμοποιεί ο αποστολέας (για τη recvfrom) ή τίθεται η διεύθυνση της τηλεγραφικής υποδοχής που χρησιμοποιεί ο παραλήπτης (για τη sendto)

– Στο *addresslen επιστρέφεται το μέγεθος της διεύθυνσης της υποδοχής του αποστολέα (για τη recvfrom), ενώ στο addresslen τίθεται η διεύθυνση της υποδοχής του παραλήπτη (για τη sendto)

A simple echoing UDP server

Client on linux03:

```
mema@linux03> ./inet_dgr_client linux02 49024
Unix is cool
Unix is cool
Tax evasion SUCKS
Tax evasion SUCKS
Can I go back?
Can I go back?
mema@linux03>
```

Server on linux02:

```
mema@linux02> ./inet_dgr_server
Socket port : 49024
Received from linux03 : Unix is cool
Received from linux03 : Tax evasion SUCKS
Received from linux03 : Can I go back?
```

```
/* inet_dgr_server.c: Internet datagram sockets server */
#include <sys/types.h>                          /* sockets */
#include <sys/socket.h>                         /* sockets */
#include <netinet/in.h>                /* Internet sockets */
#include <netdb.h>                       /* gethostbyaddr */
#include <arpa/inet.h>                       /* inet_ntoa */
#include <stdio.h>
#include <stdlib.h>
void perror_exit(char *message);
char *name_from_address(struct in_addr addr) {
   struct hostent *rem; int asize = sizeof(addr.s_addr);
   if ((rem = gethostbyaddr(&addr.s_addr, asize, AF_INET)))
      return rem->h_name;  /* reverse lookup success */
   return inet_ntoa(addr);  /* fallback to a.b.c.d form */
}
void main() {
   int n, sock; unsigned int serverlen, clientlen;
   char buf[256], *clientname;
   struct sockaddr_in server, client;
   struct sockaddr *serverptr = (struct sockaddr*) &server;
   struct sockaddr *clientptr = (struct sockaddr*) &client;
   /* Create datagram socket */
   if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
      perror_exit("socket");
   /* Bind socket to address */
   server.sin_family = AF_INET;       /* Internet domain */
   server.sin_addr.s_addr = htonl(INADDR_ANY);
   server.sin_port = htons(0);      /* Autoselect a port */
   serverlen = sizeof(server);
if (bind(sock, serverptr, serverlen) < 0)
      perror_exit("bind");
   /* Discover selected port */
   if (getsockname(sock, serverptr, &serverlen) < 0)
      perror_exit("getsockname");
   printf("Socket port: %d\n", ntohs(server.sin_port));
   while(1) { clientlen = sizeof(client);
      /* Receive message */
      if ((n = recvfrom(sock, buf, sizeof(buf), 0, clientptr,
                              &clientlen)) < 0)
         perror("recvfrom");
      buf[sizeof(buf)-1]='\0';   /* force str termination */
      /* Try to discover client's name */
      clientname = name_from_address(client.sin_addr);
      printf("Received from %s: %s\n", clientname, buf);
      /* Send message */
      if (sendto(sock, buf, n, 0, clientptr, clientlen)<0)
         perror_exit("sendto");
   }}

void perror_exit(char *message)
{
   perror(message);
   exit(EXIT_FAILURE);
}
```

```
/* inet_dgr_client.c: Internet datagram sockets client    */
#include <sys/types.h>                    /* sockets */
#include <sys/socket.h>                   /* sockets */
#include <netinet/in.h>             /* Internet sockets */
#include <netdb.h>                  /* gethostbyname */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void main(int argc, char *argv[]) {
  int sock; char buf[256]; struct hostent *rem;
  struct sockaddr_in server, client;
  unsigned int serverlen = sizeof(server);
  struct sockaddr *serverptr = (struct sockaddr *) &server;
  struct sockaddr *clientptr = (struct sockaddr *) &client;
  if (argc < 3) {
    printf("Please give host name and port\n"); exit(1);}
  /* Create socket */
  if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
    perror("socket"); exit(1); }
  /* Find server's IP address */
  if ((rem = gethostbyname(argv[1])) == NULL) {
    herror("gethostbyname"); exit(1); }
  /* Setup server's IP address and port */
  server.sin_family = AF_INET;      /* Internet domain */
  memcpy(&server.sin_addr, rem->h_addr, rem->h_length);
  server.sin_port = htons(atoi(argv[2]));
/* Setup my address */
   client.sin_family = AF_INET;        /* Internet domain */
   client.sin_addr.s_addr=htonl(INADDR_ANY); /*Any address*/
   client.sin_port = htons(0);         /* Autoselect port */
   /* Bind my socket to my address*/
   if (bind(sock, clientptr, sizeof(client)) < 0) {
      perror("bind"); exit(1); }
   /* Read continuously messages from stdin */
   while (fgets(buf, sizeof buf, stdin)) {
      buf[strlen(buf)-1] = '\0';          /* Remove '\n' */
      if (sendto(sock, buf, strlen(buf)+1, 0, serverptr,
                  serverlen) < 0) {
         perror("sendto"); exit(1); }     /* Send message */
      bzero(buf, sizeof buf);             /* Erase buffer */
      if (recvfrom(sock, buf, sizeof(buf), 0, NULL, NULL) < 0) {
         perror("recvfrom"); exit(1); }  /* Receive message */
      printf("%s\n", buf); }
}
```

Looks good but….

- Everything looks good and runs ok, but there is a BUG!
- UDP is *unreliable*

```
/* rlsd.c - a remote ls server - with paranoia */
#include  <stdio.h>
#include  <stdlib.h>
#include  <unistd.h>
#include  <sys/types.h>
#include  <sys/socket.h>
#include  <netinet/in.h>
#include  <netdb.h>
#include  <time.h>
#include  <string.h>
#include  <ctype.h>
#define PORTNUM  15000       /* rlsd listens on this port */

void perror_exit(char *msg);
void sanitize(char *str);

int main(int argc, char *argv[]) {
   struct sockaddr_in myaddr;  /* build our address here */
   int      c, lsock, csock;     /* listening and client sockets */
   FILE *sock_fp;            /* stream for socket IO */
   FILE *pipe_fp;                     /* use popen to run ls */
   char    dirname[BUFSIZ];             /* from client */
   char    command[BUFSIZ];             /* for popen() */

   /** create a TCP a socket **/
   if ((lsock = socket( AF_INET, SOCK_STREAM, 0)) < 0)
          perror_exit( "socket" );
```

```
/** bind address to socket. **/
   myaddr.sin_addr.s_addr = htonl(INADDR_ANY);
   myaddr.sin_port = htons(PORTNUM); /*port to bind socket*/
   myaddr.sin_family = AF_INET;  /* internet addr family */
   if(bind(lsock,(struct sockaddr*)&myaddr,sizeof(myaddr)))
        perror_exit( "bind" );
   /** listen for connections with Qsize=5 **/
   if ( listen(lsock, 5) != 0 )
          perror_exit( "listen" );
   while ( 1 ){ /* main loop: accept - read - write */
         /* accept connection, ignore client address */
        if ( (csock = accept(lsock, NULL, NULL)) < 0 )
                perror_exit("accept");
        /* open socket as buffered stream */
        if ((sock_fp = fdopen(csock,"r+")) == NULL)
                perror_exit("fdopen");
            /* read dirname and build ls command line */
        if (fgets(dirname, BUFSIZ, sock_fp) == NULL)
                perror_exit("reading dirname");
        sanitize(dirname);
        snprintf(command, BUFSIZ, "ls %s", dirname);
        /* Invoke ls through popen */
        if ((pipe_fp = popen(command, "r")) == NULL )
                perror_exit("popen");
        /* transfer data from ls to socket */
        while( (c = getc(pipe_fp)) != EOF )
                putc(c, sock_fp);
        pclose(pipe_fp);
        fclose(sock_fp);
   }
```

(Fork+exec+pipe creation). Αντί για ανακατεύθυνση εντολής σε αρχείο, popen. Μπορώ να διαβάσω ή γράψω με popen. ΌΧΙ και τα 2 μαζί

```
return 0;
}

/* it would be very bad if someone passed us a dirname like
 * "; rm *"  and we naively created a command  "ls ; rm *".
 * So..we remove everything but slashes and alphanumerics.
 */
void sanitize(char *str)
{
        char *src, *dest;
        for ( src = dest = str ; *src ; src++ )
                if ( *src == '/' || isalnum(*src) )
                        *dest++ = *src;
        *dest = '\0';
}

/* Print error message and exit */
void perror_exit(char *message)
{
   perror(message);
   exit(EXIT_FAILURE);
}
```

```c
/* rls.c - a client for a remote directory listing service
 *        usage: rls hostname directory */
#include <stdio.h>
#include <stdlib.h>                                    /* exit */
#include <string.h>                                   /* strlen */
#include <unistd.h>                          /* STDOUT_FILENO */
#include <sys/types.h>                             /* sockets */
#include <sys/socket.h>                            /* sockets */
#include <netinet/in.h>                  /* internet sockets */
#include <netdb.h>                         /* gethostbyname */
#define  PORTNUM 15000
#define  BUFFSIZE 256
void perror_exit(char *msg);

/* Write() repeatedly until 'size' bytes are written */
int write_all(int fd, void *buff, size_t size) {
    int sent, n;
    for(sent = 0; sent < size; sent+=n) {
        if ((n = write(fd, buff+sent, size-sent)) == -1)
            return -1; /* error */
    }
    return sent;
}

int main(int argc, char *argv[]) {
    struct sockaddr_in  servadd;   /* The address of server */
    struct hostent *hp;          /* to resolve server ip */
    int    sock, n_read;     /* socket and message length */
    char   buffer[BUFFSIZE];       /* to receive message */
if ( argc != 3 ) {
        puts("Usage: rls <hostname> <directory>");exit(1);}
    /* Step 1: Get a socket */
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1 )
        perror_exit( "socket" );
    /* Step 2: lookup server's address and connect there */
    if ((hp = gethostbyname(argv[1])) == NULL) {
        herror("gethostbyname"); exit(1);}
    memcpy(&servadd.sin_addr, hp->h_addr, hp->h_length);
    servadd.sin_port = htons(PORTNUM);  /* set port number */
    servadd.sin_family = AF_INET ;     /* set socket type */
    if (connect(sock, (struct sockaddr*) &servadd,
            sizeof(servadd)) !=0)
        perror_exit( "connect" );
    /* Step 3: send directory name + newline */
    if ( write_all(sock, argv[2], strlen(argv[2])) == -1)
        perror_exit("write");
    if ( write_all(sock, "\n", 1) == -1 )
        perror_exit("write");
    /* Step 4: read back results and send them to stdout */
    while( (n_read = read(sock, buffer, BUFFSIZE)) > 0 )
        if (write_all(STDOUT_FILENO, buffer, n_read)<n_read)
            perror_exit("fwrite");
    close(sock);
    return 0;
}
```

Server on linux01

```
k24-syspro@linux01> ./rlsd
ls: cannot open directory /home/users/mema: Permission denied
```

Client on linux02

```
mema@linux02> . /rls linux01.di.uoa.gr /home/users/k24-syspro
project1
project2
project3
students.txt
mema@linux02> . /rls linux01.di.uoa.gr /home/users/mema/
mema@linux02>
```

One more example

The ROCK PAPER SCISSORS game

- One referee process.
- Two players: a local process, a remote process
- Referee talks to the local process through pipes
- Referee talks to the remote process through sockets

Referee (prsref) program

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>                    /* For wait */
#include <sys/types.h>                   /* For sockets */
#include <sys/socket.h>                  /* For sockets */
#include <netinet/in.h>                  /* For Internet socke
#include <netdb.h>                       /* For gethostbynam

#define READ    0
#define WRITE  1

int read_data (int fd, char *buffer);
int write_data (int fd, char* message);
void prs (int *score1, int *score2, int len1, int len2);

int main(int argc, char *argv[])
{
        int n, port, sock, newsock;
        int i, pid, fd1[2], fd2[2], option, status;
        int  score1=0, score2=0;         /* Score variables */
        char buf[60], buf2[60], buf3[60];  /* Buffers */
      /* prs options */
        char *message[] = { "ROCK", "PAPER", "SCISSOR
    unsigned int serverlen, clientlen;   /* Server - client var
        struct sockaddr_in server, client;
        struct sockaddr *serverptr, *clientptr;
        struct hostent *rem;
```

```
if ( argc < 3 ){                         /* At least 2 arguments */
                fprintf(stderr, "usage: %s <n> <port>\n", arg
                exit(0);
        }

        n = atoi(argv[1]);                                /* Number o
        port = atoi(argv[2]);                             /* Port */
    /* Create socket */
        if ((sock = socket(AF_INET, SOCK_STREAM, 0)) ==

                perror("socket");
                exit(-1);
        }
        server.sin_family = AF_INET;      /* Internet domain *
        server.sin_addr.s_addr = htonl(INADDR_ANY);
        server.sin_port = htons(port);     /* The given port */
        serverptr = (struct sockaddr *) &server;
        serverlen = sizeof server;
        if (bind(sock, serverptr, serverlen) < 0){
                perror("bind"); exit(-1);
        }
        if (listen(sock, 5) < 0){
                perror("listen");exit(-1);
        }

        printf("I am the referee with PID %d waiting for gam
      request at port %d\n", (int) getpid(), port);

        if (pipe (fd1) == -1){          /* First pipe: parent -> child
                perror("pipe");exit(-1);
```

```
}
if (pipe (fd2) == -1){        /* Second pipe: child -> pare
        perror("pipe");exit(-1);
}

if ((pid = fork()) == -1)     /* Create child for player 1 *
{
        perror("fork");exit(-1);
}

if ( !pid ){                  /* Child process */
        close(fd1[WRITE]);close(fd2[READ]);  /*Clos
        srand (getppid());
        printf("I am player 1 with PID %d\n", (int) g
        for(;;)                       /* While read "REA
        {
    /* Read "READY" or "STOP" */
                read_data (fd1[READ], buf);
                option = rand()%3;
                if ( strcmp("STOP", buf)){          /* I
        /* Send random option */
                        write_data (fd2[WRITE], me
        /* Read result of this game */
                        read_data (fd1[READ], buf);
        printf ("%s", buf);                /* Print resu
                }else
                        break;
        }
        read_data (fd1[READ], buf);         /* Read fi
        printf("%s", buf);          /* Print final result *
        close(fd1[READ]); close(fd2[WRITE]);
    }
    else{                   /* Parent process */
        clientptr = (struct sockaddr *) &client;
        clientlen = sizeof client;
        close(fd1[READ]); close(fd2[WRITE]);
        printf("Player 1 is child of the referee\n");
        if ((newsock = accept(sock, clientptr, &clientle
            perror("accept"); exit(-1);
        }
        if ((rem = gethostbyaddr((char *) &client.sin_add
      sizeof client.sin_addr.s_addr, client.sin_family))
      == NULL) {
            perror("gethostbyaddr");exit(-1);
        }

        printf("Player 2 connected %s\n",rem->h_na
        write_data (newsock, "2");  /* Send player's I
        for(i = 1; i <= n; i++){
            write_data (fd1[WRITE], "READY"
            write_data (newsock, "READY");
            read_data (fd2[READ], buf);
            read_data (newsock, buf2);
            /* Create result string */
            sprintf (buf3, "Player 1:%10s\tPlayer
          buf, buf2);
            write_data (fd1[WRITE], buf3);
            write_data (newsock, buf3);
            prs(&score1,&score2,strlen(buf),strle
                }

                /* Calculate final results for each player */
                if ( score1 == score2 ){
                        sprintf(buf, "Score = %d - %d (draw
                   score1, score2);
                        sprintf(buf2, "Score = %d - %d (dra
                   score1, score2);
                }else if (score1 > score2 ){
                        sprintf(buf, "Score = %d - %d (you v
                    score1, score2);
                        sprintf(buf2, "Score = %d - %d (play
                   score1, score2);
                }else{
                        sprintf(buf, "Score = %d - %d (playe
                   score1, score2);
                        sprintf(buf2, "Score = %d - %d (you
                   score1, score2);
                }
                write_data (fd1[WRITE], "STOP");
                write_data (fd1[WRITE], buf);
                close(fd1[WRITE]); close(fd2[READ]);
                wait(&status);    /* Wait child */
                write_data (newsock, "STOP");
                write_data (newsock, buf2);
                close(newsock);  /* Close socket */
        }
        return 0;
}
```

```
int read_data (int fd, char *buffer){ /* Read formated data*/
        char temp;int i = 0, length = 0;
        if ( read ( fd, &temp, 1 ) < 0 )        /* Get length of str
                exit (-3);
        length = temp;
        while ( i < length )        /* Read $length chars */
                if ( i < ( i+= read (fd, &buffer[i], length - i)))
                        exit (-3);
        return i;        /* Return size of string */
}

/*Write formated data*/
int write_data ( int fd, char* message ){
        char temp; int length = 0;
        length = strlen(message) + 1;        /* Find length of st
        temp = length;
        if( write (fd, &temp, 1) < 0 )        /* Send length firs
                exit (-2);
        if( write (fd, message, length) < 0 )        /* Send st
                exit (-2);
        return length;        /* Return size of string */
}
```

```
/* void prs(int *score1, int *score2, int len1, int len2):
*     Each option (PAPER, ROCK, SCISSORS) has a number of letters (5, 4,
*     PAPER wins ROCK, ROCK wins SCISSORS and SCISSORS win PAPE
*     This means, for the 1st player to be the winner the difference in the
*     number of letters must be equal to 3 (SCISSORS-PAPER) or 1 (PAPER-
*     or -4 (ROCK-SCISSORS). If not, then the 2nd player wins!
*     (If we have a zero, then we call it a draw and nobody get points)
*/
void prs(int *score1, int *score2, int len1, int len2)
{
     /* len1 = buf1 length, len2 = buf2 length */
    int result = len1 - len2;
    if (result == 3 || result == 1 || result == -4) /* 1st player win
        (*score1)++;
    else if (result)      /* 2nd player wins */
        (*score2)++;
    return;
}
```

Remote player (prs)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>                    /* For bcopy */
#include <unistd.h>
#include <sys/wait.h>                   /* For wait */
#include <sys/types.h>                  /* For sockets */
#include <sys/socket.h>                 /* For sockets */
#include <netinet/in.h>                 /* For Internet sockets */
#include <netdb.h>                      /* For gethostbyname */

int read_data (int fd, char *buffer);
int write_data (int fd, char* message);

int main (int argc, char *argv[])
{
        int i, port, sock, option;
        char opt[3], buf[60], *message[] =
            { "PAPER", "ROCK", "SCISSORS" };
        unsigned int serverlen;
        struct sockaddr_in server;
        struct sockaddr *serverptr;
        struct hostent *rem;
        if (argc < 3){      /* At least 2 arguments */
                fprintf(stderr, "usage: %s <domain> <port>\
               argv[0]);
                exit(-1);
        }
```

```
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
            perror("socket");
            exit(-1);
    }
/* Find server address */
    if ((rem = gethostbyname(argv[1])) == NULL){
            perror("gethostbyname");
            exit(-1);
    }
    port = atoi(argv[2]);
    server.sin_family = AF_INET;
    bcopy((char *) rem -> h_addr, (char *) &server.sin_ad
        rem -> h_length);
    server.sin_port = htons(port);
    serverptr = (struct sockaddr *) &server;
    serverlen = sizeof server;
    if (connect(sock, serverptr, serverlen) < 0){
            perror("connect");exit(-1);
    }

    read_data (sock, buf); /* Read player's ID (1 or 2) */
    printf("I am player %d with PID %d\n", buf[0]-'0',
        (int) getpid());
    for ( i = 1; ; i++ ){    /* While read "READY" */
            read_data (sock, buf);/* Read "READY" or "
            if ( strcmp("STOP", buf) ){  /* If != "STOP" *
                    printf("Give round %d play: ", i);
                    scanf("%s", opt);
                    switch (*opt){      /* First letter of op
                    /* Note: The other 2 are \n and \0 */
                         case 'p':option = 0; break;
                         case 'r':option = 1; break;
                         case 's':option = 2; break;
                         default: fprintf(stderr, "W
                        option %c\n", *opt);
                              option = ((int)*opt
                    }
                    write_data (sock, message[option]);
                    read_data (sock, buf);
                    printf ("%s", buf);
               }else break;
          }
          read_data (sock, buf);      /* Read final score */
          printf("%s", buf);
          close(sock);
          return 0;
}

/* Read formated data */
int read_data (int fd, char *buffer){
          char temp; int i = 0, length = 0;
          if ( read ( fd, &temp, 1 ) < 0 )  /* Get length of string
               exit (-3);
          length = temp;
          while ( i < length )          /* Read $length chars */
               if ( i < ( i+= read (fd, &buffer[i], length - i) )
                    exit (-3);
```

```
        return i;               /* Return size of string */
}

/* Write formated data */
int write_data ( int fd, char* message ){
        char temp; int length = 0;
        length = strlen(message) + 1;       /* Find length of strin
        temp = length;
        if ( write (fd, &temp, 1) < 0 )      /* Send length first */
                exit (-2);
        if ( write (fd, message, length) < 0 )          /* Send strin
                exit (-2);
        return length;                                  /* Return siz
}
```

Server

```
mema@bowser> ./prsref 3 2323
I am the referee with PID 25499 …
    waiting for game request at port 2323
I am player 1 with PID 25500
Player 1 is child of the referee
Player 2 connected localhost
Player 1:     PAPER     Player 2:     PAPER
Player 1:      ROCK     Player 2:  SCISSORS
Player 1:  SCISSORS     Player 2:  SCISSORS
Score = 1 - 0 (you won)
mema@bowser>
```

Client

```
mema@bowser> ./prs localhost 2323
I am player 2 with PID 25519
Give round 1 play: p
Player 1:     PAPER     Player 2:     PAPER
Give round 2 play: s
Player 1:      ROCK     Player 2:  SCISSORS
Give round 3 play: s
Player 1:  SCISSORS     Player 2:  SCISSORS
Score = 1 - 0 (player 1 won)
mema@bowser>
```