Εργαστήριο 11^o : Προγραμματισμός Συστήματος: Υποδοχές

Τσαδήμας Ανάργυρος, Τμήμα Πληροφορικής & Τηλεματικής

Ο βασικός μηχανισμός για την επικοινωνία μεταξύ μηχανημάτων (δικτύωση) ονομάζεται «υποδοχές» (sockets). Χρησιμοποιούν 8 βασικές κλήσεις συστήματος, πέντε από τις οποίες αφορούν αποκλειστικά τις υποδοχές: socket, bind, listen, accept, connect, read, write και close. Οι δομές διευθυνσιοδότησης είναι πολύπλοκες. Θα δούμε κάποιες επιπλέον συναρτήσεις για μετατροπές και τη δομή hostent η οποία κρατάει πληροφορίες δικτύου και επιστρέφεται από την gethostbyname.

sockets

Οι υποδοχές (sockets) παρέχουν point-to-point two way communication μεταξύ δυο διεργασιών. Αποτελούν έναν από τους βασικούς τρόπους επικοινωνίας μεταξύ διεργασιών, οι οποίες μπορεί να βρίσκονται στο ίδιο μηχάνημα ή ακόμη και σε διαφορετικά. Μια υποδοχή αποτελεί το άκρο της επικοινωνίας και το οποίο μπορεί να έχει όνομα. Υπάρχουν διαφορετικοί τύποι υποδοχών.

Οι υποδοχές υπάρχουν μέσα σε «πεδία επικοινωνίας» τα λεγόμενα communication domains. Ένα socket domain καθορίζει το περιβάλλον της επικοινωνίας, το οποίο παρέχει μια δομή διευθυνσιοδότησης (addressing structure) και ένα σύνολο από πρωτόκολλα.

Τα πιο συνηθισμένα communication domains είναι το UNIX domain και το Internet domain [1].

- UNIX domain: παρέχει διευθυνσιοδότηση υποδοχέων μέσα σε ένα σύστημα. Τα unix domain sockets ονομάζονται με Unix paths.
- internet domain: καθορίζεται από τη IP διεύθυνση του μηχανήματος που το φιλοξενεί και τον αριθμό πόρτας (συνήθως ένας αριθμός πάνω από 2000 μέχρι 2^{16})

Υπάρχουν 4 ειδών sockets types, από τις οποίες οι 2 είναι οι πιο συνηθισμένες [2]:

- stream socket (SOCK_STREAM): παρέχει διπλή, σειριακή, αξιόπιστη ροή δεδομένων. Χρησιμοποιείται το πρωτόκολλο TCP.
- datagram socket (SOCK_DGRAM): παρέχει διπλή επικοινωνία μηνυμάτων. Τα λαμβανόμενα μηνύματα μπορεί να μη ληφθούν με τη σειρά με την οποία εστάλησαν. Χρησιμοποιείται το πρωτόκολλο UDP.

[1] Τα πιο γνωστά communication domains είναι: AF_INET (IPv4 Internet domain) , AF_INET6 (IPv6 Internet domain), AF_UNIX (UNIX domain), AF_UNSPEC (unspecified)

[2] Επίσης υπάρχουν και τα SOCK_RAW (datagram interface to IP) και SOCK_SEQPACKET (fixed-length, sequenced, reliable, connection-oriented messages)

```c
#include <sys/un.h>

struct sockaddr {
sa_family_t  sa_family; /* AF_UNIX*/
char  sun_path[14]; /* path to socket */
};
```

Δημιουργία και ονοματολογία υποδοχών[3]

socket

```c
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

Δημιουργεί ένα socket στο συγκεκριμένο domain, τύπου type και του καθορισμένου πρωτοκόλλου. Αν το πρωτόκολλο δεν δηλωθεί, χρησιμοποιείται το καθορισμένο πρωτόκολλο για το συγκεκριμένο socket type (βάζουμε το 0). Επιστρέφεται ο περιγραφέας της υποδοχής (ένας ακέραιος αριθμός).

Μια άλλη διεργασία δεν μπορεί να χρησιμοποιήσει την υποδοχή αν δεν δηλωθεί μια διεύθυνση. Στο UNIX domain η σύνδεση γίνεται μέσω ενός path name. Στο internet domain η σύνδεση γίνεται μέσω της διεύθυνσης του μηχανήματος και του αριθμού της πόρτας.

Διευθύνσεις υποδοχών

Μια διεύθυνση καθορίζει ένα άκρο υποδοχής σε ένα συγκεκριμένο communication domain. Η μορφή της διεύθυνσης είναι συγκεκριμένη για κάθε domain. Έτσι, διευθύνσεις με διαφορετική μορφή, για να μπορούν να περάσουν στις συναρτήσεις των υποδοχών, «μετατρέπονται» (type cast) σε μια γενική δομή διεύθυνσης sockaddr.

Οι διευθύνσεις δικτύου καθορίζονται στο ‹netinet/in.h›. Στο IPv4 Internet domain (AF_INET), μια διεύθυνση υποδοχής καθορίζεται από μια δομή sockaddr_in:

[3]Στο Solaris όταν κάνουμε compile ένα πρόγραμμα το οποίο δημιουργεί sockets πρέπει να βάλουμε τις εξής παραμέτρους στο gcc -lsocket -lnsl

```c
#include <netin/in.h>

struct in_addr {
in_addr_t s_addr; /* IPv4 address */
};

struct sockaddr_in {
sa_family_t sin_family; /* address family */
in_port_t sin_port; /* port number */
struct in_addr sin_addr; /* IPv4 address */
};
```

bind

```c
#include <sys/types.h>
#include <sys/socket.h>
int bind(int sockfd, const struct sockaddr *addr, ↩
    socklen_t addrlen)
```

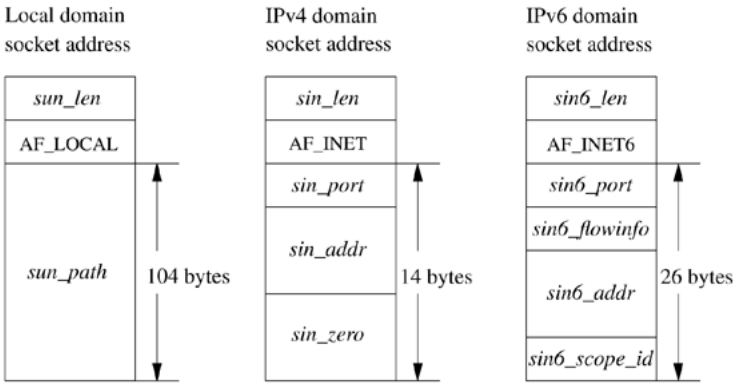
Για τη χρήση των κλήσεων των sockets πρέπει να χρησιμοποιήσουμε το ‹sys/socket.h›. Αν έχουμε να κάνουμε με UNIX domain sockets χρειάζεται το ‹sys/un.h›, ενώ στην περίπτωση του internet domain απαιτείται η ‹netinet/in.h›.

Η bind συνδέει ένα path ή μια διεύθυνση internet με ένα socket. Το πρώτο ορισμα (int sockfd) αναφέρεται στον περιγραφέα της υποδοχής και είναι αυτό που μας επιστρέφει η socket(). Το δεύτερο όρισμα είναι μια δομή (struct) η οποία κρατάει τις πληροφορίες διευθυνσιοδότησης.

Το τρίτο όρισμα αναφέρεται στο μέγεθος (σε bytes) της δομής διεθυνσιοδότησης η οποία υποδυκνύεται από τη διεύθυνση μνήμης addr. Σε επιτυχία η bind επιστρέφει 0 και σε αποτυχία -1.

Συνδέοντας stream sockets

Κατα την επικοινωνία δυο διεργασιών μέσω μιας υποδοχής, η μια διεργασία εκτελεί τον πάροχο υποδοχής και η άλλη τον πελάτη υποδοχής (ακολουθώντας το μοντέλο client-server). Ο server «δεσμεύει» (binds) την υποδοχή στο συμφωνημένο path ή διεύθυνση δικτύου. Στη συνέχεια μπλοκάρει την υποδοχή. Για μια SOCK_STREAM υποδοχή ο server καλεί τη listen() στην οποία καθο-



Σχήμα 1: Δομές διευθυνσιοδότησης για sockets πηγή

ρίζονται πόσες αιτήσεις σύνδεσης θα αποθηκεύονται στην ουρά.

listen

```
#include <sys/types.h>
#include <sys/socket.h>
 int listen(int sockfd, int backlog);
```

Η listen παίρνει 2 παραμέτρους: τον περιγραφέα υποδοχής (αυτόν που επιστρέφει η socket()) και το μέγεθος της ουράς (το μέγεθος των αιτήσεων που θα περιμένουν ενόσω ο server χειρίζεται μια συγκεκριμένη σύνδεση). Σε επιτυχία η listen επιστρέφει 0 και σε αποτυχία -1.

Ο client ξεκινά μια σύνδεση με την υποδοχή του server καλώντας την connect().

connect

```
#include <sys/types.h>
#include <sys/socket.h>
int connect(int sockfd, const struct sockaddr *↩
    addr, socklen_t addrlen)
```

Στην περίπτωση του UNIX domain η connect είναι της μορφής:

```
struct sockaddr_un server;
...
connect (sd, (struct sockaddr *)&server, length);
```

ενώ στην περίπτωση του internet domain θα είναι:

```
struct sockaddr_in server;
...
connect (sd, (struct sockaddr *)&server, length);
```

Το πρώτο όρισμα και εδώ είναι ο περιγραφέας υποδοχής. Το δεύτερο όρισμα είναι η δομή η οποία κρατά πληροφορίες για τη διεύθυνση της υποδοχής. Το τρίτο όρισμα αναφέρεται στο μέγεθος του δευτέρου ορίσματος.

Σε επιτυχία η connect επιστρέφει 0 και σε αποτυχία -1.

Στην περίπτωση του SOCK_STREAM ο server καλεί την accept() ώστε να ολοκληρωθεί η σύνδεση.

accept

```
#include <sys/types.h>
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *addr, ↩
    socklen_t *addrlen);
```

Η accept επιστρέφει έναν νέο περιγραφέα υποδοχής, ο οποίος είναι ενεργός μόνο για τη συγκεκριμένη σύνδεση (σχήμα 3). Ένας server μπορεί να έχει πολλαπλές SOCK_STREAM συνδέσεις την ίδια στιγμή. Σε αποτυχία επιστρέφει -1. Σχετίζεται μόνο με connection-based socket types. Χρειάζεται 3 παραμέτρους: η πρώτη είναι ο περι-

γραφέας της υποδοχής, το δεύτερο είναι η δομή για τη διεύθυνση της υποδοχής (δείκτης σε δομή sockaddr) και το τρίτο είναι το μέγεθος του δευτέρου ορίσματος.

Μεταφορά δεδομένων και κλείσιμο

Υπάρχουν πολλοί τρόποι για αποστολή και λήψη δεδομένων από ένα SOCK_STREAM socket. Αυτές είναι οι write(), read(), send() και recv(). Οι δυο τελεταίες είναι παρόμοιες με τις write() και read() απλά έχουν επιπρόσθετες παραμέτρους.

send

```
#include <sys/types.h>
#include <sys/socket.h>
ssize_t send(int sockfd, const void *buf, size_t ↩
    len, int flags);
```

recv

```
#include <sys/types.h>
#include <sys/socket.h>
ssize_t recv(int sockfd, void *buf, size_t len, ↩
    int flags);
```

Η επιπλέον παράμετρος flags [4] μπορεί να είναι συνδυασμός των παρακάτω flags ή και τίποτα από αυτά:

MSG_OOB	Στείλε out-of-band δεδομένα. Μόνο τα SOCK_STREAM sockets τα οποία έχουν δημιουργηθεί στο internet domain τα υποστηρίζουν
MSG_DONTROUTE	Αφορά τη διάρκεια και χρησιμοποιείται μόνο από διαγνωστικά προγράμματα
MSG_PEEK	Τα δεδομένα παραμένουν στην υποδοχή, ώστε μια επόμενη recv() να τα ξαναδεί

Πίνακας 1: Flags των send() και recv()

[4] out-of-bound data

Ένα stream socket καταστρέφεται καλώντας την close() και δίνοντας σαν παράμετρο τον περιγραφέα υποδοχής. Μπορείτε να δείτε το https://goo.gl/Ryx7Ku για μια σύνοψη όλων των κλήσεων. Όταν τρέχετε εφαρμογές οι οποίες επικοινωνούν με sockets μπορείτε να χρησιμοποιείτε την εντολή netstat για να βλέπετε πληροφορίες που αφορούν τα πρωτόκολλα επικοινωνίας, τα ports και την κατάσταση στης σύνδεσης. Μια συνηθισμένη χρήση της netstat είναι η

```
netstat -ant
```

Παράδειγμα: Stream Sockets in UNIX domain

Δείτε το ζευγάρι server-client του κώδικα 1 και 2, όπου παρουσιάζεται ένας echo server όπου ο server εμφανίζει τα μηνύματα του client. Παρομοίως, το ζευγάρι server-client του κώδικα 3 και 4 εμφανίζεται ο client να στέλνει πολλαπλά μηνύματα στον server και αυτός να του απαντά με ό,τι έλαβε.

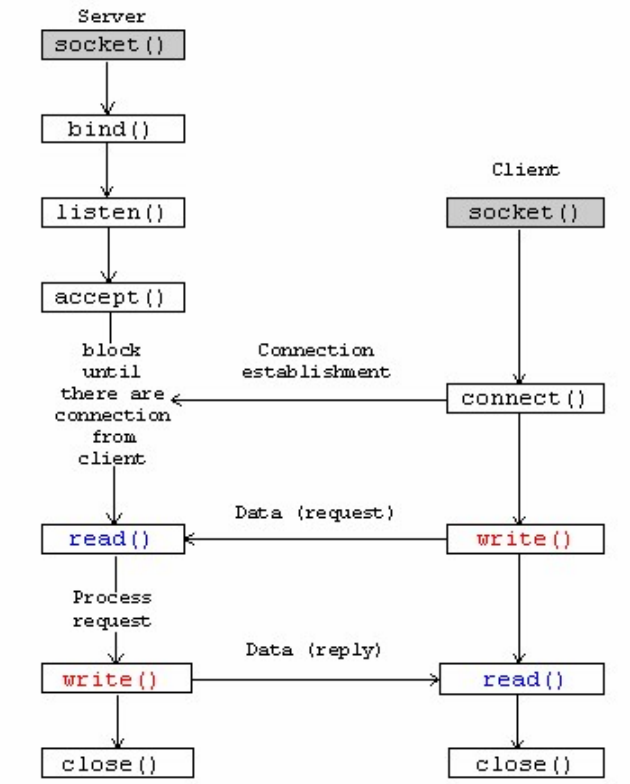
Διάταξη των byte

Όταν μεταβιβάζονται αλφαριθμητικά μεταξύ των υπολογιστών ενός δικτύου, δεν δημιουργείται ποτέ πρόβλημα, επειδή κατά την επικοινωνία διατηρείται πάντα η διάταξη των byte. Ωστόσο, κατά τη μεταβίβαση δυαδικών αριθμών μπορεί να υπάρξουν προβλήματα, αφού η διάταξη των byte σε έναν αριθμό διαφέρει από υπολογιστή σε υπολογιστή [5] (Σχήμα 4. The Intel x86 family uses the little endian format).

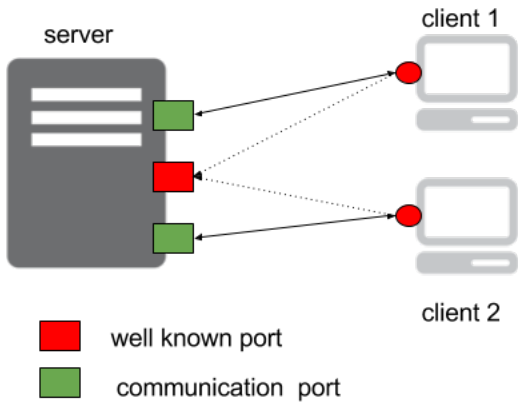
Η λύση είναι η αποστολή δυαδικών αριθμών με μια προσυμφωνημένη δικτυακή διάταξη. Έτσι όλοι οι αποστολείς θα πρέπει να μετατρέψουν τη διάταξη των byte από την τοπικη της μορφή στη δικτυακή, και όλοι οι παραλήπτες να μετατρέπουν τη δικτυακή μορφή στην τοπική. Αυτό γίνεται μόνο για δεδομένα τα οποία δεν είναι ήδη δομημένα με κάποιον άλλο τρόπο (π.χ. μια εικόνα jpeg θα τη στείλετε ως έχει).

Συνήθως δε χρειάζεται να γνωρίζετε ούτε την τοπική ούτε τη δικτυακή διάταξη byte, αφού υπάρ-

[5] http://en.wikipedia.org/wiki/Endianness



Σχήμα 2: Επικοινωνία με υποδοχές (SOCK_STREAM)



Σχήμα 3: Όταν συνδέεται ένας client

Listing 1: lab11_1_server.c: Παράδειγμα server socket σε unix domain #1

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <sys/un.h>
#include <unistd.h>
#include <string.h>

int main()
{
   int server_sockfd, client_sockfd;
   socklen_t server_len, client_len;
   struct sockaddr_un server_address;
   struct sockaddr_un client_address;

   /* Remove any old socket and create an unnamed socket ↩
        for the server. */

   unlink("server_socket");
   server_sockfd = socket(AF_UNIX, SOCK_STREAM, 0);

   /* Name the socket. */

   server_address.sun_family = AF_UNIX;
   strcpy(server_address.sun_path, "server_socket");
   server_len = sizeof(server_address);
   bind(server_sockfd, (struct sockaddr *)&server_address↩
        , server_len);

   /* Create a connection queue and wait for clients. */

   listen(server_sockfd, 5);
   while(1) {
      char ch[100];
      int bytes;

      printf("server waiting\n");

      /* Accept a connection. */

      client_len = sizeof(client_address);
      client_sockfd = accept(server_sockfd, (struct ↩
           sockaddr *)&client_address, &client_len);

      /* We can now read/write to client on ↩
           client_sockfd. */
      bytes=read(client_sockfd, ch, 100);
      ch[bytes]='\0';
      printf("Read from client : %s %d bytes\n", ch, ↩
           bytes);
      write(client_sockfd, ch, bytes);
      close(client_sockfd);

   }

}
```

Listing 2: lab11_1_client.c:Παράδειγμα client socket σε unix domain #1

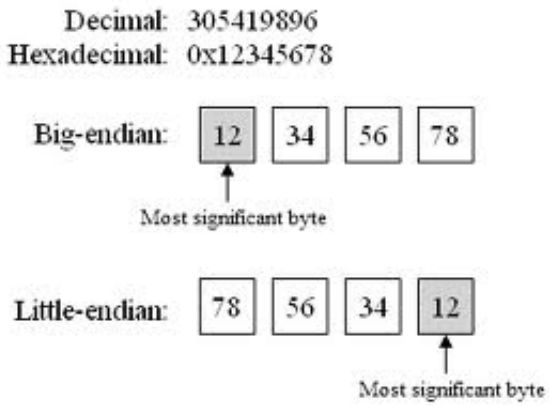
```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <sys/un.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

int main()
{
   int sockfd;
   int bytes;
   socklen_t len;
   struct sockaddr_un address;
   int result;
   char buff[100];

   /* Create a socket for the client. */

   sockfd = socket(AF_UNIX, SOCK_STREAM, 0);

   /* Name the socket, as agreed with the server. */

   address.sun_family = AF_UNIX;
   strcpy(address.sun_path, "server_socket");
   len = sizeof(address);

   /* Now connect our socket to the server's socket. */

   result = connect(sockfd, (struct sockaddr *)&address, ↩
        len);

   if(result == -1) {
      perror("oops: cannot connect!");
      exit(1);

   }

   /* We can now read/write via sockfd. */

   bytes = read(0,buff,sizeof(buff));
   if (bytes > 0) {
      printf("read %d bytes from stdin\n", bytes);
      if (write(sockfd,buff,bytes-1) != bytes-1 )
         perror("write error") ;

      bytes=read(sockfd, buff, 100);
      if (bytes>0){
         buff[bytes]='\0';
         printf("read %d bytes from server\n", bytes);
         printf("message from server = %s\n", buff);

      }

   }
   close(sockfd);
   exit(0);

}
```

Listing 3: lab11_2_server.c: Παράδειγμα server socket σε unix domain #2

```c
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>

#define SOCK_PATH "echo_socket"

int main(void)
{
    int s, s2, len;
    socklen_t t;
    struct sockaddr_un local, remote;
    char str[100];
    if ((s = socket(AF_UNIX, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    local.sun_family = AF_UNIX;
    strcpy(local.sun_path, SOCK_PATH);
    unlink(local.sun_path);
    len = strlen(local.sun_path) + sizeof(local.sun_family↩
        );
    if (bind(s, (struct sockaddr *)&local, len) == -1) {
        perror("bind");
        exit(1);
    }

    if (listen(s, 5) == -1) {
        perror("listen");
        exit(1);
    }

    for(;;) {
        int done, n;
        printf("Waiting for a connection...\n");
        t = sizeof(remote);
        if ((s2 = accept(s, (struct sockaddr *)&remote, &t↩
            )) == -1) {
            perror("accept");
            exit(1);
        }
        printf("Connected.\n");
        done = 0;
        do {
            n = recv(s2, str, 100, 0);
            if (n <= 0) {
                if (n < 0) perror("recv");
                done = 1;
            }
            if (!done)
                if (send(s2, str, n, 0) < 0) {
                    perror("send");
                    done = 1;
                }
        } while (!done);
        close(s2);
    }
    return 0;
}
```

Listing 4: lab11_2_client.c:Παράδειγμα client socket σε unix domain #2

```c
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#define SOCK_PATH "echo_socket"

int main(void)
{
    int s, t, len;
    struct sockaddr_un remote;
    char str[100];

    if ((s = socket(AF_UNIX, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    printf("Trying to connect...\n");

    remote.sun_family = AF_UNIX;
    strcpy(remote.sun_path, SOCK_PATH);
    len = strlen(remote.sun_path) + sizeof(remote.↩
        sun_family);
    if (connect(s, (struct sockaddr *)&remote, len) == -1)↩
        {
        perror("connect");
        exit(1);
    }

    printf("Connected.\n");

    while(printf("> "), fgets(str, 100, stdin), !feof(↩
        stdin)) {
        if (send(s, str, strlen(str), 0) == -1) {
            perror("send");
            exit(1);
        }

        if ((t=recv(s, str, 100, 0)) > 0) {
            str[t] = '\0';
            printf("echo> %s", str);
        } else {
            if (t < 0) perror("recv");
            else printf("Server closed connection\n");
            exit(1);
        }
    }
    close(s);

    return 0;
}
```



Σχήμα 4: Διάταξη μικρού και μεγάλου άκρου

χει τυποποιημένο σύνολο συναρτήσεων «μετάφρασης» οι οποίες πραγματοποιούν τις μετατροπές για ακεραίους των 16 και 32 bit.

htons	Μετατροπή 16bit τιμής από τοπική σε δικτυακή διάταξη byte (port number)
htonl	Μετατροπή 32bit τιμής από τοπική σε δικτυακή διάταξη byte (ip address)
ntohs	Μετατροπή 16bit τιμής από δικτυακή σε τοπική διάταξη byte (read a port number from sockaddr)
ntohl	Μετατροπή 32bit τιμής από δικτυακή σε τοπική διάταξη byte (read ip from sockaddr)

Πίνακας 2: Συναρτήσεις «μετάφρασης»

inet_ntop

Σύνταξη: inet_ntop (numeric to presentation, binary value (socket address structure) to ASCII string (dotted-decimal string))

```
#include <arpa/inet.h>
const char *inet_ntop(int af, const void *src, char *dst,↩
      socklen_t size);
```

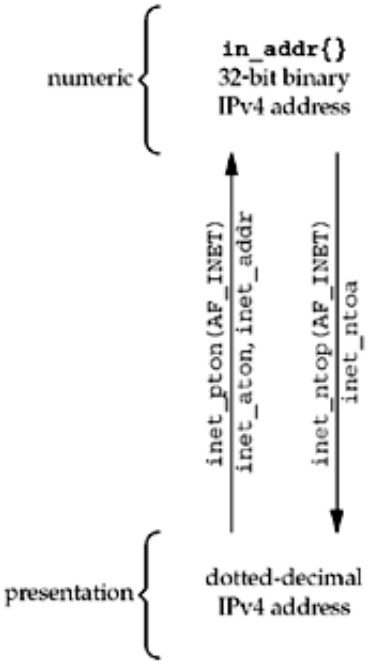
Η κλήση αυτή μετατρέπει τη δομή διεύθυνσης δικτύου src η οποία είναι της af address family σε ένα string. Το επιστρεφόμενο string αντιγράφεται σε ένα buffer που δείχνει ο dst pointer, ο οποίος πρέπει να είναι ένας no-null pointer. Το size είναι το μέγεθος των επιστρεφόμενων δεδομένων. Για το λόγο αυτό (size) υπάρχουν οι ακόλουθες δηλώσεις[6]:

Σημείωση: Για να τρέξουμε τον κώδικα του εργαστηρίου σε περιβάλλον Solaris, θα πρέπει να βάλουμε στο gcc τις παραμέτρους -lsocket και -lnsl

```
#define INET_ADDRSTRLEN 16 /* for IPv4 dotted-decimal */
#define INET6_ADDRSTRLEN 32 /* for IPv6 hex string */
```

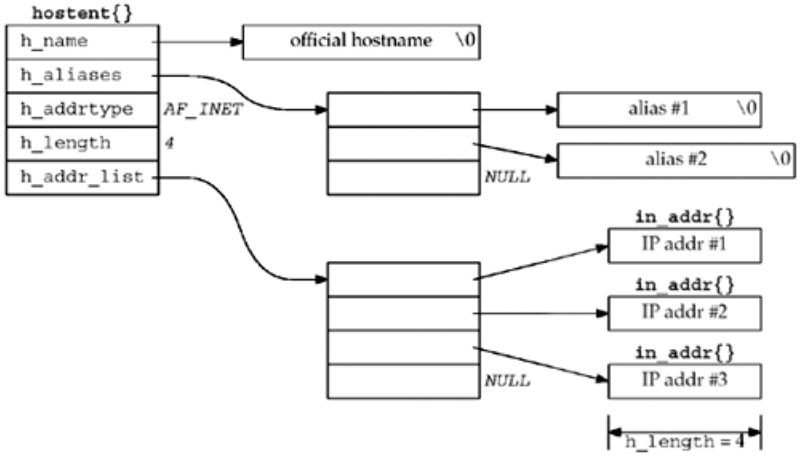
Υπάρχει και η αντίστροφη μετατροπή, από presentation to numeric (inet_pton). Μετατρέπει μια IP διεύθυνση από string σε network byte order. (Σχήμα 5)

```
#include <arpa/inet.h>
int inet_pton(int af,const char *src,void *dst);
```



Σχήμα 5: Συναρτήσεις μετατροπής διευθύνσεων network byte order σε presentation

[6]Πρέπει να κάνουμε include τη ‹netinet/in.h›



Σχήμα 6: Η δομή hostent

Δομή hostent

Σε πολλές περιπτώσεις μπορεί να γνωρίζουμε το όνομα (hostname) ενός υπολογιστή, παρά τη δικτυακή του διεύθυνση (ip address). Για να μπορούμε να πάρουμε την ip address από ένα hostname, είναι να χρησιμοποιήσουμε την gethostbyname.

Στο σχήμα 6 θα δείτε τη δομή hostent. Για να αναζητήσουμε ένα όνομα υπολογιστή στο διαδίκτυο, χρησιμοποιούμε την κλήση gethostbyname δέχεται σαν παράμετρο το hostname και μας επιστρέφει έναν δείκτη σε αυτή τη δομή, η οποία περιέχει τις IP διευθύνσεις για τον υπολογιστή αυτό. Συγκεκριμένα:

```
struct hostent {
   char *h_name; /* official name of host */
   char **h_aliases; /* alias list */
   int h_addrtype; /* host address type */
   int h_length; /* length of address */
   char **h_addr_list; /* list of addresses */
}
```

INADDR_ANY

Χρησιμοποιούμε τη σταθερά INADDR_ANY, ούτως ώστε να πάρει αυτόματα τη δικτυακή διεύθυνση του μηχανήματος. Τη χρησιμοποιούμε για να κάνουμε bind σε όλα τα network interfaces (π.χ. όταν είμαστε συνδεδεμένοι και με wifi και με ethernet, θα έχουμε 2 network interfaces με 2 ip addresses). Με αυτόν τον τρόπο αφήνουμε το λειτουργικό να επιλέξει όποια θέλει.

Παράδειγμα: Stream Sockets in internet domain

Ο κώδικας 7 παρουσιάζει έναν stream socket server, βασισμένο σε TCP. Ο κώδικας 8 παρουσιάζει τον αντίστοιχο client. Για να εκτελστεί ο server χρειάζεται μια παράμετρο που είναι η port στην οποία θα ακούει. Αντίστοιχα ο client [7] χρειάζεται δυο παραμέτρους: την ip διεύθυνση του server και την port.

Datagram sockets

Ένα datagram δεν απαιτεί την εγκαθίδρυση σύνδεσης. Κάθε μήνυμα κουβαλά τη διεύθυνση προορισμού του. Αν μια συγκεκριμένη τοπική διεύθυνση χρειάζεται, τότε μια κλήση bind() πρέπει να προηγηθεί της μεταφοράς δεδομένων. Τα δεδομένα αποστέλλονται και λαμβάνονται χρησιμοποιώντας της κλήσεις sendto() ή sendmsg(). Η sendto() είναι σαν την send() με τη διεύθυνση προορισμού να καθορίζεται. Για να λάβουμε

[7] Η δομή hostent έχει και το εξής:
#define h_addr h_addr_list[0] /* for backward compatibility */

Listing 5: lab11_3.c:Παράδειγμα μετατροπών network byte order σε presentation και αντίστροφα

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <string.h>
#include <errno.h>
#include <netinet/in.h>

int main(int argc, char *argv[])
{
   struct in_addr addr;
   if (argc < 2)
   {
      fprintf(stderr, "usage: ./quad_to_byte [ip_address↩
          ]\n");
      return -1;
   }

   char *byte_order = malloc(sizeof(argv[1]));
   if (!byte_order)
   {
      fprintf(stderr, "Could not allocate memory for ↩
          conversion.\n");
      return -1;
   }

   // Convert address to byte order
   if (!inet_pton(AF_INET, argv[1], &addr))
   {
      fprintf(stderr, "Could not convert address\n");
      free(byte_order);
      return -2;
   }

   // Print out network byte order
   fprintf(stdout, "Network byte order: %d\n", addr.↩
       s_addr);

   // Convert it back to our dot quad to verify
   if (inet_ntop(AF_INET, &addr.s_addr, byte_order, ↩
       INET_ADDRSTRLEN) == NULL)
   {
      fprintf(stderr, "Could not convert byte to address↩
          \n");
      fprintf(stderr, "%s\n", strerror(errno));
      free(byte_order);
      return -3;
   }

   // Display our dot quad converted from the network ↩
       byte order
   fprintf(stdout, "Dot quad: %s\n", byte_order);
   free(byte_order);
   return 0;
}
```

Listing 6: lab11_4.c:Παράδειγμα gethostbyname

```c
#include <stdlib.h>
#include <stdio.h>
#include <netdb.h>

/* paddr: print the IP address in a standard decimal ↩
     dotted format */
void paddr(unsigned char *a)
{
   printf("%d.%d.%d.%d\n", a[0], a[1], a[2], a[3]);
}

int main(int argc, char **argv)
{
   struct hostent *hp;
   if (argc != 2) {
      printf("Not given a hostname");
      exit(1);
   }
   char *host = argv[1];
   int i;

   hp = gethostbyname(host);
   if (!hp)
   {
      fprintf(stderr, "could not obtain address of %s\n"↩
          , host);
      return 0;
   }
   for (i = 0; hp->h_addr_list[i] != 0; i++)
      paddr((unsigned char *) hp->h_addr_list[i]);
   exit(0);
}
```

Listing 7: lab11_5_server.c:Παράδειγμα server socket σε network domain

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
void error(const char *msg)
{
    perror(msg);
    exit(1);
}

int main(int argc, char *argv[])
{
    int sockfd, newsockfd, portno;
    socklen_t clilen;
    char buffer[256];
    struct sockaddr_in serv_addr, cli_addr;
    int n;
    char str[INET_ADDRSTRLEN];

    if (argc < 2)
    {
        fprintf(stderr, "No port provided\n");
        exit(1);
    }
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
        error("ERROR opening socket");
   /* bzero((char *) &serv_addr, sizeof(serv_addr)); */
    memset((char *) &serv_addr, 0, sizeof(serv_addr));
    portno = atoi(argv[1]);
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(portno);
    if (bind(sockfd,(struct sockaddr *) &serv_addr, sizeof↩
         (serv_addr)) < 0)
        error("ERROR on binding");
    listen(sockfd, 5);
    clilen = sizeof(cli_addr);
    newsockfd = accept(sockfd, (struct sockaddr *) &↩
         cli_addr, &clilen);
    if (newsockfd < 0)
        error("ERROR on accept");

    if (inet_ntop(AF_INET, &cli_addr.sin_addr, str, ↩
         INET_ADDRSTRLEN) == NULL) {
        fprintf(stderr, "Could not convert byte to address↩
             \n");
        exit(1);
    }
    fprintf(stdout, "The client address is :%s\n", str);


    bzero(buffer, 256);
    n = read(newsockfd, buffer, 255);
    if (n < 0) error("ERROR reading from socket");
    printf("Here is the message: %s\n", buffer);
    n = write(newsockfd, "message received", 17);
    if (n < 0) error("ERROR writing to socket");
    close(newsockfd);
    close(sockfd);
    return 0;
}
```

Listing 8: lab11_5_client.c:Παράδειγμα client socket σε network domain

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

void error(const char *msg)
{
    perror(msg);
    exit(0);
}

int main(int argc, char *argv[])
{
    int sockfd, portno, n;
    struct sockaddr_in serv_addr;
    struct hostent *server;

    char buffer[256];
    if (argc < 3)
    {
        fprintf(stderr, "usage %s hostname port\n", argv↩
             [0]);
        exit(0);
    }
    portno = atoi(argv[2]);
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
        error("ERROR opening socket");
    server = gethostbyname(argv[1]);
    if (server == NULL)
    {
        fprintf(stderr, "ERROR, no such host\n");
        exit(0);
    }
    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr,
          (char *)&serv_addr.sin_addr.s_addr,
          server->h_length);
    serv_addr.sin_port = htons(portno);
    if (connect(sockfd, (struct sockaddr *) &serv_addr, ↩
         sizeof(serv_addr)) < 0)
        error("ERROR connecting");
    printf("Please enter the message: ");
    bzero(buffer, 256);
    fgets(buffer, 255, stdin);
    n = write(sockfd, buffer, strlen(buffer));
    if (n < 0)
        error("ERROR writing to socket");
    bzero(buffer, 256);
    n = read(sockfd, buffer, 255);
    if (n < 0)
        error("ERROR reading from socket");
    printf("%s\n", buffer);
    close(sockfd);
    return 0;
}
```

Listing 9: lab11_6_server.c:Παράδειγμα UDP server σε network domain

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

int main()
{
   int sfd, n;
   socklen_t len;
   char line[128];
   struct sockaddr_in saddr, caddr;

   sfd = socket(AF_INET, SOCK_DGRAM, 0);

   bzero(&saddr, sizeof(saddr));
   saddr.sin_family = AF_INET;
   saddr.sin_addr.s_addr = htonl(INADDR_ANY);
   saddr.sin_port = htons(2910);

   bind(sfd, (struct sockaddr *)&saddr, sizeof(saddr));

   printf("Server running\n");
   for (;;)
   {
      len = sizeof(caddr);
      n = recvfrom(sfd, line, 128, 0, (struct sockaddr ↩
          *)&caddr, &len);
      line[n] = '\0';
      printf("Received: %s\n", line);
      sendto(sfd, line, n, 0, (struct sockaddr *)&caddr,↩
           len);
   }

   return 0;
}
```

datagram socket μηνύματα χρησιμοποιούμε την recvfrom() ή την recvmsg(). Ενώ η recv() απαιτεί έναν buffer για τα αφιχθέντα δεδομένα, η recvfrom() απαιτεί δύο buffers, έναν για το εισερχόμενο μήνυμα και ένα για τη λήψη της διεύθυνσης αποστολέα.

Τα datagram sockets μπορούν επίσης να χρησιμοποιήσουν την connect() ώστε να συνδέσουν την υποδοχή σε συγκεκριμένη υποδοχή προορισμού. Όταν αυτό συμβαίνει, χρησιμοποιούνται οι send() και recv() για την αποστολή/παραλαβή δεδομένων. Οι accept() και listen() δεν χρησιμοποιούνται από τα datagram sockets.

Περισσότερες πληροφορίες μπορείτε να βρείτε στα [3, 11, 9, 1, 2, 4, 8, 10, 6, 5, 7] και στο βιβλίο [12].

Listing 10: lab11_6_client.c:Παράδειγμα UDP client σε network domain

```
#include <stdio.h>
#include <sys/socket.h>
#include <string.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#define MAX 100

int main(int argc, char **argv)
{
   int sfd, n;
   socklen_t len;
   char sline[MAX], rline[MAX + 1];
   struct sockaddr_in saddr;

   if (argc != 2)
   {
      printf("Usage: %s ipaddress\n", argv[0]);
      return -1;
   }

   sfd = socket(AF_INET, SOCK_DGRAM, 0);

   bzero(&saddr, sizeof(saddr));
   saddr.sin_family = AF_INET;
   inet_pton(AF_INET, argv[1], &saddr.sin_addr);
   saddr.sin_port = htons(2910);

   printf("Client Running\n");
   while (fgets(sline, MAX, stdin) != NULL)
   {
      len = sizeof(saddr);
      sendto(sfd, sline, strlen(sline), 0, (struct ↩
           sockaddr *)&saddr, len);
      n = recvfrom(sfd, rline, MAX, 0, NULL, NULL);
      rline[n] = 0;
      fputs(rline, stdout);
   }

   return 0;
}
```

Αναφορές

[1] Beej's Guide to Network Programming. http://beej.us/guide/bgnet/. Accessed: 2019-01-07.

[2] Endianness. http://en.wikipedia.org/wiki/Endianness. Accessed: 2019-01-07.

[3] Network Domain Sockets. http://www.cs.rutgers.edu/~pxk/rutgers/notes/sockets/. Accessed: 2019-01-07.

[4] out-of-bound data. http://www.gnu.org/software/libc/manual/html_node/Out_002dof_002dBand-Data.html. Accessed: 2019-01-07.

[5] Reading Data from Sockets. http://faq.cprogramming.com/cgi-bin/smartfaq.cgi?id=1044780608&answer=1108255660. Accessed: 2019-01-07.

[6] Socket Address Structures. http://www.informit.com/articles/article.aspx?p=169505&seqNum=2. Accessed: 2019-01-07.

[7] Sockets. http://www.python4science.eu/sockets.html. Accessed: 2019-01-07.

[8] Sockets Introduction. https://www.cs.rutgers.edu/~pxk/416/notes/16-sockets.html. Accessed: 2019-01-07.

[9] Sockets Tutorial. http://www.linuxhowtos.org/C_C++/socket.htm. Accessed: 2019-01-07.

[10] TCP Connection Establishment and Termination. http://www.masterraghu.com/subjects/np/introduction/unix_network_programming_v1.3/ch02lev1sec6.html. Accessed: 2019-01-07.

[11] Unix Sockets. http://beej.us/guide/bgipc/output/html/multipage/unixsock.html. Accessed: 2019-01-07.

[12] W. R. Stevens, B. Fenner, and A. M. Rudoff. UNIX Network Programming: The Sockets Networking API, volume 1. Addison-Wesley Professional, 2004.