Περιγραφητές αρχείων μετά από execl…

- Για όλες τις συναρτήσεις execl, execv, execlp, execvp…
- Οι περιγραφητές αρχείων παραμένουν ανοικτοί για το καλούμενο από την exec πρόγραμμα
- Πώς να χρησιμοποιηθούν όμως, αφού όχι κοινές μεταβλητές?
  - Χρήση περιγραφητών 0,1,2
  - Πέρασμα περιγραφητών ως ορίσματα στο καλούμενο πρόγραμμα
  - Παράδειγμα connectTest.c και connector.c

connectTest.c

```
/* File: connectTest.c */
#include <stdio.h> /* For printf */
#include <stdlib.h> /* For exit */
#include <unistd.h> /* For execlp */
#include <sys/stat.h> /* for open */
#include <fcntl.h> /* for open */
#define READ 0 /* Read end of pipe */
#define WRITE 1 /* Write end of pipe */
main(int argc, char *argv[]) {
  int fd[2], pid, fd2[2], filefd;
  if ((filefd =
        open("testFile", O_WRONLY | O_CREAT, 0666)) == -1) {
    perror("file"); exit(1); }
  if (pipe(fd) == -1) { /* Create a pipe */
    perror("pipe"); exit(1); }
  if (pipe(fd2) == -1) { /* Create a second pipe */
    perror("pipe2"); exit(1); }
  if ((pid = fork()) == -1) { /* Fork a child */
    perror("fork"); exit(1); }
  if (pid != 0) { /* Parent, writer */
    close(filefd); close(fd[READ]); close(fd[WRITE]);
    close(fd2[READ]); close(fd2[WRITE]);
    if (wait(NULL) != pid) {
        perror("Waiting for child completion:"); exit(-1); } }
  else { /* Child, reader */
    printf("Filefd = %d, fd[WRITE] = %d, fd2[WRITE] = %d\n",
                filefd, fd[WRITE], fd2[WRITE]);
    dup2(fd2[WRITE], 11); /* Duplicate write end to stdout */
    execlp(argv[1], argv[1], "11", NULL); /* Execute argv[1] */
    perror("execlp"); } }
```

Τι θα γίνει αν κλείσω εδώ και τα READ άκρα? - SIGPIPE

connector.c

```
/* File: connector.c */
#include <stdio.h> /* For printf */
#include <stdlib.h> /* For exit */
#include <unistd.h> /* For execlp */
#include <sys/stat.h> /* for open */
#include <fcntl.h> /* for open */
#include <string.h>
int main(int argc, char *argv[])
{
  char message[]="HELLO WORLD";
  if (write(3, message, strlen(message) + 1) == -1)
        perror("Write to 3-file");
  else printf("Write to file succeeded\n");
  if (write(5, message, strlen(message) + 1) == -1)
        perror("Write to 5-pipe1");
  else printf("Write to pipe1 succeeded\n");
  if (write(7, message, strlen(message) + 1) == -1)
        perror("Write to 7-pipe2");
  else printf("Write to pipe2 succeeded\n");
  if (write(11, message, strlen(message) + 1) == -1)
        perror("Write to 11-dup2");
  else printf("Write to dup2 succeeded\n");
  if (write(13, message, strlen(message) + 1) == -1)
        perror("Write to 13-Invalid");
  else printf("Write to invalid succeeded\n");
  return 1;
}
```

Εκτέλεση ελέγχου

```
linux03:/home/users/spro/sockets>./connectTest connector
Filefd = 3, fd[WRITE] = 5, fd2[WRITE] = 7
Write to file succeeded
Write to pipe1 succeeded
Write to pipe2 succeeded
Write to dup2 succeeded
Write to 13-Invalid: Bad file descriptor
linux03:/home/users/spro/sockets>
```

Παρόμοια, ανακατεύθυνση stdout

```
/* whotofile.c
 *      purpose: show how to redirect output for another program
 *         idea: fork, then in the child, redirect output, then exec
 */
#include        <stdio.h>
#include        <stdlib.h>
#include        <unistd.h>

main()
{
        int     pid ;
        int     fd;

        printf("About to run who into a file\n");

        /* create a new process or quit */
        if( (pid = fork() ) == -1 ){
                perror("fork"); exit(1);
        }
        /* child does the work */
        if ( pid == 0 ){
                close(1);                       /* close, */
                fd = creat( "userlist", 0644 );  /* then open */
                execlp( "who", "who", NULL );    /* and run      */
                perror("execlp");
                exit(1);
        }
        /* parent waits then reports */
        if ( pid != 0 ){
                wait(NULL);
                printf("Done running who.  results in userlist\n");
        }
}
```

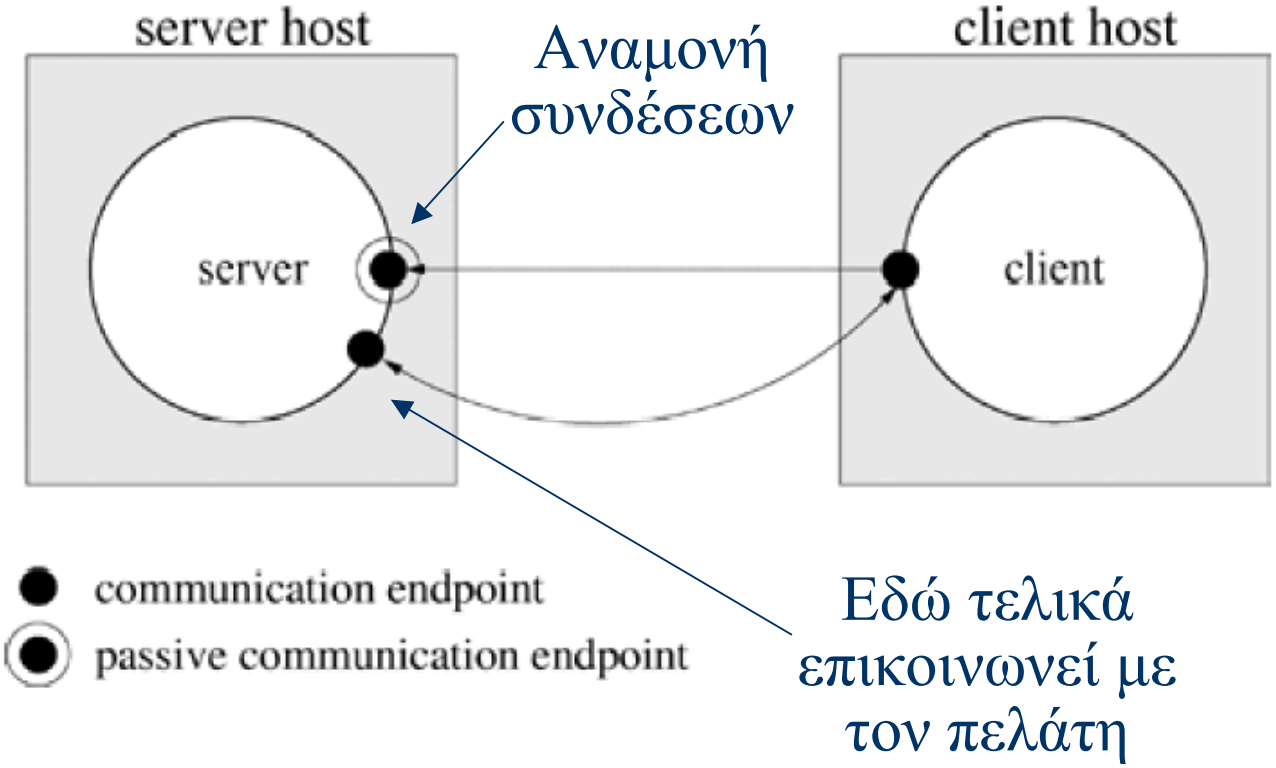
Μικρότερος διαθέσιμος περιγραφητής είναι το 1!!!

Επικοινωνία διεργασιών σε διαφορετικούς υπολογιστές

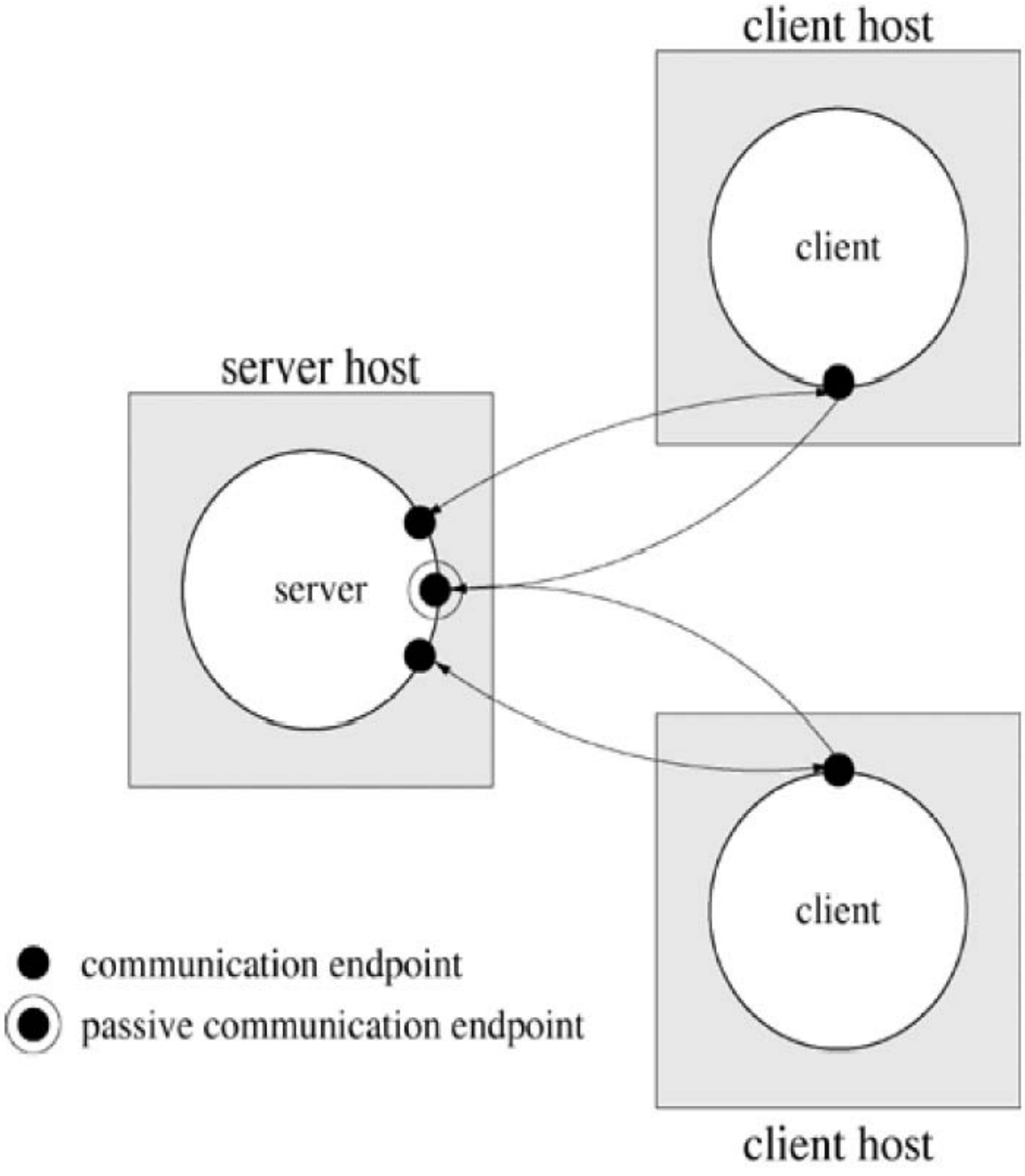
- Συνήθως μοντέλο πελάτη-εξυπηρετητή (client-server)
- Ο εξυπηρετητής παρέχει κάποια υπηρεσία
- Ο εξυπηρετητής περιμένει να δεχτεί συνδέσεις από πελάτες
- Πολλοί πελάτες συνδέονται για να χρησιμοποιήσουν την υπηρεσία
- Πελάτης-εξυπηρετητής μιλάνε απευθείας

Πρωτόκολλο TCP

- Transmission Control Protocol
- Χρήση επιβεβαιώσεων (acknowledgments) και επαναμεταδόσεις μη επιβεβαιωμένων μηνυμάτων
- Το πρόγραμμα “βλέπει” τα δεδομένα στη σωστή σειρά (δεν τοποθετούνται ανακατωμένα τα πακέτα)

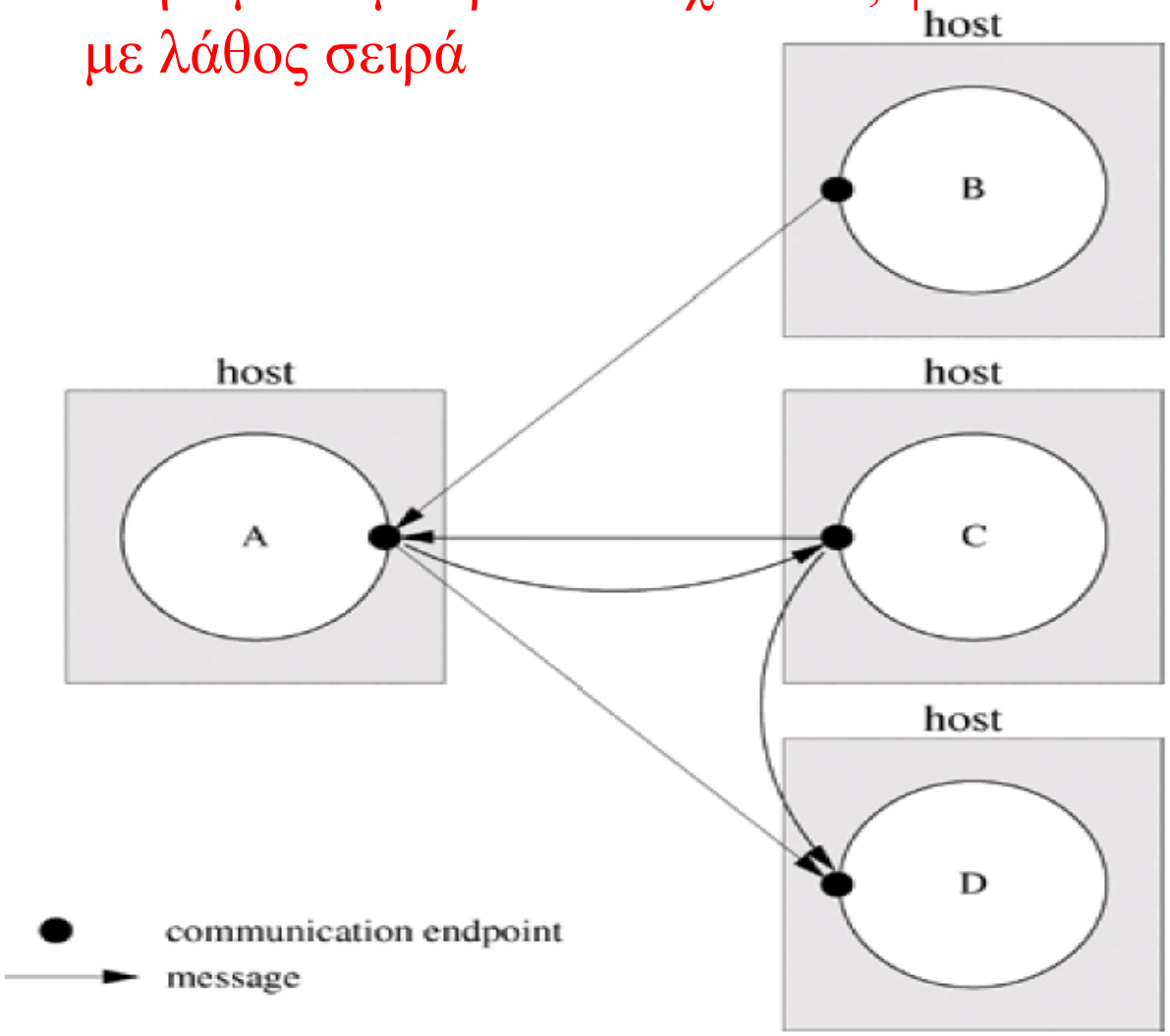


Πολλοί Πελάτες



Πρωτόκολλο UDP

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



Σύγκριση TCP-UDP

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

Σειριακός Εξυπηρετητής

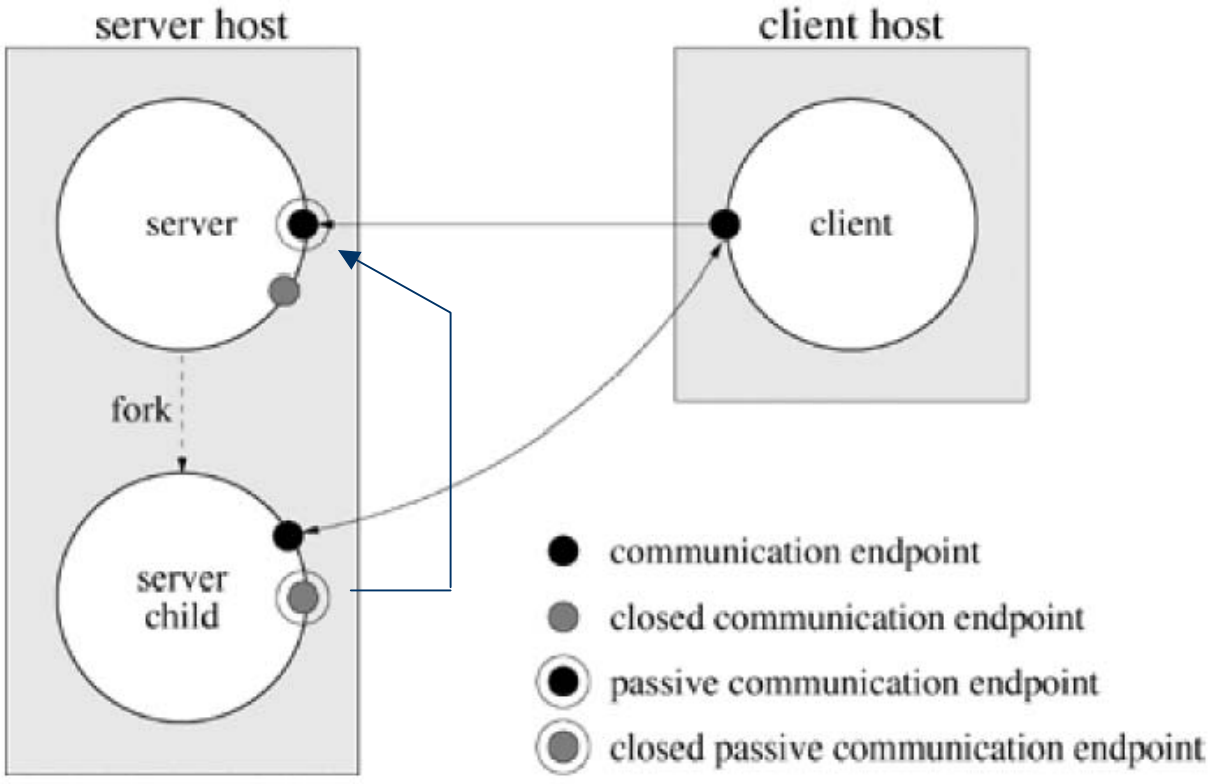
```
for ( ; ; ) {
    wait for a client request on the listening file descriptor
    create a private two-way communication channel to the client
    while (no error on the private communication channel)
        read from the client
        process the request
        respond to the client
    close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



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 file descriptor for the private communication channel
   clean up zombie children
}
```

ΠΑΝΤΑ

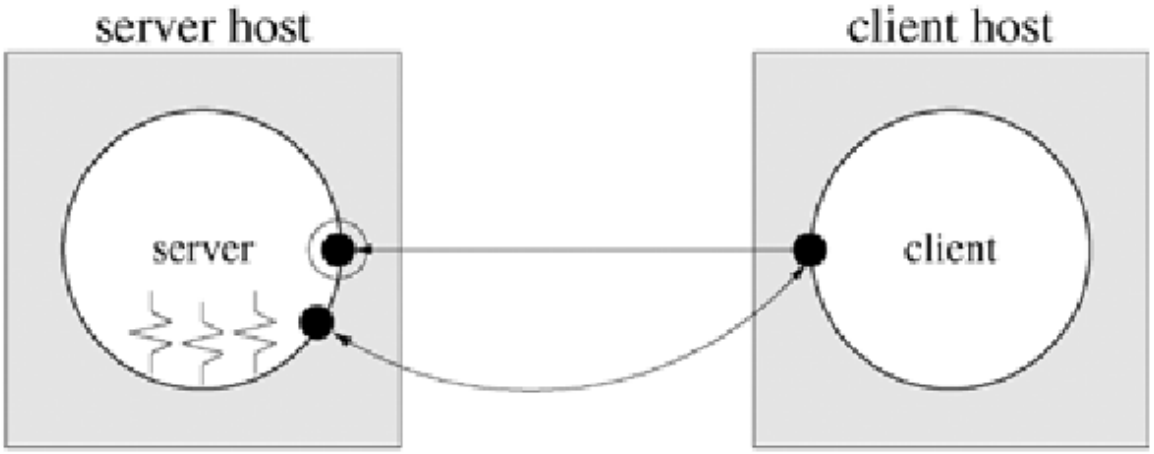
Λειτουργία Παιδιού

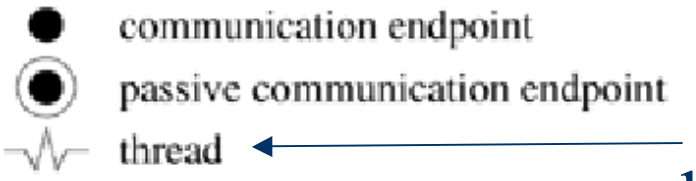
```
close the listening file descriptor
handle the client
close the communication for the private channel
exit
```

Ερωτήσεις

- Γιατί ο πατέρας εξυπηρετητής να κλείνει τον περιγραφητή?
  - Για να μη ξεμείνει από περιγραφητές
  - Για να μπορεί ο πελάτης να διαγνώσει EOF με τη read (όπως στους σωλήνες, αν όχι δεδομένα και η άλλη πλευρά κλείσει το άκρο)

Εναλλακτικό Μοντέλο





Το ίδιο γίνεται και με διεργασίες

Πολλές διεργασίες παιδιά

Καινούργιες συνδέσεις κατανέμονται σε αυτά.

ΜΕΙΟΝΕΚΤΗΜΑ

Συγχρονισμός παιδιών για διάβασμα δεδομένων πελατών (πχ από σωλήνα)

Χρήσιμες Γνώσεις…

- 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 → Πχ, SUN vs Intel αρχιτεκτονικές
  - 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 `PF_INET`), όπου στο πεδίο `h_name` της επιστρεφόμενης δομής βρίσκεται το όνομα του υπολογιστή
  - Απαίτηση: `#include <netdb.h>`

Χρήσιμες Γνώσεις (2)

- 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 `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");
```

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 `PF_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;
int sock;

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");
```

sockaddr vs sockaddr_in

```
struct sockaddr {
  unsigned short sa_family; // address family, AF_xxx
  char sa_data[14]; // 14 bytes of protocol address
};
```

```
struct sockaddr_in {
  short int sin_family; // Address family
  unsigned short int sin_port; // Port number
  struct in_addr sin_addr; // Internet address
  unsigned char sin_zero[8];
};
```

2+4+8=14

// Same size as struct sockaddr

```
struct in_addr {
  unsigned long s_addr; // that's a 32-bit long, or 4 bytes
};
```

Συναρτήσεις connect, listen, accept

- 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`

≈ 5

- 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`"

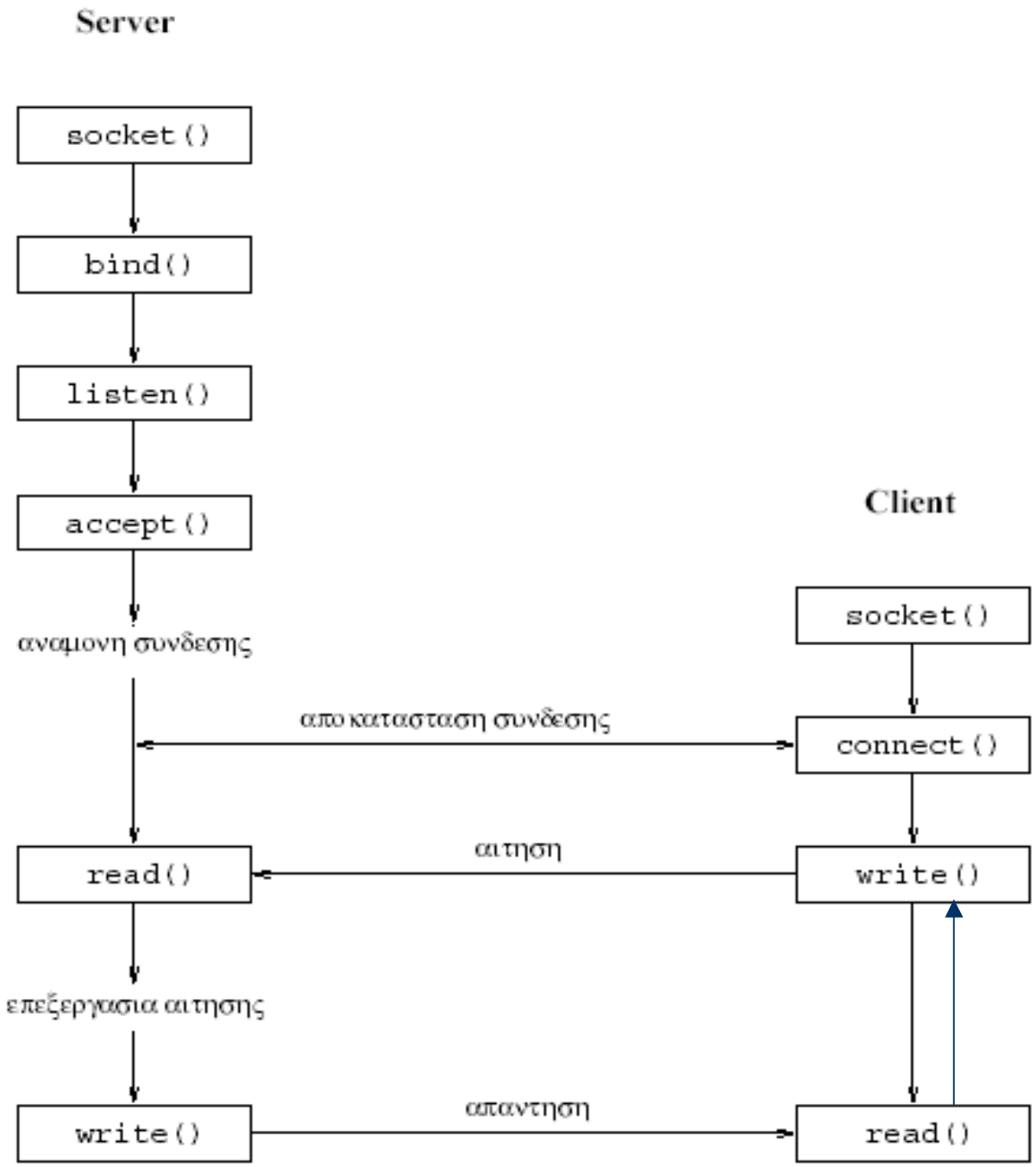
Πρακτικές λύσεις

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

Να βάζετε στην αρχή χειριστή του σήματος SIGPIPE

Για γρήγορη επαναχρησιμοποίηση των ports ενός socket, κάντε χρήση της συνάρτησης setsockopt (παράδειγμα 18.6 στο ebook)

TCP Επικοινωνία



Παράδειγμα int_str_server.c

```
/* 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 */
#include <stdlib.h>                                           /* For exit */
#include <string.h>                                /* For strlen, bzero */

void reverse(char *);

main(int argc, char *argv[])     /* Server with Internet stream sockets */
{  int port, sock, newsock; char buf[256];
   unsigned int serverlen, clientlen;
   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(PF_INET, SOCK_STREAM, 0)) < 0) { /* Create socket */
      perror("socket"); exit(1); }
   port = atoi(argv[1]);              /* Convert port number to integer */
   server.sin_family = PF_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 address */
      perror("bind"); exit(1); }
   if (listen(sock, 5) < 0) {                 /* Listen for connections */
      perror("listen"); exit(1); }
   printf("Listening for connections to port %d\n", port);
```

Παράδειγμα int_str_server.c (2)

```
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 and port %d\n",
        rem -> h_name, client.sin_port);
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) { /* Get 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 */
#include <stdlib.h> /* For exit */
#include <string.h> /* For strlen, bzero, bcopy */
main(int argc, char *argv[]) /* Client with Internet stream sockets */
{ int port, sock; char buf[256]; unsigned int serverlen;
struct sockaddr_in server, server2;
struct sockaddr *serverptr;
struct hostent *rem;
if (argc < 3) { /* Are server's host name and port number given? */
printf("Please give host name and port number\n"); exit(1); }
if ((sock = socket(PF_INET, SOCK_STREAM, 0)) < 0) { /* Create socket */
perror("socket"); exit(1); }
if ((rem = gethostbyname(argv[1])) == NULL) { /* Find server address */
perror("gethostbyname"); exit(1); }
port = atoi(argv[2]); /* Convert port number to integer */
server.sin_family = PF_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); }
if (getsockname(sock, (struct sockaddr *)&server2, &serverlen) < 0) {
  perror("Getsockname"); exit(-1); }
printf("Requested connection to host %s port %d. My port = %d\n",
       argv[1], port, server2.sin_port);
do {
bzero(buf, sizeof buf); /* Initialize buffer */
printf("Give input string: ");
fgets(buf, sizeof buf, stdin); /* Read message from stdin */
buf[strlen(buf)-1] = '\0'; /* Remove newline character */
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); exit(0); } /* Close socket and exit */
```

Αποτέλεσμα Εκτέλεσης

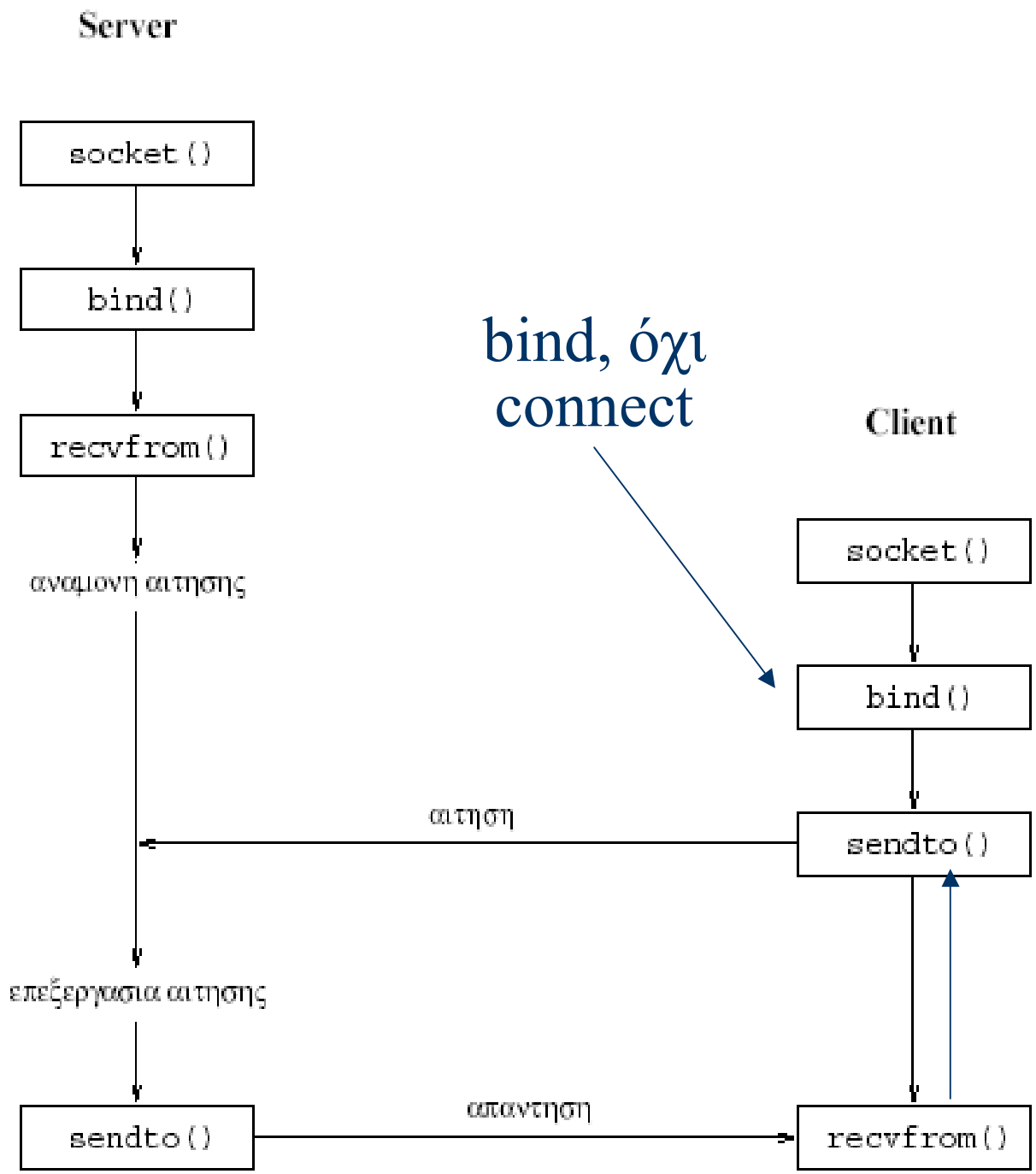
```
$ ./int_str_server 30000
Listening for connections to port 30000
Accepted connection from knossos.di.uoa.gr
Read string: test
Read string: A string
Read string: niconanomimatamimonanocin
Read string: This is a test line
Read string: Teleiosame
Read string: end
^C
$
```

```
$ ./int_str_client galini.di.uoa.gr 30000
Requested connection to host galini.di.uoa.gr 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
$
```

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 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`)

UDP Επικοινωνία



Παράδειγμα int_dgr_server.c

```
/* 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 */
#include <stdlib.h>                                          /* For exit */
#include <string.h>                                         /* For bzero */
main(int argc, char *argv[])   /* Server with Internet datagram sockets */
{  int n, port, sock; char buf[256]; unsigned int serverlen, clientlen;
   struct sockaddr_in server, client;
   struct sockaddr *serverptr, *clientptr; struct hostent *rem;
   if ((sock = socket(PF_INET, SOCK_DGRAM, 0)) < 0) {  /* Create socket */
      perror("socket"); exit(1); }
   server.sin_family = PF_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 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 */
```

Αυτόματη επιλογή

Βρες πραγματικό port

Παράδειγμα int_dgr_client.c

```
/* 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 */
#include <stdlib.h>                                          /* For exit */
#include <string.h>                          /* For strlen, bzero, bcopy */
main(int argc, char *argv[])   /* Client with Internet datagram sockets */
{  int port, sock; char buf[256];
   unsigned int serverlen, clientlen;
   struct sockaddr_in server, client;
   struct sockaddr *serverptr, *clientptr; struct hostent *rem;
   if (argc < 3) {     /* Are server's host name and port number given? */
      printf("Please give host name and port number\n"); exit(1); }
   if ((sock = socket(PF_INET, SOCK_DGRAM, 0)) < 0) {  /* Create socket */
      perror("socket"); exit(1); }
   if ((rem = gethostbyname(argv[1])) == NULL) { /* Find server address */
      perror("gethostbyname"); exit(1); }
   port = atoi(argv[2]);                /* Convert port number to integer */
   server.sin_family = PF_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 = PF_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 address */
      perror("bind"); exit(1); }
   while (fgets(buf, sizeof buf, stdin) != NULL) { /* Read continuously
                                                    messages from stdin */
      buf[strlen(buf)-1] = '\0';            /* Remove newline character */
      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); } }
```

Εκτέλεση Προγράμματος

```
$ ./int_dgr_server
Socket port: 32772
Received from galini.di.uoa.gr: prompt
Received from galini.di.uoa.gr: timeout=50
Received from galini.di.uoa.gr: default=linux
Received from galini.di.uoa.gr: boot=/dev/hda
Received from galini.di.uoa.gr: map=/boot/map
Received from galini.di.uoa.gr: install=/boot/boot.b
Received from galini.di.uoa.gr: message=/boot/message
Received from galini.di.uoa.gr: linear
Received from galini.di.uoa.gr:
Received from galini.di.uoa.gr: image=/boot/vmlinuz-2.4.7-10
Received from galini.di.uoa.gr:         label=linux
Received from galini.di.uoa.gr:         read-only
Received from galini.di.uoa.gr:         root=/dev/hda5
Received from galini.di.uoa.gr:
Received from galini.di.uoa.gr: other=/dev/hda1
Received from galini.di.uoa.gr:         optional
Received from galini.di.uoa.gr:         label=DOS
^C
$
```

```
$ ./int_dgr_client knossos.di.uoa.gr 32772 < /etc/lilo.conf
prompt
timeout=50
default=linux
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
message=/boot/message
linear

image=/boot/vmlinuz-2.4.7-10
        label=linux
        read-only
        root=/dev/hda5

other=/dev/hda1
        optional
        label=DOS
$
```

Παράδειγμα rls.c

```
/* rls.c - a client for a remote directory listing service
 *         usage: rls hostname directory
 */
#include        <stdio.h>
#include        <stdlib.h>
#include        <unistd.h>
#include        <strings.h>
#include        <string.h>
#include        <sys/types.h>
#include        <sys/socket.h>
#include        <netinet/in.h>
#include        <netdb.h>
#define         oops(msg)       { perror(msg); exit(1); }
#define          PORTNUM         15000
main(int ac, char *av[]) {
  struct sockaddr_in  servadd;  /* the number to call */
  struct hostent      *hp;      /* used to get number */
  int    sock_id, sock_fd;      /* the socket and fd  */
  char   buffer[BUFSIZ];        /* to receive message */
  int    n_read;                /* for message length */

  if ( ac != 3 ) exit(1);
  /** Step 1: Get a socket **/
  sock_id = socket( AF_INET, SOCK_STREAM, 0 ); /* get a line    */
        if ( sock_id == -1 ) oops( "socket" ); /* or fail       */
  /** Step 2: connect to server **/
  bzero( &servadd, sizeof(servadd) );     /* zero the address     */
  hp = gethostbyname( av[1] );            /* lookup host's ip #   */
  if (hp == NULL)
        oops(av[1]);                    /* or die               */
  bcopy(hp->h_addr, (struct sockaddr *)&servadd.sin_addr, hp->h_length);
  servadd.sin_port = htons(PORTNUM);      /* fill in port number  */
  servadd.sin_family = AF_INET ;          /* fill in socket type  */
  if ( connect(sock_id,(struct sockaddr *)&servadd, sizeof(servadd)) !=0)
       oops( "connect" );

  /** Step 3: send directory name, then read back results **/


  if ( write(sock_id, av[2], strlen(av[2])) == -1)
        oops("write");
  if ( write(sock_id, "\n", 1) == -1 )
        oops("write");

  while( (n_read = read(sock_id, buffer, BUFSIZ)) > 0 )
        if ( write(1, buffer, n_read) == -1 )
                oops("write");
  close( sock_id );
}
```

Παράδειγμα rlsd.c

```
/* 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  <strings.h>
#include  <string.h>
#define   PORTNUM  15000   /* our remote ls server port */
#define   HOSTLEN  256
#define   oops(msg)       { perror(msg) ; exit(1) ; }
int main(int ac, char *av[]) {
struct  sockaddr_in   saddr;   /* build our address here */
struct  hostent          *hp;   /* this is part of our    */
char    hostname[HOSTLEN];     /* address                */
int     sock_id,sock_fd;       /* line id, file desc     */
FILE    *sock_fpi, *sock_fpo;  /* streams for in and out */
FILE    *pipe_fp;              /* use popen to run ls    */
char    dirname[BUFSIZ];       /* from client            */
char    command[BUFSIZ];       /* for popen()            */
int     dirlen, c;

/** Step 1: ask kernel for a socket **/
sock_id = socket( PF_INET, SOCK_STREAM, 0 );    /* get a socket */
if ( sock_id == -1 )
        oops( "socket" );
/** Step 2: bind address to socket.  Address is host,port **/
bzero( (void *)&saddr, sizeof(saddr) ); /* clear out struct     */
gethostname( hostname, HOSTLEN );       /* where am I ?         */
hp = gethostbyname( hostname );         /* get info about host  */
bcopy( (void *)hp->h_addr, (void *)&saddr.sin_addr, hp->h_length);
saddr.sin_port = htons(PORTNUM);        /* fill in socket port  */
saddr.sin_family = AF_INET ;            /* fill in addr family  */
if ( bind(sock_id, (struct sockaddr *)&saddr, sizeof(saddr)) != 0 )
       oops( "bind" );
/** Step 3: allow incoming calls with Qsize=1 on socket **/
if ( listen(sock_id, 1) != 0 )
        oops( "listen" );
/* main loop: accept(), write(), close() */
```

Παράδειγμα rlsd.c (συν.)

```
while ( 1 ){
        sock_fd = accept(sock_id, NULL, NULL); /* wait for call */
        if ( sock_fd == -1 )
                oops("accept");
        /* open reading direction as buffered stream */
        if( (sock_fpi = fdopen(sock_fd,"r")) == NULL )
                oops("fdopen reading");
        if ( fgets(dirname, BUFSIZ-5, sock_fpi) == NULL )
                oops("reading dirname");
        sanitize(dirname);
        /* open writing direction as buffered stream */
        if ( (sock_fpo = fdopen(sock_fd,"w")) == NULL )
                oops("fdopen writing");
        sprintf(command,"ls %s", dirname);
        if ( (pipe_fp = popen(command, "r")) == NULL )
                oops("popen");
        /* transfer data from ls to socket */
        while( (c = getc(pipe_fp)) != EOF )
                putc(c, sock_fpo);
        pclose(pipe_fp);
        fclose(sock_fpo);
        fclose(sock_fpi);
}
}

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

Αντί για ανακατεύθυνση εντολής σε αρχείο, popen. Μπορώ να διαβάσω ή γράψω με popen. ΟΧΙ και τα 2 μαζί

Εκτέλεση Προγράμματος

```
linux06:/home/users/spro/sockets>./rls linux03.di.uoa.gr /home/users/spro
1.BasicCommands.ppt
2.BashScripts.ppt
3.UnixFileCommands.ppt
4.ForkAndPipes.ppt
Code.zip
WINDOWS
askisi1
askisi1.tar
askisi2.pdf
co
code
createUsers
curses.tar
cursesCode
dead.letter
example
fullOfErrors
fullOfErrors.c
hello1
hello1.c
hidden
koko.txt
kokos.tar
list
mail
mine
mine.c
mine.cc
myfork
myfork.c
mykokoDir
mytest
popo
popo.txt
programs.tar
project1
project2
public_html
scripts
sockets
testlimits
testlimits.c
unixFiles
userlist
userlist.old
userlist.pr1
linux06:/home/users/spro/sockets>
```

Παλιό Θέμα (1)

Μελετήστε το τμήμα προγράμματος C στο διπλανό σχήμα και εξηγήστε πολύ συνοπτικά τη λειτουργία του. Αν το εκτελέσιμο πρόγραμμα που προκύπτει απ' αυτό ονομάζεται "hide", συμπληρώστε τα αποτελέσματα των εντολών που φαίνονται στη συνέχεια. Υποθέστε ότι όλες οι κλήσεις συστήματος θα είναι επιτυχείς.

```
$ wc -c hide.c
616 hide.c
$ cat inp.txt
abcdefghijklmno
ABCDEFGHIJKLMNO
123456789012345
678901234567890
$ ./hide 1 < inp.txt
.............................
$ ./hide 3 < inp.txt
.............................
$ ./hide 8 < hide.c | wc -c
.............................
```

```
.....................................
main(int argc, char *argv[])
{  int i, n, status, fd[2];
   int flag = 1; char ch;
   n = atoi(argv[1]);
   for (i=0 ; i<n ; i++) {
      if (i != n-1) pipe(fd);
      if (fork() == 0) {
         if (i != n-1) {
            close(fd[0]);
            dup2(fd[1], 1);
            close(fd[1]); }
         while (read(0, &ch, 1) > 0) {
            if (flag < 0) write(1, &ch, 1);
            flag = -flag; }
         exit(0); }
      else {
         if (i != n-1) {
            dup2(fd[0], 0);
            close(fd[0]);
            close(fd[1]); } } }
.....................................
```

Παλιό Θέμα (2)

Δύο διεργασίες P_1 και P_2 επικοινωνούν μέσω υποδοχών ροής (stream sockets). Η διεργασία P_1 κάνει τρεις κλήσεις εγγραφής στην υποδοχή της S_1, σε απροσδιόριστες φάσεις της εκτέλεσής της, και γράφει, κατά σειρά, τα μηνύματα M_1, M_2 και M_3. Η διεργασία P_2 κάνει τρεις κλήσεις ανάγνωσης από την υποδοχή της S_2, τις R_1, R_2 και R_3, με τη σειρά αυτή, πάλι σε απρόβλεπτες χρονικές στιγμές. Ποια από τα παρακάτω ενδεχόμενα είναι πιθανό να συμβούν και ποια αποκλείονται παντελώς;

i. Η R_1 θα διαβάσει το μήνυμα M_1, η R_2 το M_2 και η R_3 το M_3.

ii. Η R_1 θα διαβάσει το μήνυμα M_3, η R_2 το M_1 και η R_3 το M_2.

iii. Η R_1 θα διαβάσει το μήνυμα M_1M_2 και η R_2 το M_3.

iv. Η R_2 θα διαβάσει το μήνυμα M_1M_3 και η R_3 το M_2.

v. Η R_1 θα διαβάσει το μήνυμα $M_1M_2M_3$.

vi. Η R_1 θα διαβάσει το μήνυμα M_1 και η R_2 το M_2M_3.

Δικαιολογήστε την απάντησή σας. Υπάρχουν άλλα πιθανά ενδεχόμενα, εκτός από αυτά που αναφέρονται και επιλέξατε; Ποιες θα ήταν οι απαντήσεις σας αν η επικοινωνία μεταξύ των διεργασιών γινόταν μέσω τηλεγραφικών υποδοχών (datagram sockets); Αν γινόταν μέσω σωλήνων;

Παλιό Θέμα (3)

Ποια ήταν, κατά τη γνώμη σας, η πρόθεση του προγραμματιστή που έγραψε το πρόγραμμα C στο Σχήμα 2; Δώστε μία πιθανή εκτύπωση του αντίστοιχου εκτελέσιμου προγράμματος (αν όλες οι κλήσεις συστήματος λειτουργήσουν χωρίς πρόβλημα). Υπάρχει κάποιο πολύ σοβαρό σχεδιαστικό λάθος στο πρόγραμμα αυτό; Αν ναι, εξηγήστε ποιο είναι και προτείνετε πολύ σύντομα τουλάχιστον δύο τρόπους με τους οποίους θα μπορούσε να διορθωθεί το πρόβλημα αυτό.

```
#include <stdio.h>
#include <signal.h>
int p[4] = {0, 0, 0, 0};
void handler(int sig)
{  printf("%d: %d %d %d %d\n",
          getpid(), p[0], p[1],
          p[2], p[3]); }
main()
{  int i, pid, st;
   signal(SIGUSR1, handler);
   for (i=0 ; i<4 ; i++) {
      pid = fork();
      if (!pid) { pause(); exit(0); }
      p[i] = pid; }
   for (i=0 ; i<4 ; i++)
      kill(p[i], SIGUSR1);
   kill(getpid(), SIGUSR1);
   for (i=0 ; i<4 ; i++)
      wait(&st); }
```

include <stdlib.h>