

Στην περίπτωση που δύο εντελώς ανεξάρτητες διεργασίες που τρέχουν σε διαφορετικά υπολογιστικά συστήματα χρειάζεται να επικοινωνήσουν πρέπει να χρησιμοποιηθεί το επικοινωνιακό υποδίκτυο που ενώνει τις δύο μηχανές. Την επικοινωνία αυτή καθιστούν δυνατή τα sockets.

Έτσι, μπορούν διεργασίες από διαφορετικούς πατέρες να επικοινωνούν.

Το **socket** ορίζεται ως το άκρο (endpoint) ενός καναλιού επικοινωνίας διεργασιών. Υπάρχει μια συγκεκριμένη διαδικασία δημιουργίας αυτού του καναλιού, μετά από αυτήν όμως οι διεργασίες μπορούν να επικοινωνούν με συνηθισμένες κλήσεις `read()` και `write()`.

Η δημιουργία ενός socket δεν συνεπάγεται και τη δημιουργία του καναλιού. Για να επιτευχθεί κάτι τέτοιο θα πρέπει δύο διεργασίες να δημιουργήσουν η κάθε μία ένα socket και έπειτα η μια από αυτές να περιμένει σύνδεση και η άλλη να τη ζητήσει. Με άλλα λόγια, η μια διεργασία αναλαμβάνει το ρόλο του εξυπηρετητή (**server** - αυτή που περιμένει τη σύνδεση) και η άλλη το ρόλο του πελάτη (**client** - αυτή που κάνει την αίτηση για σύνδεση). Μπορούμε να έχουμε έναν server και πολλούς clients.

Τα sockets που δημιουργούνται από διαφορετικά προγράμματα χρησιμοποιούν “ονόματα” για να αναφερθούν μεταξύ τους και να γίνει η σύνδεση.

Το πεδίο από το οποίο λαμβάνει όνομα το socket αποτελεί το domain ή address family. Δύο είναι τα domain που χρησιμοποιούνται από τις εφαρμογές για τη διευθυνσιοδότηση των sockets τους:

§ Unix domain (AF_UNIX)

§ Internet domain (AF_INET)

-> **Unix domain (AF_UNIX)** στο οποίο κάθε socket ταυτοποιείται μέσω ενός ονόματος “αρχείου” (και επομένως πρέπει οι διεργασίες να έχουν πρόσβαση σε ένα κοινό σύστημα αρχείων). Κάθε κανάλι ταυτοποιείται μονοσήμαντα με την τριάδα {πρωτόκολλο / τοπικό αρχείο / απομακρυσμένο αρχείο}, όπου η λέξη απομακρυσμένο δηλώνει την άλλη διεργασία. Είναι φανερό ότι αυτός ο τρόπος δεν μπορεί να χρησιμοποιηθεί πάνω από το δίκτυο και δεν θα μας απασχολήσει στη συνέχεια.

-> **Internet domain (AF_INET)** στο οποίο κάθε socket ταυτοποιείται από μια δυάδα που αποτελείται από την IP διεύθυνση (**IP address**) του συστήματος στο οποίο έχει δημιουργηθεί το socket και ένα θετικό ακέραιο ο οποίος ονομάζεται **port number** του socket και δηλώνει σε ποια “θύρα” επικοινωνίας μέσα στο σύστημα αυτό έχει “συνδεθεί” το socket. Κάθε κανάλι επικοινωνίας ταυτοποιείται μονοσήμαντα με την πεντάδα {πρωτόκολλο / τοπική μηχανή / τοπικό port / απομακρυσμένη μηχανή / απομακρυσμένο port}.

Όσον αφορά τη **μετάδοση των δεδομένων** πάνω στο κανάλι επικοινωνίας μετά τη δημιουργία του, υπάρχουν δύο τρόποι:

§ Stream communication (SOCK_STREAM)

§ Datagram communication (SOCK_DGRAM)

-> **Stream communication (SOCK_STREAM)**, όπου τα δεδομένα στέλνονται πάνω σε ήδη υπάρχον κανάλι επικοινωνίας (για αυτό και ο συγκεκριμένος τρόπος ονομάζεται connection oriented). Το πρωτόκολλο επικοινωνίας TCP μέσα στον πυρήνα του λειτουργικού συστήματος φροντίζει και αναλαμβάνει όλους τους ελέγχους για λάθη στη μετάδοση των δεδομένων καθώς και την επαναμετάδοση των πακέτων που χάθηκαν. Έτσι εξασφαλίζονται αξιόπιστα κανάλια επικοινωνίας στα οποία οι πληροφορίες φτάνουν με τη σωστή σειρά και πλήρεις, αν και με μικρότερη ταχύτητα.

-> **Datagram communication (SOCK_DGRAM)**, όπου τα δεδομένα μεταδίδονται σε μικρά αυτόνομα “πακέτα” που ονομάζονται datagrams και το κανάλι επικοινωνίας δημιουργείται για κάθε μήνυμα ξεχωριστά (connectionless), ενώ η παραλαβή του δεν είναι εγγυημένη. Ο έλεγχος μετέδοσης με τη σωστή σειρά και η ευθύνη για τον κατακερματισμό (fragmentation) και την επανασυναρμολόγηση (reassembly) των δεδομένων ανήκουν στις εφαρμογές. Με τον τρόπο αυτό, που αντιστοιχεί στο πρωτόκολλο επικοινωνίας UDP, επιτυγχάνονται υψηλές ταχύτητες μετάδοσης.

Θα ασχοληθούμε κυρίως με stream sockets στο internet domain, που είναι και ο πιο διαδεδομένος τρόπος υλοποίησης πολλών από τις υπηρεσίες που παρέχονται πάνω από δίκτυο.

Η αρχικοποίηση ενός socket περιλαμβάνει καταρχήν τη δημιουργία του με μια κλήση `socket()`:

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

Η παράμετρος `protocol` προσδιορίζει ένα πρωτόκολλο το οποίο θα υλοποιήσει την επικοινωνία πάνω στο κανάλι η οποία μπορεί να είναι 0 για να κάνει το λειτουργικό σύστημα την επιλογή.

Με χρήση της κλήσης `bind()` δίνεται διεύθυνση σε ένα socket:

```
int bind(int sd, const struct sockaddr *addr, int addrlen);
```

Η πρώτη παράμετρος `sd` προσδιορίζει τον socket descriptor που έχουμε δημιουργήσει με κλήση στην `socket()`. Η τρίτη παράμετρος είναι το μέγεθος της δομής `struct sockaddr_in` η οποία περνάει με δείκτη ως δεύτερη παράμετρος. Τα πεδία της δομής που μας ενδιαφέρουν είναι τα `sin_family`, `sin_port`, `sin_addr`.

Το πρώτο πεδίο `sin_family` πρέπει να είναι ίδιο με την παράμετρο `domain` κατά την δημιουργία του socket.

Το πεδίο `sin_port` καθορίζει το port number του socket και δεν πρέπει να χρησιμοποιείται από κάποιο άλλο socket. Μπορεί όμως να έχει την τιμή 0, οπότε διαλέγει το σύστημα ένα ελεύθερο port (> 1024).

Το τελευταίο από τα πεδία είναι η `internet διεύθυνση` της μηχανής στην οποία θα δέχεται συνδέσεις το socket. Αν π.χ. μία μηχανή είναι συνδεδεμένη ταυτόχρονα σε περισσότερα από ένα δίκτυα, τότε το socket μπορεί να προσδιορίσει από ποιο δίκτυο μόνο θα δέχεται συνδέσεις. Συνήθως όμως χρησιμοποιείται η τιμή `INADDR_ANY`, που σημαίνει ότι δεχόμαστε συνδέσεις από παντού.

Οι δύο τελευταίες παράμετροι είναι αναγκαίο να έχουν μια μορφή η οποία θα είναι αποδεκτή από οποιαδήποτε μηχανή συνδέεται σε δίκτυο, ώστε να μη υπάρχουν ασυμβατότητες. Για τον σκοπό αυτό υπάρχουν οι συναρτήσεις βιβλιοθήκης `htonl()` (host to network - long) και `htons()` (host to network - short) οι οποίες μετατρέπουν το όρισμά τους (long και short αντίστοιχα) στην γενικά αποδεκτή μορφή.

Μια διεργασία-πελάτης μπορεί να ζητήσει σύνδεση με κάποιον εξυπηρετητή με χρήση της κλήσης `connect()`:

```
int connect(int sd, struct sockaddr *addr, int addrlen);
```

Αφού μια διεργασία-εξυπηρετητής διευθυνσιοδοτήσει το socket της με χρήση της `bind()` μπορεί να δεχτεί κλήσεις σύνδεσης με χρήση της κλήσης `accept()`:

```
int accept(int sd, struct sockaddr *addr, int *addrlen);
```

Θα πρέπει να υλοποιήσετε έναν client που επικοινωνεί με δικό μας server μέσω sockets. Μέσω της επικοινωνίας θα μεταφέρονται και θα εγκρίνονται αιτήματα μετακίνησης και θα παρέχονται πληροφορίες αναφορικά με την τρέχουσα θερμοκρασία κα.

```
./ask4 [--host HOST] [--port PORT] [--debug]
```

Το πρόγραμμα είναι εφικτό να δεχθεί τα παρακάτω ορίσματα κατά την εκτέλεση

--host HOST: Συνδέεται στο HOST αντί για το default.

--port PORT: Συνδέεται στο PORT αντί για το default

--debug: Εκτυπώνει σε μήνυμα τα δεδομένα που έχουν αποσταλεί ή ληφθεί

Default: HOST `iot.dslab.pub.ds.open-cloud.xyz` PORT 18080

Εντολές από terminal:

help : Εκτυπώνει help message

exit: Έξοδος

get: Ανάκτηση δεδομένων server

N name surname reason: Άδεια εξόδου στην καραντίνα (Στην περίπτωση λανθασμένου μηνύματος ο server απαντάει 'try again')

Παράδειγμα get:

Ο server θα απαντήσει με τις πιο πρόσφατες μετρήσεις αισθητήρα στην ακόλουθη μορφή

X YYY ZZZZ WWWWWWWWWW, όπου

YYY είναι η φωτεινότητα

ZZZZ είναι η θερμοκρασία (χρειάζεται διαίρεση με 100)

WWWWWWWWWWW είναι UNIX timestamp (χρησιμοποιώ localtime()) και

X είναι ένας ακέραιος αριθμός που δείχνει τον τύπο συμβάντος, σύμφωνα με το παρακάτω πρωτόκολλο

0: boot , 1: setup, 2: interval, 3: button, 4: motion

Μια διεργασία είναι εφικτό να δεχθεί δεδομένα από

ένα σύνολο file descriptors ή από το terminal, μέσω της select(). Η select() επιτρέπει σε ένα πρόγραμμα να παρακολουθεί πολλούς περιγραφητές αρχείων, έως ότου ένας ή περισσότεροι από τους περιγραφητές αρχείων γίνουν «έτοιμοι». Ένας περιγραφητής θεωρείται έτοιμος εάν είναι δυνατή η εκτέλεση μιας λειτουργίας εισόδου / εξόδου (π.χ. ανάγνωση ή έγγραφη) χωρίς αποκλεισμό.

```
size_t read ( int fd , void * buf , size_t count ) ;
```

Η συνάρτηση read() διαβάζει τα περιεχόμενα του ανοιχτού αρχείου που καθορίζεται από τον περιγραφέα του στο όρισμα εισόδου «fd». Αυτή η συνάρτηση διαβάζει και αντιγράφει το πλήθος των byte στην προσωρινή μνήμη που δείχνει το buf. Το αναγνωριστικό "fd" είναι ένας ακέραιος αριθμός που επιστρέφεται ως αποτέλεσμα της συνάρτησης open() όταν ανοίγει το αρχείο.

Εάν η συνάρτηση read() επιστρέψει με επιτυχία, επιστρέφει τον αριθμό των byte που διαβάστηκαν. Ένα αποτέλεσμα ίσο με 0 σημαίνει ότι το αρχείο διαβάζεται μέχρι το τέλος και -1 σημαίνει ότι παρουσιάζεται σφάλμα.

```
int write ( int fd , void * buf , size_t n ) ;
```

Η συνάρτηση write() γράφει σε ένα ανοιχτό αρχείο. Αυτή η συνάρτηση εγγράφει τα περιεχόμενα του buffer που υποδεικνύεται με "buf" στο αρχείο που καθορίζεται από τον περιγραφέα του στο όρισμα εισόδου "fd". Το μέγεθος του μπλοκ που θα εγγραφεί στο αρχείο πρέπει να καθοριστεί στο όρισμα εισαγωγής "n".

Εάν η κλήση είναι επιτυχής, επιστρέφει τον αριθμό των χαρακτήρων που έχουν εισαχθεί. Εάν παρουσιαστεί σφάλμα κατά την έγγραφη, επιστρέφει ένα αποτέλεσμα ίσο με -1.