Επικοινωνία μεταξύ διεργασιών

- Μέσω σωλήνων
  - Για διεργασίες που έχουν κοινό πρόγονο ο οποίος έχει δημιουργήσει το σωλήνα
  - Επικοινωνία προς μία κατεύθυνση μόνο
  - Μία διεργασία γράφει στο άκρο γραψίματος του σωλήνα και η άλλη διαβάζει από το άκρο διαβάσματος
- Μέσω υποδοχών
  - Μεταξύ διεργασιών που οι ίδιες δημιουργούν τις υποδοχές
  - Επικοινωνία και προς τις δύο κατευθύνσεις
  - Οι διεργασίες μπορεί να εκτελούνται στον ίδιο υπολογιστή ή ακόμα και σε διαφορετικούς υπολογιστές που είναι συνδεδεμένοι μέσω ενός δικτύου

- Κλήση συστήματος **pipe**
  - int **pipe**(int *fd[ ]*)
  - Δημιουργεί ένα σωλήνα με άκρο διαβάσματος που αντιστοιχεί στον περιγραφέα *fd[0]* και άκρο γραψίματος που αντιστοιχεί στον περιγραφέα *fd[1]*
  - Απόγονοι της διεργασίας που δημιούργησε το σωλήνα, συμπεριλαμβανομένης και της ίδιας της διεργασίας-δημιουργού, μπορούν να επικοινωνήσουν μέσω του σωλήνα, που αποτελεί ουσιαστικά ένα είδος ενδιάμεσης μνήμης, γραφοντας με τη **write** και διαβάζοντας με τη **read** από το κατάλληλο άκρο
  - Τα μη χρησιμοποιούμενα άκρα πρέπει να ελευθερώνονται με την **close**
  - Η **pipe** επιστρέφει 0 σε επιτυχία ή -1 σε αποτυχία

- Χρήση της κλήσης **pipe**

```
/* File: pipe_demo.c */
#include <stdio.h>   /* For printf */
#define READ  0      /* Read end of pipe */
#define WRITE 1      /* Write end of pipe */
char *phrase = "This is a test phrase.";
main()
{  int pid, fd[2], bytes;
   char message[100];
   if (pipe(fd) == -1) {         /* Create a pipe */
      perror("pipe");
      exit(1); }
   if ((pid = fork()) == -1) {   /* Fork a child */
      perror("fork");
      exit(1); }
   if (pid == 0) {               /* Child, writer */
      close(fd[READ]);           /* Close unused end */
      write(fd[WRITE], phrase, strlen(phrase)+1);
      close(fd[WRITE]); }        /* Close used end */
   else {                        /* Parent, reader */
      close(fd[WRITE]);          /* Close unused end */
      bytes = read(fd[READ], message, sizeof(message));
      printf("Read %d bytes: %s\n", bytes, message);
      close(fd[READ]); } }       /* Close used end */
```

```
% pipe_demo
Read 23 bytes: This is a test phrase.
%
```

- Πεδία υποδοχών
  - `AF_INET`
    - Πεδίο Internet
    - Επικοινωνία και σε διαφορετικούς υπολογιστές
    - Διεύθυνση: Internet διεύθυνση και αριθμός θύρας
  - `AF_UNIX`
    - Πεδίο Unix
    - Επικοινωνία στον ίδιο υπολογιστή
    - Διεύθυνση: Κόμβος στο σύστημα αρχείων
- Είδη υποδοχών
  - `SOCK_STREAM`
    - Υποδοχές ροής (TCP)
  - `SOCK_DGRAM`
    - Τηλεγραφικές υποδοχές (UDP)

	TCP	UDP
Απαίτηση σύνδεσης	ΝΑΙ	ΟΧΙ
Αξιοπιστία	ΝΑΙ	ΟΧΙ
Όρια μηνυμάτων	ΟΧΙ	ΝΑΙ
Διαδοχικότητα μηνυμάτων	ΝΑΙ	ΟΧΙ

- Συναρτήσεις βιβλιοθήκης **htons**, **htonl**, **ntohs** και **ntohl**
  - 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>`
- Συναρτήσεις βιβλιοθήκης **gethostbyname** και **gethostbyaddr**
  - struct hostent ***gethostbyname**(char **name*)
  - struct hostent ***gethostbyaddr**(char **addr*, int *len*, int *type*)
  - Επιστροφή ενός δείκτη σε δομή struct hostent για έναν υπολογιστή είτε δεδομένου του ονόματός του *name*, όπου στο πεδίο h_addr της δομής βρίσκεται η Internet διεύθυνσή του και στο πεδίο h_length το μέγεθός της, είτε δεδομένης της διεύθυνσής του *addr*, του μεγέθους της *len* και του είδους της *type* (πάντα `AF_INET`), όπου στο πεδίο h_name της επιστρεφόμενης δομής βρίσκεται το όνομα του υπολογιστή
  - Απαίτηση: `#include <netdb.h>`

- Όλες οι κλήσεις συστήματος που ακολουθούν και χρησιμοποιούνται για διαχείριση υποδοχών απαιτούν
  - `#include <sys/types.h>`
  - `#include <sys/socket.h>`

  και επιστρεφουν -1 σε περίπτωση αποτυχίας
- Κλήση συστήματος **socket**
  - int **socket**(int *domain*, int *type*, int *protocol*)
  - Δημιουργεί μία υποδοχή και επιστρέφει τον περιγραφέα αρχείου που αντιστοιχεί σ'αυτήν
  - Το *domain* πρέπει να είναι `AF_INET` ή `AF_UNIX`
  - Το *type* πρέπει να είναι `SOCK_STREAM` ή `SOCK_DGRAM`
  - Σαν *protocol*, πάντα δίνουμε το default (`0`)

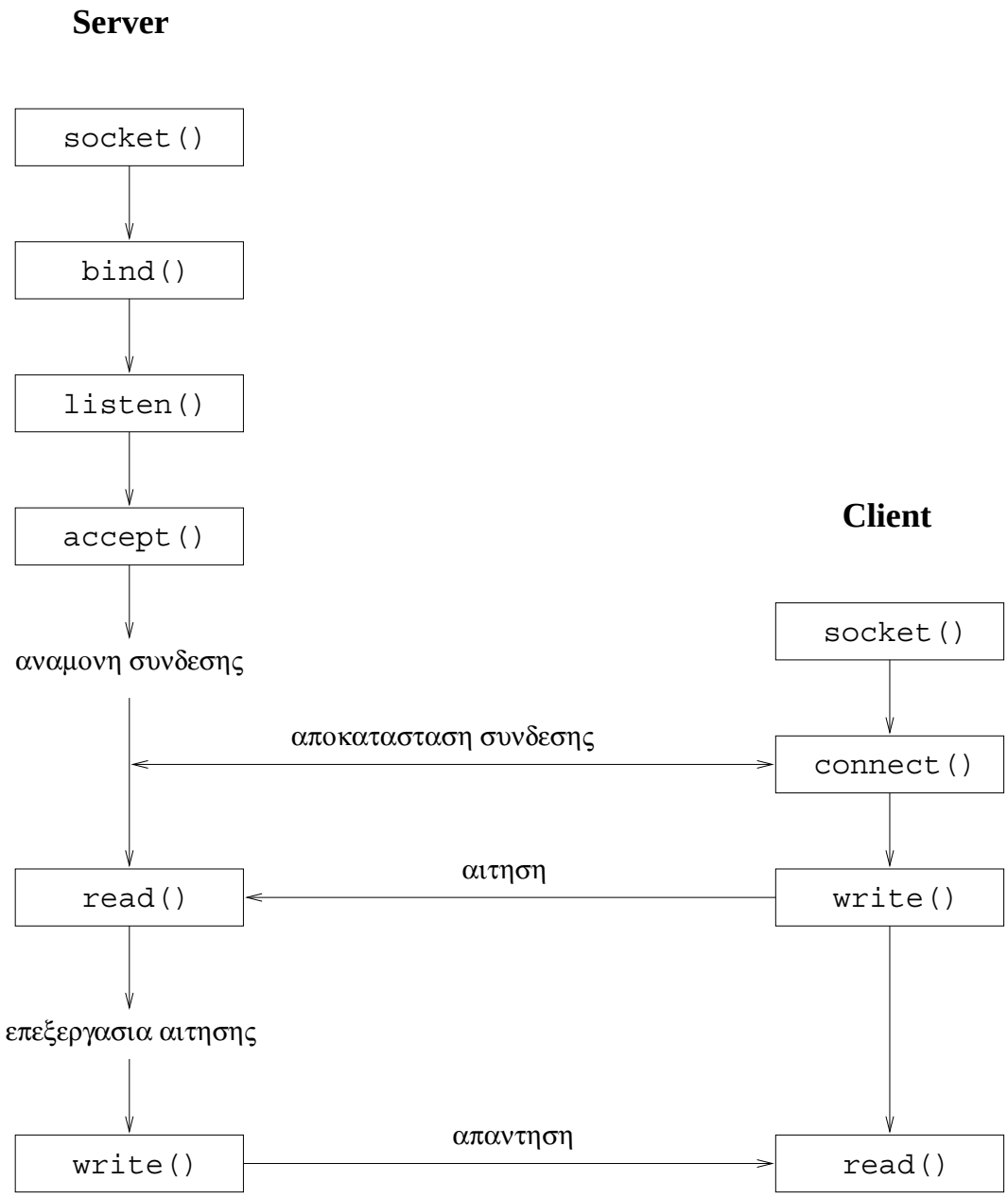
- Κλήση συστήματος **bind**
  - int **bind**(int *fd*, struct sockaddr **address*, int *addresslen*)
  - Συνδέει την υποδοχή που αντιστοιχεί στον περιγραφέα αρχείου *fd* με ένα όνομα/διεύθυνση **address*
  - Πεδίο Internet
    - Ορίζεται ένα struct sockaddr_in `name` και δίνονται τα `AF_INET` στο πεδίο `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>`
  - Πεδίο Unix
    - Ορίζεται ένα struct sockaddr_un `name` και δίνονται τα `AF_UNIX` στο πεδίο `name.sun_family` και `path` στο πεδίο `name.sun_path`, όπου `path` είναι το όνομα-μονοπάτι του κόμβου που χρησιμοποιείται στο σύστημα αρχείων και αφού η διεύθυνση του `name` προσαρμοσθεί σε (struct sockaddr *) δίνεται στο *address*
    - Απαίτηση: `#include <sys/un.h>`
  - Το μέγεθος του `name` δίνεται στο *addresslen*

- 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*, 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
- Κλήση συστήματος **connect**
  - int **connect**(int *fd*, struct sockaddr **address*, 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*

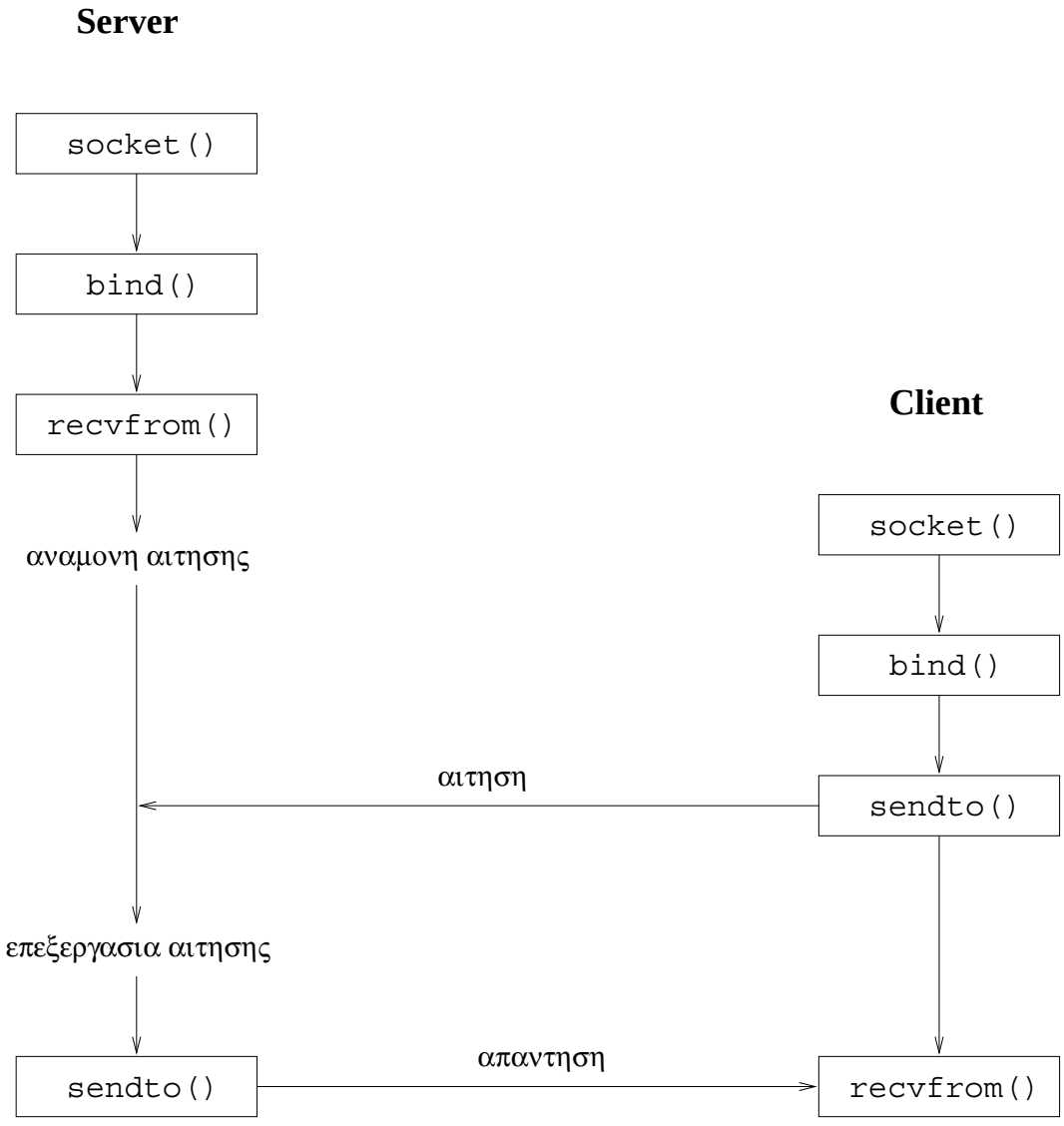
- Κλήσεις συστήματος **recvfrom** και **sendto**
  - int **recvfrom**(int *fd*, char **buf*, int *count*, int *flags*,
    struct sockaddr **address*, int **addresslen*)
  - int **sendto**(int *fd*, char **buf*, int *count*, int *flags*,
    struct sockaddr **address*, int *addresslen*)
  - Χρησιμοποιούνται αντί των **read** και **write** για την παραλαβή και την αποστολή μηνυμάτων μέσω τηλεγραφικών υποδοχών
  - Τα *fd*, *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**)

- Κλήση συστήματος **getsockname**
  - int **getsockname**(int *fd*, struct sockaddr **address*, int **addresslen*)
  - Επιστρέφει στη δομή **address* τη διεύθυνση με την οποία έχει συνδεθεί η υποδοχή που αντιστοιχεί στον περιγραφέα αρχείου *fd* και στο **addresslen* το μέγεθος της διεύθυνσης αυτής
  - Είναι χρήσιμη στην περίπτωση που δεν προκαθορισθεί συγκεκριμένος αριθμός θύρας κατά τη σύνδεση μέσω της **bind** μίας υποδοχής με μία διεύθυνση (με απόδοση του 0 στο `name.sin_port`) για να βρεθεί ο αριθμός της πραγματικής θύρας που διατέθηκε από το σύστημα για τη σύνδεση
- 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*

- TCP επικοινωνία client/server



- UDP επικοινωνία client/server



- *Να γραφούν client και server προγράμματα C που να επιδεικνύουν την επικοινωνία μέσω υποδοχών ροής στο πεδίο Internet.*

```
/* File: int_str_server.c */
#include <sys/types.h>                                    /* For sockets */
#include <sys/socket.h>                                   /* For sockets */
#include <netinet/in.h>                                /* For Internet sockets */
#include <netdb.h>                                        /* For gethostbyaddr */
#include <stdio.h>                                                /* For I/O */

void reverse(char *);

main(int argc, char *argv[])         /* Server with Internet stream sockets */
{  int port, sock, newsock, serverlen, clientlen; char buf[256];
   struct sockaddr_in server, client;
   struct sockaddr *serverptr, *clientptr;
   struct hostent *rem;
   if (argc < 2) {                /* Check if server's port number is given */
      printf("Please give the port number\n");
      exit(1); }
   if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {     /* Create socket */
      perror("socket"); exit(1); }
   port = atoi(argv[1]);                   /* Convert port number to integer */
   server.sin_family = AF_INET;                              /* Internet domain */
   server.sin_addr.s_addr = htonl(INADDR_ANY);       /* My Internet address */
   server.sin_port = htons(port);                            /* The given port */
   serverptr = (struct sockaddr *) &server;
   serverlen = sizeof server;
   if (bind(sock, serverptr, serverlen) < 0) { /* Bind socket to an address */
      perror("bind"); exit(1); }
   if (listen(sock, 1) < 0) {                      /* Listen for connections */
      perror("listen"); exit(1); }
   printf("Listening for connections to port %d\n", port);
```

```
   while(1) {
      clientptr = (struct sockaddr *) &client;
      clientlen = sizeof client;
      if ((newsock = accept(sock, clientptr, &clientlen)) < 0) {
         perror("accept"); exit(1); }                  /* Accept connection */
      if ((rem = gethostbyaddr((char *) &client.sin_addr.s_addr,
            sizeof client.sin_addr.s_addr,         /* Find client's address */
            client.sin_family)) == NULL) {
         perror("gethostbyaddr"); exit(1); }
      printf("Accepted connection from %s\n", rem -> h_name);
      switch (fork()) {              /* Create child for serving the client */
         case -1:
            perror("fork"); exit(1);
         case 0:                                        /* Child process */
            do {
               bzero(buf, sizeof buf);                 /* Initialize buffer */
               if (read(newsock, buf, sizeof buf) < 0) { /* Receive message */
                  perror("read"); exit(1); }
               printf("Read string: %s\n", buf);
               reverse(buf);                             /* Reverse message */
               if (write(newsock, buf, sizeof buf) < 0) {   /* Send message */
                  perror("write"); exit(1); }
            } while (strcmp(buf, "dne") != 0);           /* Finish on "end" */
            close(newsock);                                 /* Close socket */
            exit(0); } } }

void reverse(char *s)                    /* Function for reversing a string */
{  int c, i, j;
   for (i = 0, j = strlen(s) - 1 ; i < j ; i++, j--) {
      c = s[i];
      s[i] = s[j];
      s[j] = c; } }
```

```
/* File: int_str_client.c */
#include <sys/types.h>                                    /* For sockets */
#include <sys/socket.h>                                   /* For sockets */
#include <netinet/in.h>                                /* For Internet sockets */
#include <netdb.h>                                        /* For gethostbyname */
#include <stdio.h>                                                 /* For I/O */
main(int argc, char *argv[])         /* Client with Internet stream sockets */
{  int port, sock, serverlen; char buf[256];
   struct sockaddr_in server;
   struct sockaddr *serverptr;
   struct hostent *rem;
   if (argc < 3) { /* Check if server's host name and port number are given */
      printf("Please give the host name and the port number\n"); exit(1); }
   if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {     /* Create socket */
      perror("socket"); exit(1); }
   if ((rem = gethostbyname(argv[1])) == NULL) {   /* Find server's address */
      perror("gethostbyname"); exit(1); }
   port = atoi(argv[2]);                  /* Convert port number to integer */
   server.sin_family = AF_INET;                           /* Internet domain */
   bcopy((char *) rem -> h_addr, (char *) &server.sin_addr, rem -> h_length);
   server.sin_port = htons(port);     /* Server's Internet address and port */
   serverptr = (struct sockaddr *) &server;
   serverlen = sizeof server;
   if (connect(sock, serverptr, serverlen) < 0) {     /* Request connection */
      perror("connect"); exit(1); }
   printf("Requested connection to host %s port %d\n", argv[1], port);
   do {
      bzero(buf, sizeof buf);                          /* Initialize buffer */
      printf("Give input string: ");
      gets(buf);                                  /* Read message from stdin */
      if (write(sock, buf, sizeof buf) < 0) {               /* Send message */
         perror("write"); exit(1); }
      bzero(buf, sizeof buf);                          /* Initialize buffer */
      if (read(sock, buf, sizeof buf) < 0) {             /* Receive message */
         perror("read"); exit(1); }
      printf("Read string:       %s\n", buf);
   } while (strcmp(buf, "dne") != 0);                    /* Finish on "end" */
   close(sock);                                             /* Close socket */
   exit(0); }
```

```
% int_str_server 30000
Listening for connections to port 30000
Accepted connection from daphne.di.uoa.gr
Read string: test
Read string: A string
Read string: niconanomimatamimonanocin
Read string: This is a test line
Read string: Teleiosame
Read string: end
^C%
```

```
% int_str_client daphne 30000
Requested connection to host daphne port 30000
Give input string: test
Read string:       tset
Give input string: A string
Read string:       gnirts A
Give input string: niconanomimatamimonanocin
Read string:       niconanomimatamimonanocin
Give input string: This is a test line
Read string:       enil tset a si sihT
Give input string: Teleiosame
Read string:       emasoieleT
Give input string: end
Read string:       dne
%
```

- *Να γραφούν client και server προγράμματα C που να επιδεικνύουν την επικοινωνία μέσω υποδοχών ροής στο πεδίο Unix.*

```
/* File: un_str_server.c */
#include <sys/types.h>                                    /* For sockets */
#include <sys/socket.h>                                   /* For sockets */
#include <sys/un.h>                                  /* For Unix sockets */
#include <stdio.h>                                            /* For I/O */
#include <signal.h>                                       /* For signals */

void reverse(char *);
void sigchld_handler();

main(int argc, char *argv[])             /* Server with Unix stream sockets */
{  int sock, newsock, serverlen, clientlen; char buf[256];
   struct sockaddr_un server, client;
   struct sockaddr *serverptr, *clientptr;
   if (argc < 2) {                     /* Check if socket filename is given */
      printf("Please give the socket filename\n");
      exit(1); }
   signal(SIGCHLD, sigchld_handler); /* To be informed on child termination */
   if ((sock = socket(AF_UNIX, SOCK_STREAM, 0)) < 0) {     /* Create socket */
      perror("socket"); exit(1); }
   server.sun_family = AF_UNIX;                               /* Unix domain */
   strcpy(server.sun_path, argv[1]);                     /* My Unix address */
   unlink(argv[1]);                  /* Remove socket filename if it exists */
   serverptr = (struct sockaddr *) &server;
   serverlen = sizeof server;
   if (bind(sock, serverptr, serverlen) < 0) { /* Bind socket to an address */
      perror("bind"); exit(1); }
   if (listen(sock, 1) < 0) {                     /* Listen for connections */
      perror("listen"); exit(1); }
   printf("Listening for connections to socket %s\n", argv[1]);
```

```
   while(1) {
      clientptr = (struct sockaddr *) &client;
      clientlen = sizeof client;
      while ((newsock = accept(sock, clientptr, &clientlen)) < 0);
      printf("Accepted connection\n");                  /* Accept connection */
      switch (fork()) {              /* Create child for serving the client */
         case -1:
            perror("fork"); exit(1);
         case 0:                                          /* Child process */
            do {
               bzero(buf, sizeof buf);                 /* Initialize buffer */
               if (read(newsock, buf, sizeof buf) < 0) { /* Receive message */
                  perror("read"); exit(1); }
               printf("Read string: %s\n", buf);
               reverse(buf);                             /* Reverse message */
               if (write(newsock, buf, sizeof buf) < 0) {   /* Send message */
                  perror("write"); exit(1); }
            } while (strcmp(buf, "dne") != 0);           /* Finish on "end" */
            close(newsock);                                 /* Close socket */
            exit(0); } } }

void reverse(char *s)                    /* Function for reversing a string */
{  int c, i, j;
   for (i = 0, j = strlen(s) - 1 ; i < j ; i++, j--) {
      c = s[i];
      s[i] = s[j];
      s[j] = c; } }

void sigchld_handler()     /* Handler for SIGCHLD accepts child termination */
{  int status;
   wait(&status); }
```

```
/* File: un_str_client.c */
#include <sys/types.h>                                    /* For sockets */
#include <sys/socket.h>                                   /* For sockets */
#include <sys/un.h>                                    /* For Unix sockets */
#include <stdio.h>                                           /* For I/O */
main(int argc, char *argv[])             /* Client with Unix stream sockets */
{  int sock, serverlen; char buf[256];
   struct sockaddr_un server;
   struct sockaddr *serverptr;
   if (argc < 2) {                     /* Check if socket filename is given */
      printf("Please give the socket filename\n"); exit(1); }
   if ((sock = socket(AF_UNIX, SOCK_STREAM, 0)) < 0) {     /* Create socket */
      perror("socket"); exit(1); }
   server.sun_family = AF_UNIX;                              /* Unix domain */
   strcpy(server.sun_path, argv[1]);               /* Server's Unix address */
   serverptr = (struct sockaddr *) &server;
   serverlen = sizeof server;
   if (connect(sock, serverptr, serverlen) < 0) {     /* Request connection */
      perror("connect"); exit(1); }
   printf("Requested connection to socket %s\n", argv[1]);
   do {
      bzero(buf, sizeof buf);                          /* Initialize buffer */
      printf("Give input string: ");
      gets(buf);                                  /* Read message from stdin */
      if (write(sock, buf, sizeof buf) < 0) {              /* Send message */
         perror("write"); exit(1); }
      bzero(buf, sizeof buf);                          /* Initialize buffer */
      if (read(sock, buf, sizeof buf) < 0) {             /* Receive message */
         perror("read"); exit(1); }
      printf("Read string:       %s\n", buf);
   } while (strcmp(buf, "dne") != 0);                    /* Finish on "end" */
   close(sock);                                             /* Close socket */
   exit(0); }
```

```
% un_str_server str_socket
Listening for connections to socket str_socket
Accepted connection
Read string: Testing stream sockets in the Unix domain...
Read string: abcdefghijklmnopqrstuvwxyz
Read string: >>>>>>><<<<<<
Read string: Is it OK?
Read string: Fine!
Read string: end
^C% rm str_socket
%
```

```
% un_str_client str_socket
Requested connection to socket str_socket
Give input string: Testing stream sockets in the Unix domain...
Read string:       ...niamod xinU eht ni stekcos maerts gnitseT
Give input string: abcdefghijklmnopqrstuvwxyz
Read string:       zyxwvutsrqponmlkjihgfedcba
Give input string: >>>>>>><<<<<<
Read string:       <<<<<<>>>>>>
Give input string: Is it OK?
Read string:       ?KO ti sI
Give input string: Fine!
Read string:       !eniF
Give input string: end
Read string:       dne
%
```

- *Να γραφούν client και server προγράμματα C που να επιδεικνύουν την επικοινωνία μέσω τηλεγραφικών υποδοχών στο πεδίο Internet.*

```
/* File: int_dgr_server.c */
#include <sys/types.h>                                    /* For sockets */
#include <sys/socket.h>                                   /* For sockets */
#include <netinet/in.h>                                /* For Internet sockets */
#include <netdb.h>                                        /* For gethostbyaddr */
#include <stdio.h>                                                 /* For I/O */
main(int argc, char *argv[])       /* Server with Internet datagram sockets */
{  int n, port, sock, serverlen, clientlen; char buf[256];
   struct sockaddr_in server, client;
   struct sockaddr *serverptr, *clientptr;
   struct hostent *rem;
   if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {      /* Create socket */
      perror("socket"); exit(1); }
   server.sin_family = AF_INET;                              /* Internet domain */
   server.sin_addr.s_addr = htonl(INADDR_ANY);       /* My Internet address */
   server.sin_port = htons(0);                               /* Select any port */
   serverptr = (struct sockaddr *) &server;
   serverlen = sizeof server;
   if (bind(sock, serverptr, serverlen) < 0) { /* Bind socket to an address */
      perror("bind"); exit(1); }
   if (getsockname(sock, serverptr, &serverlen) < 0) {     /* Selected port */
      perror("getsockname"); exit(1); }
   printf("Socket port: %d\n", ntohs(server.sin_port));
   clientptr = (struct sockaddr *) &client;
   clientlen = sizeof client;
   while(1) {
      bzero(buf, sizeof buf);                             /* Initialize buffer */
      if ((n = recvfrom(sock, buf, sizeof buf, 0, clientptr, &clientlen)) < 0) {
         perror("recvfrom"); exit(1); }                    /* Receive message */
      if ((rem = gethostbyaddr((char *) &client.sin_addr.s_addr,
            sizeof client.sin_addr.s_addr, client.sin_family)) == NULL) {
         perror("gethostbyaddr"); exit(1); }       /* Find client's address */
      printf("Received from %s: %s\n", rem -> h_name, buf);
      if (sendto(sock, buf, n, 0, clientptr, clientlen) < 0) {
         perror("sendto"); exit(1); } } }                    /* Send message */
```

```
/* File: int_dgr_client.c */
#include <sys/types.h>                                    /* For sockets */
#include <sys/socket.h>                                   /* For sockets */
#include <netinet/in.h>                                /* For Internet sockets */
#include <netdb.h>                                        /* For gethostbyname */
#include <stdio.h>                                                 /* For I/O */
main(int argc, char *argv[])       /* Client with Internet datagram sockets */
{  int port, sock, serverlen, clientlen; char buf[256];
   struct sockaddr_in server, client;
   struct sockaddr *serverptr, *clientptr;
   struct hostent *rem;
   if (argc < 3) { /* Check if server's host name and port number are given */
      printf("Please give the host name and the port number\n"); exit(1); }
   if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {      /* Create socket */
      perror("socket"); exit(1); }
   if ((rem = gethostbyname(argv[1])) == NULL) {   /* Find server's address */
      perror("gethostbyname"); exit(1); }
   port = atoi(argv[2]);                  /* Convert port number to integer */
   server.sin_family = AF_INET;                           /* Internet domain */
   bcopy((char *) rem -> h_addr, (char *) &server.sin_addr, rem -> h_length);
   server.sin_port = htons(port);     /* Server's Internet address and port */
   serverptr = (struct sockaddr *) &server;
   serverlen = sizeof server;
   client.sin_family = AF_INET;                           /* Internet domain */
   client.sin_addr.s_addr = htonl(INADDR_ANY);       /* My Internet address */
   client.sin_port = htons(0);                            /* Select any port */
   clientptr = (struct sockaddr *) &client;
   clientlen = sizeof client;
   if (bind(sock, clientptr, clientlen) < 0) { /* Bind socket to an address */
      perror("bind"); exit(1); }
   while (gets(buf) != NULL) {     /* Read continuously messages from stdin */
      if (sendto(sock, buf, strlen(buf)+1, 0, serverptr, serverlen) < 0) {
         perror("sendto"); exit(1); }                      /* Send message */
      bzero(buf, sizeof buf);     /* Initialize buffer */
      if (recvfrom(sock, buf, sizeof buf, 0, serverptr, &serverlen) < 0) {
         perror("recvfrom"); exit(1); }                 /* Receive message */
      printf("%s\n", buf); } }
```

```
% int_dgr_server
Socket port: 1216
Received from daphne.di.uoa.gr: #!/bin/sh
Received from daphne.di.uoa.gr: #@(#)ready      7.1 (ULTRIX) 7/22/92
Received from daphne.di.uoa.gr: while :
Received from daphne.di.uoa.gr: do
Received from daphne.di.uoa.gr:         sleep 2
Received from daphne.di.uoa.gr:         echo -n "
Received from daphne.di.uoa.gr: Are you ready? (y/n): "
Received from daphne.di.uoa.gr:         read X
Received from daphne.di.uoa.gr:         case "$X" in
Received from daphne.di.uoa.gr:         [yY]*)  break
Received from daphne.di.uoa.gr:                 ;;
Received from daphne.di.uoa.gr:         esac
Received from daphne.di.uoa.gr: done
^C%
```

```
% int_dgr_client daphne 1216 < /etc/ready
#!/bin/sh
#@(#)ready      7.1 (ULTRIX) 7/22/92
while :
do
        sleep 2
        echo -n "
Are you ready? (y/n): "
        read X
        case "$X" in
        [yY]*)  break
                ;;
        esac
done
%
```

- *Να γραφούν client και server προγράμματα C που να επιδεικνύουν την επικοινωνία μέσω τηλεγραφικών υποδοχών στο πεδίο Unix.*

```
/* File: un_dgr_server.c */
#include <sys/types.h>                                    /* For sockets */
#include <sys/socket.h>                                   /* For sockets */
#include <sys/un.h>                                    /* For Unix sockets */
#include <stdio.h>                                            /* For I/O */
main(int argc, char *argv[])           /* Server with Unix datagram sockets */
{  int n, sock, serverlen, clientlen; char buf[256];
   struct sockaddr_un server, client;
   struct sockaddr *serverptr, *clientptr;
   if (argc < 2) {                     /* Check if socket filename is given */
      printf("Please give the socket filename\n"); exit(1); }
   if ((sock = socket(AF_UNIX, SOCK_DGRAM, 0)) < 0) {      /* Create socket */
      perror("socket"); exit(1); }
   server.sun_family = AF_UNIX;                              /* Unix domain */
   strcpy(server.sun_path, argv[1]);                     /* My Unix address */
   unlink(argv[1]);                  /* Remove socket filename if it exists */
   serverptr = (struct sockaddr *) &server;
   serverlen = sizeof server;
   if (bind(sock, serverptr, serverlen) < 0) { /* Bind socket to an address */
      perror("bind"); exit(1); }
   printf("Waiting for data to ping\n");
   clientptr = (struct sockaddr *) &client;
   clientlen = sizeof client;
   while(1) {
      bzero(buf, sizeof buf);                          /* Initialize buffer */
      if ((n = recvfrom(sock, buf, sizeof buf, 0, clientptr, &clientlen)) < 0) {
         perror("recvfrom"); exit(1); }                  /* Receive message */
      printf("Received from %s: %s\n", client.sun_path, buf);
      if (sendto(sock, buf, n, 0, clientptr, clientlen) < 0) {
         perror("sendto"); exit(1); } } }                   /* Send message */
```

```
/* File: un_dgr_client.c */
#include <sys/types.h>                                    /* For sockets */
#include <sys/socket.h>                                   /* For sockets */
#include <sys/un.h>                                  /* For Unix sockets */
#include <stdio.h>                                           /* For I/O */
main(int argc, char *argv[])           /* Client with Unix datagram sockets */
{  int sock, serverlen, clientlen; char buf[256], sname[20];
   struct sockaddr_un server, client;
   struct sockaddr *serverptr, *clientptr;
   if (argc < 2) {                     /* Check if socket filename is given */
      printf("Please give the socket filename\n"); exit(1); }
   if ((sock = socket(AF_UNIX, SOCK_DGRAM, 0)) < 0) {      /* Create socket */
      perror("socket"); exit(1); }
   server.sun_family = AF_UNIX;                                /* Unix domain */
   strcpy(server.sun_path, argv[1]);               /* Server's Unix address */
   serverptr = (struct sockaddr *) &server;
   serverlen = sizeof server;
   sprintf(sname, "s%d", getpid());     /* Create my socket filename (=PID) */
   client.sun_family = AF_UNIX;                                /* Unix domain */
   strcpy(client.sun_path, sname);                        /* My Unix address */
   clientptr = (struct sockaddr *) &client;
   clientlen = sizeof client;
   if (bind(sock, clientptr, clientlen) < 0) { /* Bind socket to an address */
      perror("bind"); exit(1); }
   while (gets(buf) != NULL) {     /* Read continuously messages from stdin */
      if (sendto(sock, buf, strlen(buf)+1, 0, serverptr, serverlen) < 0) {
         perror("sendto"); exit(1); }                        /* Send message */
      bzero(buf, sizeof buf);                           /* Initialize buffer */
      if (recvfrom(sock, buf, sizeof buf, 0, serverptr, &serverlen) < 0) {
         perror("recvfrom"); exit(1); }                   /* Receive message */
      printf("%s\n", buf); }
   unlink(sname); }                              /* Remove my socket filename */
```

```
% un_dgr_server dgr_socket
Waiting for data to ping
Received from s3520: #  @(#).login      4.1     ULTRIX  7/2/90
Received from s3520: tset -I -Q
Received from s3520: set mail=/usr/spool/mail/root
Received from s3520: set prompt="# "
^C% rm dgr_socket
%
```

```
% cat /.proto...login | un_dgr_client dgr_socket
#       @(#).login      4.1     ULTRIX  7/2/90
tset -I -Q
set mail=/usr/spool/mail/root
set prompt="# "
%
```