Δίκτυα Επικοινωνιών II: Network Programming
UDP Sockets, Signals

Δρ. Απόστολος Γκάμας

Διδάσκων 407/80

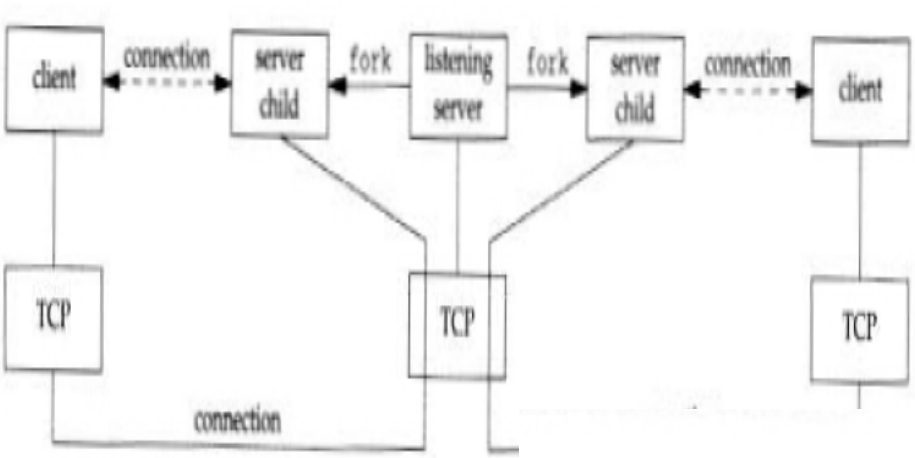
gkamas@uop.gr

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

UDP vs TCP

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

UDP vs TCP (client/server)

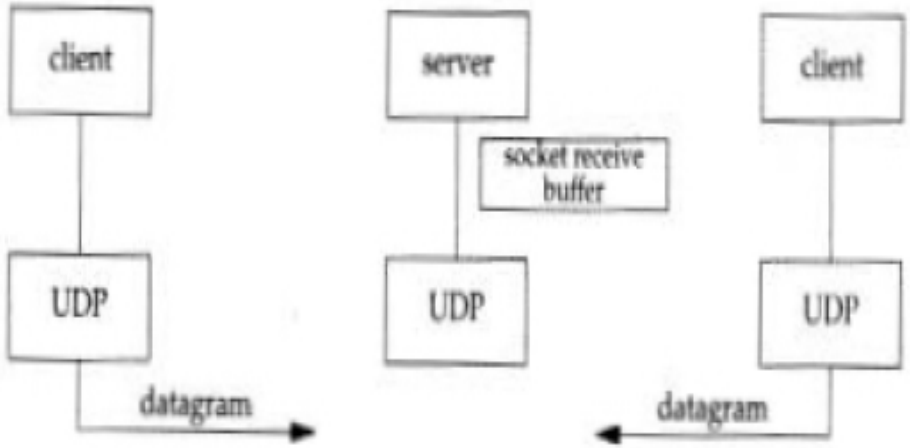


TCP

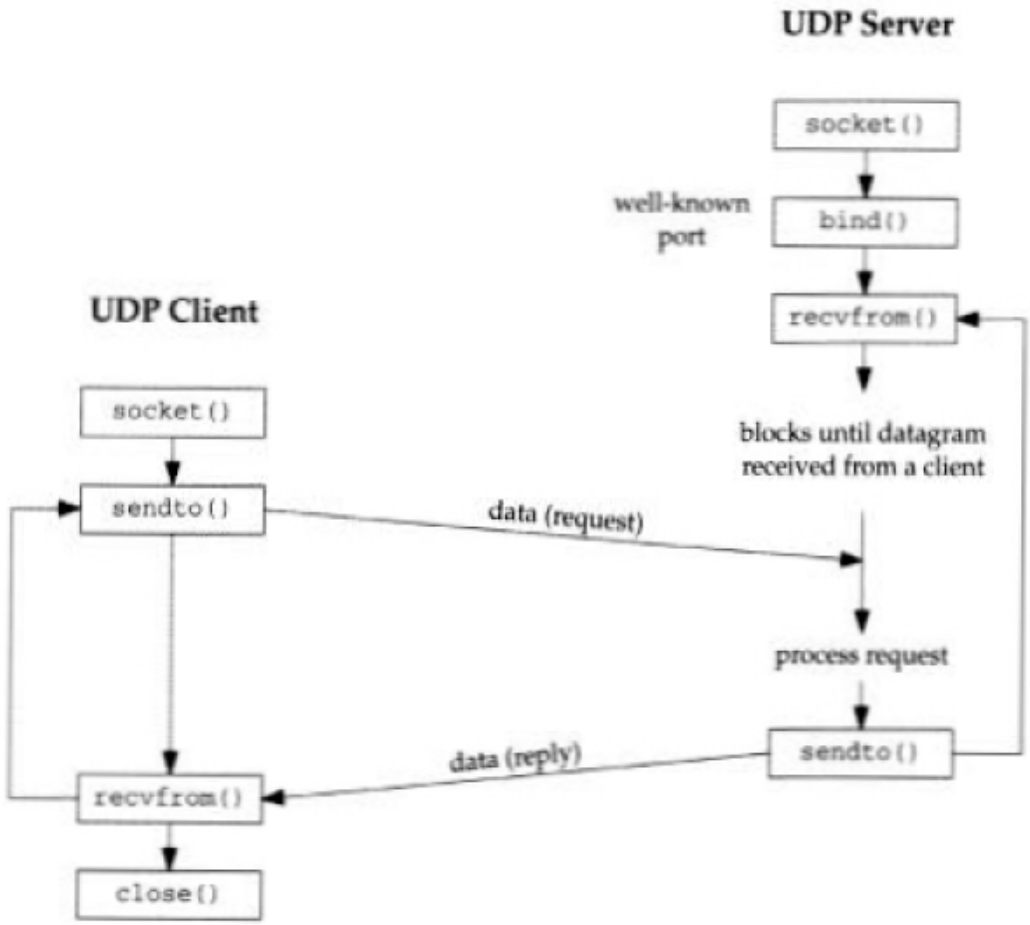
Ο client επικοινωνεί με έναν «αφοσιωμένο» αντίγραφο του server, ενώ ο «αρχικός» server μπορεί να δέχεται νέες κλήσεις σύνδεσης

UDP

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



UDP Client/Server interaction (Datagram Communication)



UDP Sockets - Connectionless

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

Πρωτογενείς κλήσεις για τα sockets

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

Η κλήση “recvfrom”

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

Η κλήση “sendto”

- Σύνταξη:
  ssize_t sendto (int sockfd, void *buff, size_t nbytes, int flags, const struct sockaddr *to, socklen_t *addrlen);

- Στέλνει τα δεδομένα (datagram) που βρίσκονται στη παράμετρο void *buff με μέγεθος nbytes στον προορισμό (δηλώνεται μέσω της δομής const struct sockaddr *to)

- Δεν υπάρχει καμία σύνδεση (connection) με τον παραλήπτη

Ένας απλός UDP Server

udpcliserv/udpserv01.c

```
1 #include    "unp.h"

2 int
3 main(int argc, char **argv)
4 {
5     int     sockfd;
6     struct sockaddr_in servaddr, cliaddr;

7     sockfd = Socket(AF_INET, SOCK_DGRAM, 0);

8     bzero(&servaddr, sizeof(servaddr));
9     servaddr.sin_family = AF_INET;
10    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
11    servaddr.sin_port = htons(SERV_PORT);

12    Bind(sockfd, (SA *) &servaddr, sizeof(servaddr));

13    dg_echo(sockfd, (SA *) &cliaddr, sizeof(cliaddr));
14 }
```

udpcliserv/udpserv01.c

… Ένας απλός UDP Server

lib/dg_echo.c

```
1 #include    "unp.h"

2 void
3 dg_echo(int sockfd, SA *pcliaddr, socklen_t clilen)
4 {
5     int     n;
6     socklen_t len;
7     char    mesg[MAXLINE];

8     for ( ; ; ) {
9         len = clilen;
10        n = Recvfrom(sockfd, mesg, MAXLINE, 0, pcliaddr, &len);

11        Sendto(sockfd, mesg, n, 0, pcliaddr, len);
12    }
13 }
```

lib/dg_echo.c

Ένας απλός UDP Client

udpcliserv/udpcli01.c

```
1 #include    "unp.h"

2 int
3 main(int argc, char **argv)
4 {
5     int     sockfd;
6     struct sockaddr_in servaddr;

7     if (argc != 2)
8         err_quit("usage: udpcli <IPaddress>");

9     bzero(&servaddr, sizeof(servaddr));
10    servaddr.sin_family = AF_INET;
11    servaddr.sin_port = htons(SERV_PORT);
12    Inet_pton(AF_INET, argv[1], &servaddr.sin_addr);

13    sockfd = Socket(AF_INET, SOCK_DGRAM, 0);

14    dg_cli(stdin, sockfd, (SA *) &servaddr, sizeof(servaddr));

15    exit(0);
16 }
```

udpcliserv/udpcli01.c

… Ένας απλός UDP Client

lib/dg_cli.c

```
1 #include    "unp.h"

2 void
3 dg_cli(FILE *fp, int sockfd, const SA *pservaddr, socklen_t servlen)
4 {
5     int     n;
6     char    sendline[MAXLINE], recvline[MAXLINE + 1];

7     while (Fgets(sendline, MAXLINE, fp) != NULL) {

8         Sendto(sockfd, sendline, strlen(sendline), 0, pservaddr, servlen);

9         n = Recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);

10        recvline[n] = 0;        /* null terminate */
11        Fputs(recvline, stdout);
12    }
13 }
```

lib/dg_cli.c

Signals

- Μέσω των signals μία διεργασία μπορεί να ειδοποιηθεί για διάφορα γεγονότα
- Τα signals ονομάζονται και software interrupts
- Signals μπορούν να σταλούν από μία διεργασία σε μία άλλη διεργασία ή από τον kernel σε μία διεργασία
- Signals μπορούν να σταλούν με την εντολή kill:
  - Απο την γραμμή εντολών: kill –sigid –pid
  - Σε ένα C πρόγραμμα: int kill(pid_t pid, int sigid);
- Κάθε signal έχει εξ'ορισμού του μία (προ)καθορισμένη συμπεριφορά

Signal ids και προκαθορισμένη συμπεριφορά

Signal	id	default
SIGKILL	9	Terminate
SIGALRM	14	Terminate
SIGSYS	12	Core
SIGCHLD	20	Ignore

Όταν μια διεργασία-παιδί τερματίζεται στέλνει ένα σήμα SIGCHLD (id=20) στον πατέρα της

Το σήμα αυτό αγνοείται (προκαθορισμένη συπεριφορα: ignore)

Καθορισμός συμπεριφοράς ενός Signal

- Πρωτογενείς κλήσεις: signal() και sigaction()
- Στο πρόγραμμα μας μπορούμε να καθορίσουμε επ'ακριβώς την συμπεριφορά μίας διεργασίας, ύστερα από την λήψη ενός signal:
  - Εκτέλεση μίας συνάρτησης (signal handler)
  - Καμία ενέργεια (SIG_IGN)
  - Προκαθορισμένη συμπεριφορά (SIG_DFL)
- Τα σήματα SIGKILL και SIGSTOP δεν μπορούν ούτε να «αγνοηθούν» (SIG_IGN) ούτε να «πιαστούν» από μία συνάρτηση (signal handler) -> εκτελείται πάντοτε η default ενέργεια

Οι πρωτογενείς κλήσεις για τα signals

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

Η κλήση “signal”

- Σύνταξη:
  signal (int sig, void (*disp)(int));

- Καθορίζει την συμπεριφορά της διεργασίας, μέσα στην οποία καλείται, ύστερα από την λήψη ενός signal sig
- Το όρισμα *disp ορίζει αυτή την συμπεριφορά
- Το *disp μπορεί να είναι η διεύθυνση του signal handler (pointer σε συνάρτηση), SIG_IGN ή SIG_DFL

Zombie processes

- Για τον σωστό τερματισμό μιας διεργασίας-παιδί πρέπει ο πατέρας να «ειδοποιηθεί» (η «ειδοποίηση» γίνεται όταν ο πατέρας εκτελέσει την πρωτογενή κλήση wait ή waitpid)
- Μία διεργασία-παιδί η οποία έχει τερματιστεί και της οποίας η διεργασία-πατέρας δεν έχει λάβει ακόμα ειδοποίηση του τερματισμού της ονομάζεται zombie (defunct) διεργασία
- Μία zombie διεργασία υπάρχει μόνο σαν μία καταχώρηση στο process table, όπου διατηρούνται πληροφορίες σχετικά με αυτήν
- Μία zombie διεργασία μπορούμε να την σκοτώσουμε από την γραμμή εντολών με την εντολή kill

Handling zombie processes

- Όταν σε μία διεργασία-πατέρας κάνουμε **fork** (γεννάμε διεργασίες-παιδιά), θα πρέπει να καλέσουμε για τις διεργασίες-παιδιά την **wait** ή την **waitpid**, έτσι ώστε να τις εμποδίσουμε από το να γίνουν **zombie processes**
- Για το σκοπό αυτό στην διεργασία-πατέρα εγκαθιστούμε έναν **signal handler** στο **signal SIGCHLD**, και μέσα σε αυτόν τον **signal handler** καλούμε την **wait** ή την **waitpid**

Οι πρωτογενείς κλήσεις για wait, waitpid

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

Η κλήση “wait”

- Σύνταξη:
  pid_t wait(int *statloc);

- Επιστρέφει το process id του τερματιζόμενου παιδιού
- Στον δείκτη statloc επιστρέφεται το termination status του παιδιού
- Η wait είναι blocking συνάρτηση

Η κλήση “waitpid”

- Σύνταξη:
  pid_t waitpid(pid_t pid, int *statloc, int options);

- Επιστρέφει το process id του τερματιζόμενου παιδιού
- Δίνεται η δυνατότητα καθορισμού, μέσω του pid, του παιδιού για το οποίο θα εκτελεστεί η συνάρτηση αυτή
- Στον δείκτη statloc επιστρέφεται το termination status του παιδιού
- Δίνεται η δυνατότητα καθορισμού επιπλέον επιλογών. Η πιο συνηθισμένη επιλογή είναι η WNOHANG, η οποία εμποδίζει την waitpid να γίνει blocked αν δεν υπάρχουν τερματιζόμενα παιδιά

wait vs waitpid

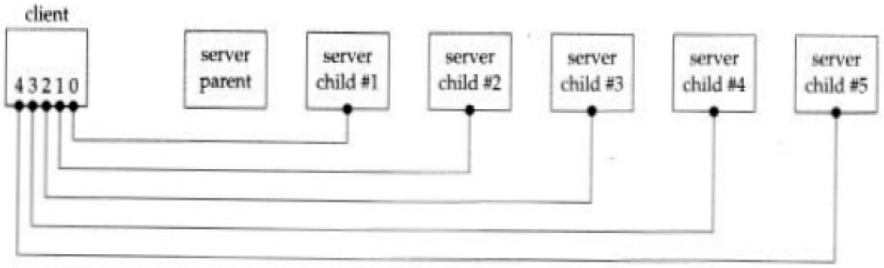
- Υποθέτουμε ότι έχουμε καθορίσει σαν signal handler για το signal SIGCHLD την συνάρτηση sig_chld (e.g. signal(SIGCHLD, sig_chld);)

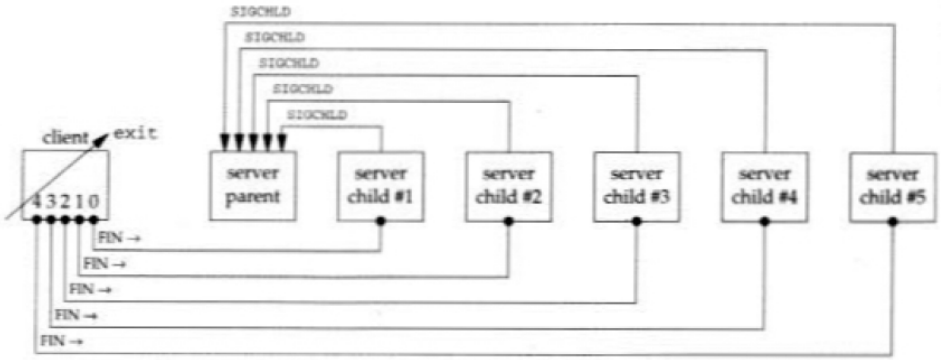
tcpcliserv/sigchldwait.c

```
 1 #include    "unp.h"
 2 void
 3 sig_chld(int signo)
 4 {
 5     pid_t   pid;
 6     int     stat;
 7     pid = wait(&stat);
 8     printf("child %d terminated\n", pid);
 9     return;
10 }
```

tcpcliserv/sigchldwait.c

wait vs waitpid





- Τα **signals** δεν αποθηκεύονται σε ουρά
- Με την **wait** υπαρχει πρόβλημα όταν δημιουργηθούν ταυτόχρονα σήματα: ο **signal handler** θα εκτελεστεί μια φορά και τα υπόλοιπα **signals** θα χαθούν

wait vs waitpid

- Υποθέτουμε ότι έχουμε καθορίσει σαν signal handler για το signal SIGCHLD την συνάρτηση sig_chld
  (e.g. signal(SIGCHLD, sig_chld);)
  waitpid with WHOHANG option: non blocking call

tcpcliserv/sigchldwaitpid.c

```
1 #include    "unp.h"
2 void
3 sig_chld(int signo)
4 {
5     pid_t   pid;
6     int     stat;
7     while ( (pid = waitpid(-1, &stat, WNOHANG)) > 0)
8         printf("child %d terminated\n", pid);
9     return;
10 }
```

tcpcliserv/sigchldwaitpid.c

Handling interrupted system calls

- Όταν ένα **blocking system call** (π.χ. **accept**) διακοπεί από ένα **signal**, τότε μετά την επιστροφή από τον **signal handler** το **system call** θέτει την μεταβλητή **errno** ίσο με **EINTR**
- Το **system call** δεν είναι σίγουρο ότι θα επανακκινηθεί αυτόματα από τον **kernel**
- Για το λόγο αυτό θα πρέπει ο προγραμματιστής να φροντίζει για την επανεκκίνηση του **system call**

Handling interrupted system calls

```
for ( ; ; ) {
    clilen = sizeof(cliaddr);
    if ( (connfd = accept(listenfd, (SA *) &cliaddr, &clilen)) < 0) {
        if (errno == EINTR)
            continue;       /* back to for() */
        else
            err_sys("accept error");
    }
```