Τ. Ε. Ι. ΗΠΕΙΡΟΥ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ Τ.Ε

ΔΙΚΤΥΑΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ : ΧΕΙΡΙΣΜΟΣ SOCKETS

Στέλιος Σιακαλλής ΑΜ 10770

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Επιβλέπων
ΣΤΕΡΓΙΟΥ ΕΛΕΥΘΕΡΙΟΣ
ΕΠΙΚΟΥΡΟΣ ΚΑΘΗΓΗΤΗΣ

ΑΡΤΑ 2013-2014

ΕΥΧΑΡΙΣΤΙΕΣ

Η ολοκλήρωση αυτής της πτυχιακής υλοποιήθηκε με την υποστήριξη ενός αριθμού ανθρώπων στους οποίους θα ήθελα να εκφράσω τις θερμότερες ευχαριστίες μου. Πρώτα από όλους θα ήθελα να ευχαριστήσω τους γονείς μου, οι οποιοί στήριξαν τις σπουδές μου με διάφορους τρόπους φροντίζοντας για την καλύτερη δυνατή μόρφωση μου.

Τέλος, θα ήθελα να ευχαριστήσω τον επιβλέβοντα καθηγητή μου κ.Στεργίου για την εμπιστοσύνη και την αφιέρωση πολύτιμου χρόνου ώστε να ολοκληρωθεί η εργασία αυτή.

Στέλιος Σιακαλλής

Νοέμβριος 2013

ΠΕΡΙΛΗΨΗ

Σκοπός της πτυχιακής είναι η κατανόηση του διαδικτυακού προγραμματισμού και ειδικότερα στην γλώσσα προγραμματισμού C. Μέσα από την πτυχιακή αυτή θα εξετάσουμε τι είναι ο διαδικτυακός προγραμματισμός και θα δούμε πως δημιουργούμε sockets, πως συνδέεται ο client με τον server και ανταλλάσουν διάφορα μηνύματα και πληροφορίες καθώς και πως διαγράφουμε μετά τα sockets που δημιουργήσαμε. Με λίγα λόγια όλο τον κύκλο ζωής των sockets καθώς και την διαχείριση αυτών. Ακόμη θα δούμε περιληπτικά μερικούς όρους που θα βοηθήσουν στην κατανόηση της λειτουργιάς του προγράμματος μας.

Η συγκεκριμένη πτυχιακή βασίστηκε στην γλώσσα C,δημιουργήθηκε σε λειτουργικό τύπου UNIX,δοκιμάστηκε και σε Windows μέσω του Cygwin. Επίσης στο πρόγραμμα χρησιμοποιήθηκαν threads για την καλύτερη και ομαλότερη λειτουργιά των διαφόρων client με τον server.

ΠΕΡΙΕΧΟΜΕΝΑ

1 ο ΚΕΦΑΛΑΙΟ

ΔΙΚΤΥΑ ΚΑΙ ΔΙΕΘΥΝΣΕΙΣ

1.1 Εισαγωγή

Σήμερα οι άνθρωποι χρησιμοποιούν τους υπολογιστές για να κάνουν τηλεφωνικές κλήσεις , να παρακολουθήσουν τηλεόραση , να στείλουν άμεσα μηνύματα στους τους φίλους τους , να παίξουν παιχνίδια με άλλους ανθρώπους , και να αγοράσουν σχεδόν οτιδήποτε μπορείτε να σκεφτείτε , από τραγούδια μέχρι και αυτοκίνητα . Η δυνατότητα των προγραμμάτων να επικοινωνούν μέσω του Διαδικτύου κάνει όλα αυτά δυνατά . Είναι δύσκολο να πούμε πόσοι μεμονωμένοι υπολογιστές είναι πλέον προσβάσιμοι μέσω του Internet , αλλά μπορούμε με ασφάλεια να πούμε ότι αυξάνεται με ταχείς ρυθμούς . Επιπλέον, οι νέες εφαρμογές που αναπτύσσονται κάθε μέρα δεν θα αργήσει ο αριθμός αυτός να είναι σε δισεκατομμύρια. Με την ώθηση για ολοένα αυξανόμενο εύρος ζώνης και την πρόσβαση , ο αντίκτυπος του Διαδικτύου θα συνεχίσει να αυξάνεται για το προβλέψιμο μέλλον .

Πώς ένα πρόγραμμα επικοινωνεί με ένα άλλο πρόγραμμα σε ένα δίκτυο ;

Ο στόχος αυτής της πτυχιακής είναι η κατανόηση της απάντησης στο ερώτημα αυτό , στο πλαίσιο της γλώσσας προγραμματισμού C . Για μεγάλο χρονικό διάστημα , C ήταν η πρώτη γλώσσα επιλογής για την υλοποίηση λογισμικού για δίκτυα επικοινωνίας . Πράγματι , η διεπαφή προγραμματισμού εφαρμογών (API) που είναι γνωστή ως Sockets αναπτύχθηκε για πρώτη φορά στην C.

Πριν ξεκινήσουμε τις λεπτομέρειες των υποδοχών , ωστόσο, αξίζει να ρίξουμε μια σύντομη ματιά στη μεγάλη εικόνα των δικτύων και πρωτοκόλλων για να δούμε πού ταιριάζει ο κώδικας μας .

1.2 Δίκτυα, Πακέτα και πρωτόκολλα επικοινωνίας.

Ένα δίκτυο υπολογιστών αποτελείται από μηχανήματα συνδέονται μεταξύ τους με κανάλια επικοινωνίας . Καλούμε αυτά τα μηχανήματα hosts και routers . Hosts είναι υπολογιστές που τρέχουν εφαρμογές όπως το πρόγραμμα περιήγησης στο Web σας ή ένα πρόγραμμα κοινής χρήσης αρχείων. Τα προγράμματα που τρέχουν σε κεντρικούς είναι οι πραγματικοί « χρήστες » του δικτύου . Router (ή ακόμη και gateways) είναι μηχανήματα των οποίων η εργασία είναι να αναμεταδώσει ή να διαβιβάσει πληροφορίες από ένα κανάλι επικοινωνίας σε ένα άλλο. Μπορούν να τρέξουν προγράμματα, αλλά συνήθως δεν τρέχουν προγράμματα εφαρμογών .

Οι δρομολογητές είναι σημαντικοί μόνο και μόνο επειδή δεν είναι εφικτό να συνδεθεί κάθε host απευθείας σε κάθε άλλο host . Αντί 'αυτού , μερικοί host συνδέονται σε ένα router , το οποίο συνδέεται με άλλους routers , και έτσι σχηματίζουν το δίκτυο .

Εικόνα 1 - Μια τοπολογία δικτύου, η τοπολογία αστέρα

Με τον όρο πληροφορία εννοούμε ακολουθίες bytes που κατασκευάζονται και ερμηνεύονται από τα προγράμματα . Στα δίκτυα υπολογιστών αυτή η ακολουθία πληροφοριών γενικά ονομάζεται πακέτα (packets). Ένα πακέτο περιέχει πληροφορίες ελέγχου που το δίκτυο χρησιμοποιεί για να κάνει τη δουλειά του και μερικές φορές επίσης περιλαμβάνει τα δεδομένα των χρηστών . Ένα παράδειγμα είναι οι πληροφορίες που προσδιορίζουν τον προορισμό του πακέτου . Οι routers κάνουν χρήση αυτών των πληροφοριών ελέγχου για να καταλάβουν πώς να διαβιβάσουν το κάθε πακέτο .

Ένα πρωτόκολλο (protocol) είναι μια συμφωνία σχετικά με τα πακέτα που ανταλλάσσονται από την επικοινωνία των προγραμμάτων και τι σημαίνουν . Ένα πρωτόκολλο λέει πώς τα πακέτα είναι δομημένα για παράδειγμα, που βρίσκονται οι πληροφορίες προορισμού στο πακέτο και πόσο μεγάλο είναι ,

καθώς και το πώς οι πληροφορίες πρέπει να ερμηνευτούν . Ένα πρωτόκολλο είναι συνήθως σχεδιασμένο για να λύσει ένα συγκεκριμένο πρόβλημα με τη χρήση των δεδομένων δυνατοτήτων . Για παράδειγμα , το Hypertext Transfer Protocol (HTTP) λύνει το πρόβλημα της μεταφορά υπερκειμένου αντικείμενων μεταξύ των servers , όπου είναι αποθηκευμένα ή δημιουργούνται , και οι Web browsers τα καθιστούν ορατά και χρήσιμα για τους χρήστες . Τα πρωτόκολλα ανταλλαγής μηνυμάτων λύνουν το πρόβλημα επιτρέποντας δύο ή περισσότερους χρήστες να ανταλλάσσουν μηνύματα κειμένου σε πραγματικό χρόνο.

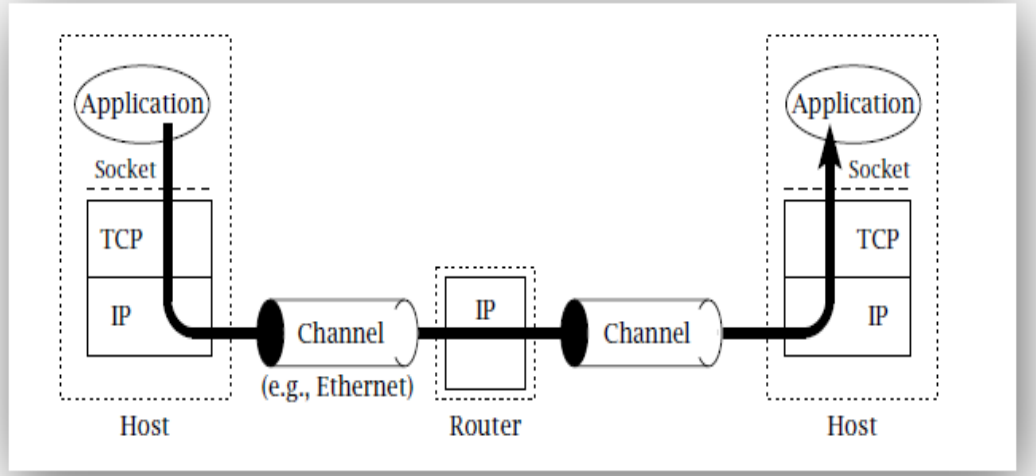
Η εφαρμογή ενός χρήσιμου δικτύου απαιτεί την επίλυση ενός μεγάλου αριθμού διαφορετικών προβλημάτων . Για την ομαλή και εύχρηστη λειτουργιά , χρησιμοποιούνται διαφορετικά πρωτόκολλα με σκοπό την επίλυση διαφόρων προβλημάτων. Τα πρωτόκολλα TCP / IP είναι μια τέτοια συλλογή από λύσεις , που μερικές φορές ονομάζεται σουίτα πρωτοκόλλων .

Τυχαίνει να είναι η σουίτα των πρωτοκόλλων που χρησιμοποιούνται στο Διαδίκτυο, αλλά μπορεί να χρησιμοποιηθεί καθώς και σε αυτόνομα ιδιωτικά δίκτυα . Στο εξής , όταν μιλάμε για το δίκτυο , εννοούμε οποιοδήποτε δίκτυο το οποίο χρησιμοποιεί τη στοίβα πρωτοκόλλων TCP / IP . Τα κύρια πρωτόκολλα της σουίτας TCP / IP είναι το Internet Protocol (IP) , το Transmission Control Protocol (TCP) και το User Datagram Protocol (UDP).

Αποδεικνύεται ότι είναι χρήσιμο να οργανωθούν τα πρωτόκολλα σε στρώματα. Το TCP / IP και σχεδόν όλες οι άλλες σουίτες του πρωτοκόλλων οργανώνονται με αυτόν τον τρόπο.

Τα προγράμματα αποκτούν πρόσβαση στις υπηρεσίες που δίνονται από UDP και TCP μέσω του Sockets API όπως φαίνεται στο σχήμα πιο κάτω με διακεκομμένη γραμμή. Το βέλος συμβολίζει την ροη δεδομένων από το πρόγραμμα μέσω της υλοποίησης TCP και IP ενός host μέσα από το δίκτυο

και πίσω στην υλοποίηση TCP και IP στη άλλη μεριά του δικτύου.



Εικόνα 2 - Δίκτυο TCP/IP

Στο TCP / IP , το κάτω στρώμα αποτελείται από τα κανάλια επικοινωνίας, για την επικοινωνία, παράδειγμα Ethernet ή dial-up συνδέσεις μέσω μόντεμ . Αυτά τα κανάλια χρησιμοποιούνται από το στρώμα δικτύου, το οποίο ασχολείται με το πρόβλημα των πακέτων προώθησης προς τον προορισμό τους (δηλαδή , ότι κάνουν οι δρομολογητές) . Το μονό στρώμα δικτύου πρωτόκολλου στη σουίτα TCP / IP είναι το Internet protocol. Το οποίο λύνει το πρόβλημα του να καταστεί η ακολουθία των διαύλων και των δρομολογητών μεταξύ οποιονδήποτε δύο host και μοιάζει με ένα μόνο κανάλι από host σε host .

Το Internet protocol παρέχει μια υπηρεσία δεδομενογράμματος (datagram service) οπού κάθε πακέτο διαχειρίζεται και παραδίνεται από το δίκτυο, ανεξαρτήτως , όπως τις επιστολές ή δέματα που αποστέλλονται μέσω του ταχυδρομείου. Για να γίνει αυτό εφικτό , κάθε πακέτο IP πρέπει να περιέχει τη διεύθυνση προορισμού του , όπως άλλωστε και κάθε πακέτο σας στο

ταχυδρομείο απευθύνεται σε κάποιον. Αν και οι περισσότερες εταιρείες διανομής πακέτων εγγυώνται την παράδοση ενός πακέτου , η IP κάνει την καλύτερη της προσπάθεια. Δηλαδή επιχειρεί να παραδώσει κάθε πακέτο , αλλά μπορεί να τα χάσει , να αναδιατάξει ή να αντιγράψει τα πακέτα κατά την μεταφορά τους μέσω του δικτύου .

Το στρώμα πάνω από την IP ονομάζεται το στρώμα μεταφοράς (transport layer). Προσφέρει μια επιλογή μεταξύ δύο πρωτόκολλων (TCP και UDP). Κάθε πρωτόκολλο βασίζεται στην υπηρεσία που παρέχεται από την IP , αλλά το κάνουν με διαφορετικούς τρόπους για να παρέχουν διαφορετικά είδη μεταφορών, που χρησιμοποιούνται από τα πρωτόκολλα εφαρμογής με διαφορετικές ανάγκες. Το TCP και UDP έχουν μία κοινή λειτουργία, την διευθυνσιοδότηση.

Υπενθυμίζουμε ότι η IP παραδίδει τα πακέτα στους hosts, σαφώς , απαιτείται μια λεπτομερέστερη αναλυτικότερη αντιμετώπιση για να πάρει ένα πακέτο σε ένα συγκεκριμένο πρόγραμμα εφαρμογής , ίσως ένα από τα πολλά που χρησιμοποιούν στο δίκτυο στον ίδιο ξενιστή (host). Τόσο το TCP και το UDP χρησιμοποιούν διευθύνσεις, που ονομάζονται αριθμοί θύρας (port numbers), για τον εντοπισμό των προγραμμάτων στους ξενιστές (hosts) . Το TCP και UDP ονομάζονται αλλιώς πρωτόκολλα μεταφοράς άκρο σε άκρο (end to end transport protocols), επειδή μεταφέρουν δεδομένα σε όλη τη διαδρομή από το ένα πρόγραμμα στην άλλο (ενώ η IP μεταφέρει μόνο τα δεδομένα από τον ένα ξενιστή (host) στον άλλο) .

Το TCP έχει σχεδιαστεί για να εντοπίσει και να ανακάμψει από τις απώλειες , τις επαναλήψεις , και άλλα σφάλματα που μπορεί να συμβούν στο κανάλι ξενιστή προς ξενιστή (host to host) που παρέχεται από την IP. Το TCP παρέχει ένα αξιόπιστο κανάλι ροής δεδομένων (byte stream), έτσι ώστε οι αιτήσεις να μην έχουν να αντιμετωπίσουν αυτά τα προβλήματα . Είναι μια σύνδεση με πρωτόκολλο connection-oriented, δηλαδή πριν τη χρήση του για

να επικοινωνήσει μεταξύ δύο προγραμμάτων θα πρέπει πρώτα να δημιουργήσει μια σύνδεση TCP, η οποία περιλαμβάνει την ολοκλήρωση της ανταλλαγής μηνυμάτων χειραψίας (handshake messages) μεταξύ των εφαρμογών TCP επί των δύο επικοινωνούντων υπολογιστών. Η χρήση του πρωτόκολλου TCP είναι επίσης παρόμοια με πολλούς τρόπους με την λειτουργία αρχείων εισόδου / εξόδου (I/O). Στην πραγματικότητα, ένα αρχείο που είναι γραμμένο από ένα πρόγραμμα και διαβάζεται από ένα άλλο είναι η λογική στο μοντέλο της επικοινωνίας κατά τη διάρκεια μιας σύνδεσης TCP.

Η UDP, από την άλλη πλευρά, δεν προσπαθήσει να ανακάμψει από τα λάθη που βιώνουν οι IP, αλλά επεκτείνει το IP Datagram της υπηρεσίας βέλτιστης προσπάθειας, έτσι ώστε να λειτουργεί καλύτερα μεταξύ των προγραμμάτων εφαρμογής αντί μεταξύ των ξενιστών (Hosts). Έτσι, οι εφαρμογές που χρησιμοποιούν το UDP πρέπει να είναι προετοιμασμένες να αντιμετωπίσουν τις απώλειες, την αναδιάταξη, και ούτω καθεξής.

1.3 Διευθύνσεις

Όταν στέλλεται ένα γράμμα , γράφετε και την διεύθυνση του αποστολέα έτσι ώστε ο ταχυδρόμος να ξέρει που να παραδώσει το γράμμα σας. Ή ακόμη για να μιλήσετε με κάποιον στο τηλέφωνο πρέπει να δώσετε τον αριθμό τηλεφώνου στο σύστημα.

Με παρόμοιο τρόπο για να επικοινωνήσει ένα πρόγραμμα με ένα άλλο πρέπει να δώσετε κάτι στο δίκτυο για να αναγνωρίσει το άλλο πρόγραμμα. Μια διεύθυνση στο Internet, χρησιμοποιείται από την IP και έναν αριθμό θύρας, έτσι ώστε η επιπλέον διεύθυνση να ερμηνευθεί από το πρωτόκολλο μεταφοράς (TCP ή UDP).

Οι διευθύνσεις του Internet είναι δυαδικοί αριθμοί. Έρχονται σε δύο εκδόσεις, που αντιστοιχούν στο δύο εκδόσεις του πρωτοκόλλου Internet που έχουν τυποποιηθεί. Η πιο συνηθισμένη είναι η έκδοση 4 (IPv4). Η άλλη είναι η έκδοση 6 (IPv6), η οποία μόλις τώρα αρχίζει να αναπτύσσεται.

Οι διευθύνσεις IPv4 είναι μεγέθους 32 bit. Αυτό είναι αρκετό μόνο να εντοπίσει περίπου 4 δισεκατομμύρια διακριτούς προορισμούς, όμως δεν είναι πραγματικά αρκετά μεγάλη για τη σημερινό Internet. (Αυτό μπορεί να φαίνεται σαν μια τεράστια παρτίδα, αλλά λόγω του τρόπου που έχουν κατανεμηθεί, πολλές έχουν χαθεί στην πορεία. Περισσότερο από το ήμισυ του συνόλου των διευθύνσεων IPv4 έχει ήδη διατεθεί.) Για το λόγο αυτό, το IPv6 εισήχθη τελευταία. Οι διευθύνσεις IPv6 είναι μεγέθους 128bit.

1.3.1 **Γράφωντας τις διευθύνσεις IP**

Οι διευθύνσεις IPv4 είναι συμβατικά γραμμένες ως μια ομάδα τεσσάρων δεκαδικών αριθμών που χωρίζονται από τελείες (π.χ. 10.1.2.3) , αυτό ονομάζεται ο συμβολισμός διακεκομμένη τετράδας (dotted quad notation). Οι τέσσερις πρώτοι αριθμοί σε μια σειρά πριν από πρώτη τελεία αντιπροσωπεύουν τα περιεχόμενα των τεσσάρων bytes της διεύθυνσης Διαδικτύου , το καθένα είναι ένας αριθμός μεταξύ 0 και 255 .

Τα υπόλοιπα 16 bytes μιας διεύθυνσης IPv6 , από την άλλη πλευρά , κατά συνθήκη αντιπροσωπεύονται ως ομάδες δεκαεξαδικών ψηφίων , που χωρίζονται από το σύμβολο της άνω – κάτω τελείας (π.χ. , 2000 : fdb8 : 0000:0000:0001:00 ab : 853c : 39A1) . Κάθε ομάδα ψηφίων αντιπροσωπεύει 2 bytes της διεύθυνσης . Τα μηδενικά στην αρχή είναι δυνατόν να παραλείπονται , έτσι η πέμπτη και έκτη ομάδα στο προηγούμενο παράδειγμα θα μπορούσε να αποδοθεί ως μόνο : 1 : ab : . επίσης , μία αλληλουχία ομάδων που περιέχει μόνο μηδενικά μπορεί να παραλειφθεί εντελώς. Έτσι, το

παραπάνω παράδειγμα θα μπορούσε να είναι γραμμένο το 2000 : fdb8 :: 1:00 ab : 853c : 39A1 .

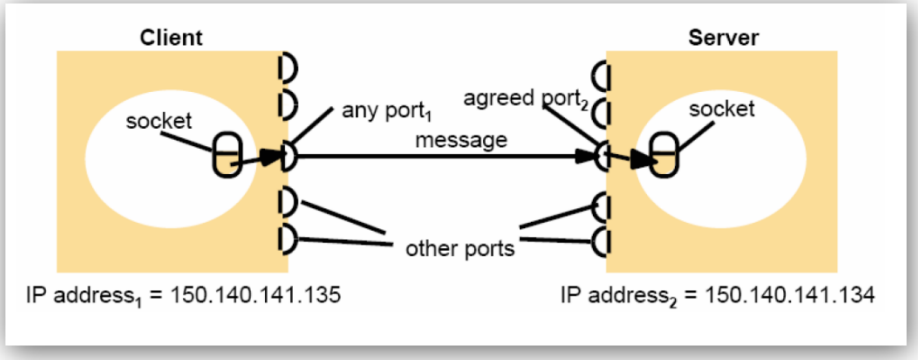
Τεχνικά , κάθε διεύθυνση Διαδικτύου αναφέρεται στη σύνδεση μεταξύ ενός ξενιστή και ένα υποκείμενο κανάλι επικοινωνίας με άλλα λόγια , μια διεπαφή δικτύου (network interface) . Ένας host μπορεί να έχει πολλές διεπαφές, δεν είναι ασυνήθιστο , για παράδειγμα , μπορεί να έχει συνδέσεις με τα ενσύρματη (Ethernet) και ασύρματα δίκτυα (Wi-Fi). Επειδή κάθε τέτοια σύνδεση με το δίκτυο ανήκει σε ένα μόνο ξενιστή , μια διεύθυνση Διαδικτύου προσδιορίζει έναν ξενιστή , καθώς και τη σύνδεσή του με το δίκτυο.

Ωστόσο , το αντίστροφο δεν είναι αληθές , γιατί ένας ενιαίος ξενιστής μπορεί να έχει πολλαπλές διεπαφές, και κάθε διεπαφή μπορεί να έχει πολλαπλές διευθύνσεις. (Στην πραγματικότητα , το ίδιο interface μπορεί να έχει τόσο IPv4 και IPv6 διευθύνσεις).

1.3.2 **Αριθμοί θύρας (port numbers)**

Είπαμε προηγουμένως ότι παίρνει δύο κομμάτια της διεύθυνσης για να μεταφέρει ένα μήνυμα σε ένα πρόγραμμα . Ο αριθμός θύρας TCP ή UDP ερμηνεύεται πάντα σε σχέση με μια διεύθυνση στο Internet . Επιστρέφοντας στην προηγούμενες παρομοιώσεις μας , ένας αριθμός θύρας αντιστοιχεί σε έναν αριθμό δωματίων σε μία δεδομένη διεύθυνση , ας πούμε , σε ένα μεγάλο κτίριο. Η ταχυδρομική υπηρεσία χρησιμοποιεί τη διεύθυνση του δρόμου για να πάρει την επιστολή στο κεντρικό γραμματοκιβώτιο. Όποιος αδειάζει το γραμματοκιβώτιο είναι τότε υπεύθυνος για να πάρει την επιστολή προς το σωστό δωμάτιο εντός του κτιρίου. Ή αν εξετάσουμε μια εταιρεία με ένα εσωτερικό τηλεφωνικό σύστημα, για να μιλήσετε σε ένα άτομο στην εταιρεία , πρέπει πρώτα να καλέσετε τον κύριο αριθμό τηλεφώνου της εταιρείας για να συνδεθείτε στο εσωτερικό τηλεφωνικό σύστημα και στη συνέχεια να καλέσετε την επέκταση του συγκεκριμένου τηλεφώνου του

ατόμου με το οποίο θέλετε να μιλήσετε . Σε αυτές τις παρομοιώσεις , η διεύθυνση Διαδικτύου (IP) είναι ο αριθμός δρόμου ή ο κύριος αριθμό της εταιρείας , ενώ ο αριθμός θύρας (port number) αντιστοιχεί στον αριθμό του δωματίου ή την επέκταση στο τηλέφωνο . Οι αριθμοί θύρας είναι οι ίδιου δυαδικού μεγεθους,16 bit και στις δύο διευθύνσεις IPv4 και IPv6. Έτσι , ο καθένας είναι σε κλίμακα 1 έως 65.535 (το 0 είναι δεσμευμένο) .



Εικόνα 3 - Παράδειγμα μιας σύνδεσης

1.4 Σύστημα Ονομάτων Περιοχών (DNS)

Το Domain Name System ή DNS (Σύστημα Ονομάτων Τομέων ή Χώρων ή Περιοχών) είναι ένα ιεραρχικό σύστημα ονοματοδοσίας για δίκτυα υπολογιστών, που χρησιμοποιούν το πρωτόκολλο IP. Το σύστημα DNS μπορεί και αντιστοιχίζει ονόματα με διευθύνσεις IP ή άλλα ονόματα στο Διαδίκτυο ή κάποιο άλλο δίκτυο.

Ο χώρος ονομάτων τομέων(Domain Name Space) του DNS είναι δομημένος ιεραρχικά σε δενδρική δομή, με τα ονόματα να φέρουν πληροφορία που αντανακλά τη θέση τους στη δομή αυτή. Ο χώρος ονομάτων DNS ενός ιδιωτικού δικτύου μπορεί να διαφέρει με τον χώρο ονομάτων DNS του Διαδικτύου ή κάποιου άλλου διαδικτύου. Η αντιστοίχιση ονομάτων με αριθμητικές διευθύνσεις προέκυψε επειδή οι αριθμητικές διευθύνσεις IP δεν

είναι εύχρηστες από τους ανθρώπους. Οι άνθρωποι αποδίδουν μεγαλύτερη σημασία και θυμούνται ευκολότερα τα ονόματα.

1.4.1 **Δομή DNS**

Οι χώροι ή περιοχές ή τομείς (domains) χωρίζονται σε επίπεδα, και κάθε επίπεδο συχνά περιέχει κατώτερα επίπεδα, για παράδειγμα ένας τομέας πρώτου επιπέδου μπορεί να περιέχει ιεραρχικά τομείς δεύτερου επιπέδου κτλ. Η αλλαγή επιπέδου των ονομάτων χώρου είναι πολλές φορές ισοδύναμη με αλλαγή ζώνης DNS (DNS zone). Χρησιμοποιώντας την ορολογία που χρησιμοποιείται στην δενδρική δομή η ζώνη DNS είναι ένας κόμβος και ένα όνομα χώρου είναι ένα φύλλο. Όλες οι ζώνες DNS είναι και ονόματα χώρου αλλά το αντίστροφο δεν ισχύει πάντα. Στην πράξη οι ζώνες DNS είναι τα φυσικά αρχεία που βρίσκονται σε εξυπηρετητές DNS και περιέχουν τις αντιστοιχίσεις ονομάτων και διευθύνσεων ή άλλων ονομάτων ως εγγραφές DNS (DNS records ή resource records). Δηλαδή οι ζώνες DNS είναι απλές βάσεις δεδομένων και οι εγγραφές DNS είναι τα δεδομένα.

Οι ζώνες DNS συνήθως σημαίνουν την αλλαγή διαχείρισης μιας περιοχής/χώρου και περιέχουν εγγραφές DNS (με κατεύθυνση από το όνομα) μόνο με το όνομα χώρου ή τομείς του. Όταν κάποιος κατοχυρώνει ένα όνομα χώρου στο σύστημα DNS στη ουσία παίρνει τον έλεγχο της ζώνης DNS αυτού του ονόματος χώρου.

Το Σύστημα DNS βασίζεται σε μια διανεμημένη βάση δεδομένων η οποία «τρέχει» στους εξυπηρετητές (servers) του συστήματος και αποτελείται από ζώνες DNS οργανωμένες σε μια δενδρική δομή. Οι εξυπηρετητές DNS χωρίζονται στους αρχικούς (root) εξυπηρετητές, τους εξουσιοδοτημένους (authoritative) εξυπηρετητές, και τους αποθηκευτικούς (caching) εξυπηρετητές. Οι εξουσιοδοτημένοι εξυπηρετητές DNS χωρίζονται σε

πρωτεύοντες και εναλλακτικούς (masters and slaves). Συνήθως κάποιος από τους πρωτεύοντες εξουσιοδοτημένους εξυπηρετητές ενός ονόματος χώρου είναι ο πρωταρχικός. Σε αυτόν γίνονται συνήθως οι αλλαγές.

Πελάτες των υπηρεσιών που παρέχουν οι εξυπηρετητές DNS είναι οι λύτες DNS (DNS resolvers). Οι λύτες είναι λογισμικό που χρησιμοποιείται από έναν χρήστη ή κάποιο πρόγραμμα που ζητά τις υπηρεσίες DNS. Οι λύτες διαβάζουν τα ονόματα του DNS από δεξιά προς τα αριστερά. Κάθε τελεία δείχνει την αρχή ενός υποσυνόλου και το σύνολο που περιλαμβάνει όλα τα σύνολα είναι η πιο δεξιά τελεία που ονομάζεται ρίζα και συνήθως παραλείπεται.

Π.χ. όταν γράφουμε το όνομα "DNS.example.wikipedia.www.el.ipduh.com" εννοούμε "DNS.example.wikipedia.www.el.ipduh.com." . Η τελική τελεία είναι το σύνολο που περιλαμβάνει όλο το σύστημα και το υποσύνολο που ονομάζεται "com.". Το σύνολο "com." περιλαμβάνει το σύνολο "ipduh.com.", το σύνολο "ipduh.com." περιλαμβάνει το "el.ipduh.com." Το σύνολο "el.ipduh.com." περιλαμβάνει το σύνολο "www.el.ipduh.com."

Οι άνθρωποι διαβάζουν τα ονόματα DNS από αριστερά προς τα δεξιά και πάντα παραλείπουν την τελευταία τελεία. Στα ονόματα DNS επιτρέπεται η χρήση αλφαριθμητικών στοιχείων παυλών και τελειών. Συνεχόμενες παύλες και συνεχόμενες τελείες απαγορεύονται. Στα ονόματα χώρου τα κεφαλαία γράμματα είναι ισοδύναμα με τα μικρά γράμματα. Π.χ example.net και exAmpLE.nET είναι το ίδιο όνομα.

Η σχέση μεταξύ ονομάτων και διευθύνσεων IP δεν είναι 1 προς 1. Δηλαδή σε ένα όνομα μπορεί να αντιστοιχούν πολλές διευθύνσεις IP και σε μια διεύθυνση πολλά ονόματα.

1.4.2 **Σημασία του DNS**

Το σύστημα DNS προέκυψε επειδή στους ανθρώπους ονόματα σημαίνουν περισσότερα από αριθμητικές διευθύνσεις αλλά στην συνέχεια το σύστημα DNS απέκτησε και άλλες χρήσεις εξίσου σημαντικές.

Το DNS επιτρέπει την ανεύρεση ενός εξυπηρετητή (server) ή μιας υπηρεσίας σε έναν εξυπηρετητή χρησιμοποιώντας ένα όνομα. Ένας εξυπηρετητής μπορεί να προσφέρει ταυτόχρονα περισσότερες από μια υπηρεσίες, σύμφωνα με διάφορα πρωτόκολλα, όπως το HTTP, το FTP, το POP, το IMAP και το SMTP, δίνοντας τη δυνατότητα στο χρήστη να συνδεθεί σε μια ιστοσελίδα (HTTP), σε μια αποθήκη αρχείων (FTP), ή να λάβει email (POP ή IMAP). Για ένα χρήστη είναι ευκολότερο να θυμάται το όνομα της ιστοσελίδας www.google.gr παρά το χ.ψ.ω.ζ:80 (ο συνδυασμός διεύθυνσης IP και θύρας TCP στην οποία βρίσκεται ο εξυπηρετητής HTTP του www.google.gr).

Επίσης το DNS χρησιμοποιείται για να αντιστοιχίσει διευθύνσεις IP με ονόματα. Έτσι ο διαχειριστής ενός δικτύου μπορεί να χρησιμοποιήσει ονόματα για να επικοινωνήσει ή να απλώς να θυμάται ονόματα μηχανημάτων, τοποθεσίες, ονόματα χώρου, και ότι άλλο σκεφτεί. Τα ονόματα των διευθύνσεων IP λειτουργούν και κατά κάποιον τρόπο σαν εγγυήσεις μιας και μόνο οι διαχειριστές των δικτύων - κάτοχοι των διευθύνσεων μπορούν να τα αλλάξουν. Στην λειτουργία του ηλεκτρονικού ταχυδρομείου το όνομα της διεύθυνσης IP του εξυπηρετητή ηλεκτρονικού ταχυδρομείου (Mail Server) θεωρείται απόδειξη του ότι είναι αυτός που λέει.

Το σύστημα DNS δίνει, τέλος, τη δυνατότητα αντιστοίχισης μεταξύ ονομάτων, καθώς και τη δυνατότητα αντιστοίχισης ενός ονόματος σε πολλαπλές διευθύνσεις IP (round robin DNS και IP sorting), πράγμα που βοηθά στη διαμοίραση του φόρτου μιας δικτυακής υπηρεσίας σε περισσότερους του ενός εξυπηρετητές ή την κατεύθυνση των πελατών δικτυακών υπηρεσιών σε γεωγραφικά κοντινότερους εξυπηρετητές.

Στο σύστημα DNS είναι δυνατή η αντιστοίχιση άπειρων ονομάτων σε μία διεύθυνση IP ή μια ομάδα διευθύνσεων IP. Αυτό διευκολύνει λογιστικά την διαχείριση εξυπηρετητών δικτυακών υπηρεσιών και βοηθά στην οικονομία διευθύνσεων IP.

1.5 Συμπεράσματα

Περιληπτικά είδαμε ότι μία διεύθυνση IP (Ip address - Internet Protocol address), είναι ένας μοναδικός αριθμός που χρησιμοποιείται από συσκευές για τη μεταξύ τους αναγνώριση και συνεννόηση σε ένα δίκτυο υπολογιστών και ότι υπάρχουν 2 εκδόσεις αυτή την στιγμή στο διαδίκτυο. Ακόμη ότι το Domain Name Service (DNS), δίνει τη δυνατότητα να αντιστοιχηθούν ονόματα υπολογιστών (hostnames) σε μια διεύθυνση IP. Με αυτό τον τρόπο, οι άνθρωποι μπορούν εύκολα να θυμούνται ένα όνομα και όχι μια σειρά αριθμών. Πως γίνεται όμως η δημιουργία της επικοινωνίας αυτής μεταξύ client-server θα το αναλύσουμε στο επόμενο κεφάλαιο.

2ο ΚΕΦΑΛΑΙΟ

CLIENT-SERVER

2.1 Εισαγωγή

Στην ταχυδρομική και τηλεφωνική μας παρομοίωση , κάθε επικοινωνία ξεκινά από το ένα μέρος , όταν ο ένας στέλνει ένα e-mail ή να κάνει το τηλεφώνημα , ενώ το άλλο μέρος ανταποκρίνεται στην απάντηση επικοινωνίας με την αποστολή e-mail ή να σηκώσετε το τηλέφωνο και να μιλήσετε . Η επικοινωνία μέσω του διαδικτύου είναι παρόμοια. Οι όροι client και server αναφέρονται σε αυτούς τους ρόλους. Το πρόγραμμα του client ξεκινούν την επικοινωνία , ενώ το πρόγραμμα του server περιμένει, παθητικά και στη συνέχεια απαντά στους πελάτες που να επικοινωνήσουν . Μαζί , ο πελάτης (client) και ο διακομιστής (server) συνθέτουν την εφαρμογή . Οι όροι client και server είναι περιγραφικά της τυπικής κατάστασης κατά την οποία ο διακομιστής κάνει μια ιδιαίτερη λειτουργία προς παράδειγμα , μια υπηρεσία βάσης δεδομένων όπου επιτρέπει σε κάθε πελάτη να μπορεί να επικοινωνεί με αυτήν .

Το κατά πόσον ένα πρόγραμμα ενεργεί ως client ή server καθορίζει τη γενική μορφή της χρήσης του Sockets API για να δημιουργήσουν δίαυλο επικοινωνίας με ομολόγους του (Ο πελάτης είναι ο ομότιμους του server και αντίστροφα). Επιπλέον , η διάκριση πελάτη-διακομιστή είναι σημαντικό, διότι ο πελάτης πρέπει να γνωρίζει τη διεύθυνση του διακομιστή και τον αριθμό θύρας αρχικά , αλλά όχι το αντίστροφο . Με το Sockets API , ο διακομιστής μπορεί , αν χρειαστεί , να μάθει πληροφορίες για τη διεύθυνση

του πελάτη , όταν λαμβάνει την αρχική επικοινωνία από τον πελάτη . Αυτό είναι ανάλογο με μια τηλεφωνική κλήση, όταν σας καλεί ένα πρόσωπο που δεν χρειάζεται να γνωρίζετε τον αριθμό του τηλεφώνου του. Όπως και με μια τηλεφωνική κλήση , μόλις πραγματοποιηθεί η σύνδεση , η διάκριση μεταξύ server και client εξαφανίζεται .

2.2 Ορισμός

Στην επιστήμη των υπολογιστών το μοντέλο αρχιτεκτονικής λογισμικού πελάτη-διακομιστή αποτελεί μία συνήθη μέθοδο ανάπτυξης λογισμικού στην οποία ο πελάτης (ένα τμήμα λογισμικού) ζητά κάτι (π.χ. έναν πόρο, τα αποτελέσματα ενός υπολογισμού) και ένα άλλο τμήμα λογισμικού, ο διακομιστής (ή εξυπηρετητής), του το επιστρέφει. Κάθε διακομιστής μπορεί να εξυπηρετεί πολλαπλούς πελάτες.

Όταν ένας υπολογιστής εκτελεί κυρίως τέτοια προγράμματα εξυπηρετητές συνεχόμενα, 24 ώρες την ημέρα, τότε μπορούμε να αναφερθούμε σε όλον τον υπολογιστή ως εξυπηρετητή, αφού αυτή είναι η κύρια λειτουργία του. Παρομοίως, ως πελάτη μπορούμε να θεωρήσουμε είτε κάποιο λογισμικό που επικοινωνεί και υποβάλει αιτήματα στον εξυπηρετητή, είτε σε όλο τον υπολογιστή όταν ο εξυπηρετητής είναι άλλος υπολογιστής και οι 2 υπολογιστές είναι συνδεδεμένοι σε ένα δίκτυο.

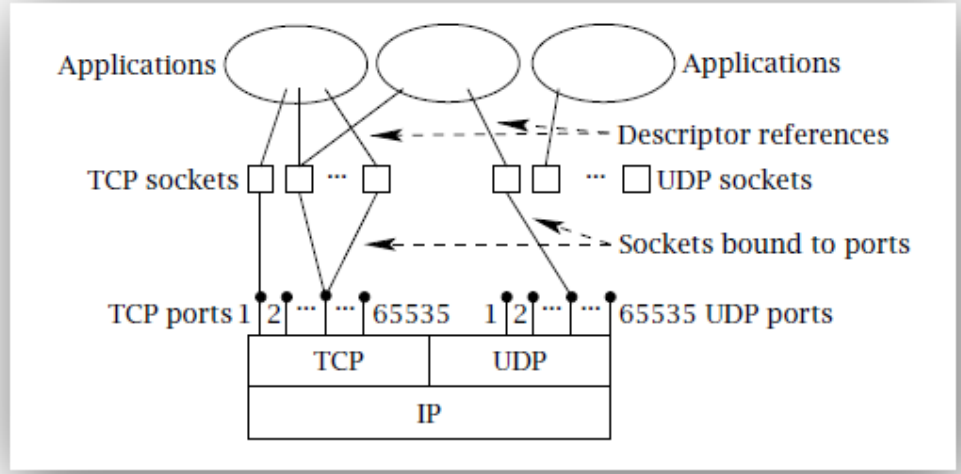
Εικόνα 4 - Οι εξυπηρετητές του ιδρύματος
Wikimedia

2.3 Διαδιεργασιακή επικοινωνία(IPC) και υποδοχές δικτύου (Sockets)

Διαδιεργασιακή επικοινωνία (InterProcess Communication, IPC) ονομάζεται στην πληροφορική ένα σύνολο μηχανισμών που παρέχουν τα λειτουργικά συστήματα των ηλεκτρονικών υπολογιστών, οι οποίοι διευκολύνουν την ανταλλαγή δεδομένων και τον συγχρονισμό μεταξύ ταυτοχρόνως εκτελούμενων διεργασιών μέσω δομών δεδομένων του πυρήνα. Τέτοιοι μηχανισμοί είναι απαραίτητοι στα μοντέρνα λειτουργικά συστήματα όπου, χάρη στον μηχανισμό της εικονικής μνήμης, κάθε διεργασία έχει τον δικό της ιδιωτικό χώρο εικονικών διευθύνσεων στον οποίον έχει πρόσβαση μόνο αυτή και ο πυρήνας. Προκειμένου να υπάρχει μία στοιχειώδης προστασία μνήμης

μεταξύ διαφορετικών διεργασιών, καμία διεργασία δεν έχει δικαίωμα ανάγνωσης ή εγγραφής στον χώρο διευθύνσεων των υπολοίπων. Αν λοιπόν χρειάζεται δύο διαφορετικές διεργασίες να επικοινωνήσουν μεταξύ τους ή να ανταλλάξουν δεδομένα, αυτό μπορεί να γίνει μόνο μέσω του συστήματος αρχείων (π.χ. μία διεργασία να γράψει ένα αρχείο και μία άλλη να το διαβάσει) ή μέσω μίας μεθόδου διαδιεργασιακής επικοινωνίας. Με το προγραμματιστικό μοντέλο των υποδοχών (Sockets) οι διεργασίες οι οποίες επικοινωνούν μπορούν να εκτελούνται σε διαφορετικούς υπολογιστές που διασυνδέονται μέσω ενός δικτύου.

Σε πολλές περιπτώσεις ένα εκτελούμενο πρόγραμμα (η μητρική ή γονική διεργασία) δημιουργεί δευτερεύουσες (θυγατρικές) διεργασίες ώστε να εκμεταλλευτεί πιθανά οφέλη από τον ταυτοχρονισμό. Με αυτόν τον τρόπο, σε ένα παράλληλο σύστημα οι υπολογισμοί που απαιτούνται από μία εφαρμογή μπορούν να κατανεμηθούν σε πολλαπλούς επεξεργαστές με τον καθένα να εκτελεί διαφορετική διεργασία, ενώ σε ένα σειριακό σύστημα αν μία διεργασία ανασταλεί (π.χ. σε μία κλήση συστήματος) καθώς περιμένει την απελευθέρωση ενός πόρου (π.χ. πρόσβαση στον σκληρό δίσκο) ή μία είσοδο από τον χρήστη), κάποια άλλη διεργασία μπορεί να συνεχίσει τους υπολογισμούς. Είναι φανερό επομένως ότι η διαδιεργασιακή επικοινωνία δεν είναι απαραίτητη μόνο για την ανταλλαγή δεδομένων μεταξύ ανεξάρτητων διεργασιών, αλλά και για τον συντονισμό στενά συνεργαζόμενων διεργασιών οι οποίες εκτελούνται παράλληλα, σε συστήματα πολλαπλών επεξεργαστών, ή ψευδοπαράλληλα, δηλαδή με ταχύτατη και διαφανή εναλλαγή πολλαπλών ταυτοχρόνως εκτελούμενων διεργασιών στον μοναδικό επεξεργαστή.



Εικόνα 5 – Sockets, protocols και ports

Η διαδιεργασιακή επικοινωνία στα λειτουργικά συστήματα τύπου UNIX καθορίζεται από το πρότυπο POSIX.

2.3.1 **POSIX**

POSIX (Portable Operating System Interface for Unix) στην πληροφορική καλείται το όνομα μιας σειράς πρότυπων που καθορίζουν τις βασικές υπηρεσίες που παρέχει ένα λειτουργικό σύστημα, και κυρίως το API των διαθέσιμων κλήσεων συστήματος. Δημιουργήθηκε με στόχο την εξασφάλιση της συμβατότητας μεταξύ των ποικίλλων λειτουργικών συστημάτων UNIX, έτσι ώστε τα προγράμματα που τρέχουν σε έναν υπολογιστή με UNIX να μπορούν να μεταγλωττιστούν σε οποιονδήποτε άλλον υπολογιστή με UNIX χωρίς να χρειάζεται να τροποποιηθεί ο πηγαίος κώδικας (ή μέρος αυτού). Η συμμόρφωση με το POSIX αποτελεί προϋπόθεση για να μπορεί ένα λειτουργικό σύστημα να ονομάζεται UNIX.

2.3.2 **Σήματα (signals)**

Μία περιορισμένη μορφή διαδιεργασιακής επικοινωνίας είναι τα σήματα (signals), διακοπές λογισμικού που δρουν ως σινιάλα και μπορούν να αποστέλλονται σε μία διεργασία από κάποια άλλη ή από τον πυρήνα, αναγκάζοντάς την να τα χειριστεί ασύγχρονα μόλις τα λάβει και ακολούθως να επιστρέφει στην κανονική ροή εκτέλεσης. Κάθε σήμα διακρίνεται από έναν ακέραιο με τον οποίο είναι συσχετισμένο κάποιο συμβολικό όνομα (SIGxxxx). Το μοντέλο είναι το εξής: ένα πρόγραμμα στον κώδικά του δηλώνει ότι μία συνάρτηση είναι χειριστής ενός συγκεκριμένου σήματος (εγκατάσταση χειριστή). Όταν ληφθεί το σήμα αυτό κατά την εκτέλεση της αντίστοιχης διεργασίας, αυτομάτως η τελευταία διακόπτει ότι έκανε (εντολή Α) και εκτελεί τον χειριστή. Μόλις ο χειριστής επιστρέψει ο έλεγχος δίνεται ξανά στην εντολή Α. Ένας χειριστής εγκαθίσταται με την κλήση συστήματος sigaction() (αν δίνεται η ειδική τιμή SIG_IGN ως όρισμα στη sigaction() αντί για όνομα συνάρτησης, τότε το αντίστοιχο σήμα αγνοείται από τη διεργασία), ενώ με τις κλήσεις kill() ή sigqueue() το τρέχον πρόγραμμα αποστέλλει ένα συγκεκριμένο σήμα σε μία συγκεκριμένη διεργασία (η διαφορά τους είναι ότι η sigqueue(), μαζί με το σήμα, αποστέλλει και έναν επιπλέον ακέραιο με δεδομένα του προγραμματιστή). Σχεδόν για κάθε σήμα υπάρχει κάποιος προεπιλεγμένος χειριστής ο οποίος παρέχεται από το πρότυπο POSIX και δεν χρειάζεται να γραφεί ή να εγκατασταθεί χειροκίνητα (συνήθως επιφέρει τον τερματισμό της διεργασίας ή αγνοεί τελείως το σήμα)· για κάποια συγκεκριμένα σήματα μάλιστα δεν μπορεί να υποσκελιστεί από χειριστή του προγραμματιστή.

2.3.3 **Σωληνώσεις (Pipes)**

Η παλαιότερη μέθοδος διαδιεργασιακής επικοινωνίας στο Unix είναι οι σωληνώσεις (pipes). Πρόκειται για δομές δεδομένων του πυρήνα που επιτρέπουν σε δύο συγγενείς διεργασίες να ανταλλάσουν αμφίδρομα

δεδομένα. Εμφανίζονται στον χώρο του χρήστη ως ζεύγη περιγραφέων αρχείων, χωρίς να αντιστοιχούν σε πραγματικά αρχεία, όπου ο ένας περιγραφέας είναι άκρο εγγραφής και ο άλλος άκρο ανάγνωσης. Οι συνήθεις κλήσεις συστήματος για Είσοδο / Έξοδο σε αρχεία (read(), write(), close()) βρίσκουν εφαρμογή και εδώ. Σε κάθε σωλήνωση μπορούν να αντιστοιχούν πολλαπλά ζεύγη περιγραφέων σε διαφορετικές διεργασίες, υλοποιώντας έτσι τη διαδιεργασιακή επικοινωνία. Αυτή η πολλαπλότητα γίνεται εφικτή μέσω της κατασκευής νέων διεργασιών (μοντέλο fork(), όπου η θυγατρική κληρονομεί όλους τους ανοικτούς περιγραφείς της γονικής διεργασίας). Είναι ευθύνη του προγραμματιστή να κλείσει τα άκρα ανάγνωσης ή εγγραφής στις αντίστοιχες διεργασίες ώστε να επιτύχει την επιθυμητή συμπεριφορά (π.χ. μία διεργασία να γράφει και η γονική της να διαβάζει τη σωλήνωση). Το μειονέκτημα είναι ότι με σωληνώσεις μόνο συγγενείς διεργασίες μπορούν να επικοινωνήσουν (γονική με θυγατρική ή αδελφές διεργασίες μεταξύ τους). Μία σωλήνωση δημιουργείται με την κλήση συστήματος pipe(). Η ανάγνωση από σωλήνωση με όλα τα άκρα εγγραφής κλειστά, είτε διαβάζει όλα τα δεδομένα που τοποθετήθηκαν στη σωλήνωση προτού κλείσει κάποιο άκρο εγγραφής και δεν αναγνώστηκαν ακόμη, είτε (αν δεν ισχύει αυτή η περίπτωση) η read() επιστρέφει αμέσως 0. Η εγγραφή σε σωληνώσεις με όλα τα άκρα ανάγνωσης κλειστά οδηγεί στην αποστολή του σήματος SIGPIPE από τον πυρήνα στην καλούσα διεργασία και στην αποτυχία της κλήσης write(). Σε σχέση με τις σωληνώσεις ενδιαφέρον παρουσιάζει η κλήση συστήματος dup(), η οποία δέχεται ως όρισμα έναν περιγραφέα αρχείου και τον κλωνοποιεί αναθέτοντας στο αντίγραφο τον πρώτο ελεύθερο αναγνωριστικό αριθμό (έτσι π.χ. μπορεί να αντικατασταθεί με κάποια σωλήνωση η τυπική είσοδος ή η τυπική έξοδος μίας διεργασίας).

Ακολουθεί ένα παράδειγμα χειρισμού σωληνώσεων σε γλώσσα προγραμματισμού C:

```
#include <stdio.h>
```

```
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#define MAXLINE 512
int main(void) {
  int n, fd[2];
  pid_t pid;
  char line[MAXLINE];
    if (pipe(fd) < 0)
   {
   perror("creating pipe");
   exit(1);
   } /* Αποτυχία κατασκευής της σωλήνωσης */
   if ((pid = fork()) < 0) {
   perror("cannot fork");
    exit(1);
   } /* Αποτυχία κατασκευής της θυγατρικής διεργασίας */
   else if (pid > 0) /* γονική διεργασία */
   {
   close(fd[0]); /* κλείσιμο του άκρου ανάγνωσης */
   write(fd[1], "message through pipe\n", 21);
   /* κλήση εγγραφής σε αρχείο - εδώ χρησιμοποιείται για εγγραφή στο ανοιχτό
   άκρο της σωλήνωσης */ close(fd[1]);
   }
   else { /* θυγατρική διεργασία */
   close(fd[1]); /* κλείσιμο του άκρου εγγραφής */
   n = read(fd[0], line, MAXLINE); /* κλήση ανάγνωσης από αρχείο - εδώ
   χρησιμοποιείται για ανάγνωση από το ανοιχτό άκρο της σωλήνωσης */
   write(STDOUT_FILENO, line, n);
   close(fd[0]);
   }
exit(0);
}
```

Εναλλακτικά μπορούν να χρησιμοποιηθούν κατονομασμένες σωληνώσεις (named pipes ή FIFOs). Αυτές κατά τη δημιουργία τους αντιστοιχίζονται σε οντότητες του συστήματος αρχείων, εικονικά αρχεία, ώστε να μπορούν να προσπελαστούν από μη συγγενείς διεργασίες που γνωρίζουν το μονοπάτι του εικονικού αρχείου. Ο χειρισμός τους γίνεται όπως των κοινών αρχείων, οπότε κάθε διεργασία για κάθε FIFO διατηρεί μόνο έναν περιγραφέα αρχείου ο οποίος επιστρέφεται από μία κοινή κλήση open(). Το μειονέκτημα είναι ότι ως αποτέλεσμα μία κατονομασμένη σωλήνωση λειτουργεί μονόδρομα, με κάθε διεργασία να μπορεί μόνο να γράφει σε αυτήν ή μόνο να διαβάζει από αυτήν, όπως συμβαίνει πάντα στο POSIX με τα αρχεία. Ένα FIFO δημιουργείται με την κλήση συστήματος mkfifo() και η διεργασία που το

δημιούργησε πρέπει, αφού το κλείσει με την close(), να το διαγράψει από το σύστημα αρχείων με την κλήση συστήματος unlink(). Συνήθως πριν από την κλήση της close() η διεργασία αυτή διαβάζει δεδομένα από το FIFO, μέσω της κλήσης read(), τα οποία έχει γράψει εκεί κάποια άλλη διεργασία που γνώριζε το μονοπάτι του FIFO στο σύστημα αρχείων και το έχει ανοίξει για εγγραφή. Φυσιολογικά, κάθε κλήση open() για ανάγνωση μίας κατονομασμένης σωλήνωσης μπλοκάρει την καλούσα διεργασία μέχρι η σωλήνωση να ανοιχτεί από κάποια άλλη διεργασία για εγγραφή (και αντίστροφα). Αυτή η συμπεριφορά μπορεί να τροποποιηθεί με κατάλληλα ορίσματα (μη ανασταλτική open()) και σε αυτήν την περίπτωση το άνοιγμα για εγγραφή σε FIFO που δεν έχει ανοιχτεί για ανάγνωση αποτυγχάνει, ενώ το άνοιγμα για ανάγνωση σε FIFO που δεν έχει ανοιχτεί για εγγραφή επιστρέφει αμέσως. Οι αναγνώσεις από το FIFO διατηρούν το μέγεθος των αιτήσεων εγγραφής που προηγήθηκαν. Πολλές διεργασίες μπορούν ταυτόχρονα να προσπελαύνουν το ίδιο αρχείο ή FIFO, πιθανώς για να ανταλλάξουν δεδομένα, με αποτέλεσμα να εμφανίζονται συνθήκες συναγωνισμού. Το πρόβλημα επιλύεται με τα κλειδώματα αρχείων, δηλαδή εξειδικευμένα mutex που παρέχει ο πυρήνας για αμοιβαίο αποκλεισμό. Τα κλειδώματα είναι είτε κοινόχρηστα είτε αποκλειστικά (με πολλαπλές διεργασίες να μπορούν να κατέχουν ταυτόχρονα ένα κοινόχρηστο κλείδωμα για το ίδιο αρχείο αλλά όχι ένα αποκλειστικό κλείδωμα, ή παράλληλα ένα κοινόχρηστο κι ένα αποκλειστικό κλείδωμα) και αφορούν ένα αρχείο και όχι έναν περιγραφέα αρχείου: αν ο τελευταίος κλωνοποιηθεί, με μία κλήση dup() ή fork(), δεν υπάρχουν πλέον δύο κλειδώματα αλλά δύο αναφορές στο ίδιο κλείδωμα. Τα κλειδώματα αρχείων είναι προσπελάσιμα μέσω της εξειδικευμένης κλήσης συστήματος flock() ή στο πλαίσιο της fcntl(), μίας γενικής χρήσης κλήσης χειρισμού αρχείων. Όταν μία διεργασία ανοίγει ένα αρχείο για ανάγνωση ζητά ένα κοινόχρηστο κλείδωμα, ενώ όταν το ανοίγει για εγγραφή ζητά ένα αποκλειστικό κλείδωμα. Στην τελευταία περίπτωση, αν το κλείδωμα του αρχείου το κατέχει κάποια άλλη διεργασία τότε η καλούσα μπλοκάρει μέχρι το κλείδωμα να ελευθερωθεί.

2.4 Συμπεράσματα

Η επικοινωνία διαφορετικών διεργασιών σε διαφορετικούς υπολογιστές σε ένα δίκτυο επιτυγχάνεται με την χρησιμοποίηση των sockets. Η υλοποίηση της επικοινωνίας αυτής γίνεται με σήματα και σωληνώσεις. Εμείς θα ασχοληθούμε με την υλοποίηση προγράμματος client-server με sockets στην γλώσσα προγραμματισμού C και στα λειτουργικά συστήματα UNIX. Τι είναι

η γλώσσα προγραμματισμού C και τι είναι λειτουργικό σύστημα UNIX θα ασχοληθούμε στο αμέσως επόμενο κεφάλαιο.

3 ο ΚΕΦΑΛΑΙΟ

UNIX ΚΑΙ C

3.1 Εισαγωγή

Λειτουργικό σύστημα ή ΛΣ (Operating System ή OS) ονομάζεται στην επιστήμη της πληροφορικής το λογισμικό του υπολογιστή που είναι υπεύθυνο για τη διαχείριση και τον συντονισμό των εργασιών, καθώς και την κατανομή των διαθέσιμων πόρων. Το λειτουργικό σύστημα παρέχει ένα θεμέλιο, ένα μεσολαβητικό επίπεδο λογικής διασύνδεσης μεταξύ λογισμικού και υλικού, διαμέσου του οποίου οι εφαρμογές αντιλαμβάνονται εμμέσως τον υπολογιστή. Μια από τις κεντρικές αρμοδιότητες του λειτουργικού συστήματος είναι η διαχείριση του υλικού, απαλλάσσοντας έτσι το λογισμικό του χρήστη από τον άμεσο και επίπονο χειρισμό του υπολογιστή και καθιστώντας ευκολότερο τον προγραμματισμό τους.

Εικόνα 6 - Τυπική θέση του λειτουργικού συστήματος σε ένα υπολογιστικό σύστημα

Στις μέρες μας, τα δημοφιλέστερα λειτουργικά συστήματα στους μικροϋπολογιστές, (συμπεριλαμβανομένων των προσωπικών υπολογιστών), έχουν διαμορφωθεί σε δύο μεγάλες οικογένειες: αυτή των Unix-συμβατών και την οικογένεια των Microsoft Windows. Οι κεντρικοί υπολογιστές και τα ενσωματωμένα συστήματα χρησιμοποιούν μια ποικιλία άλλων λειτουργικών συστημάτων, τα περισσότερα από τα οποία δεν έχουν άμεση συγγένεια με τα Windows ή με το Unix.

3.2 Unix

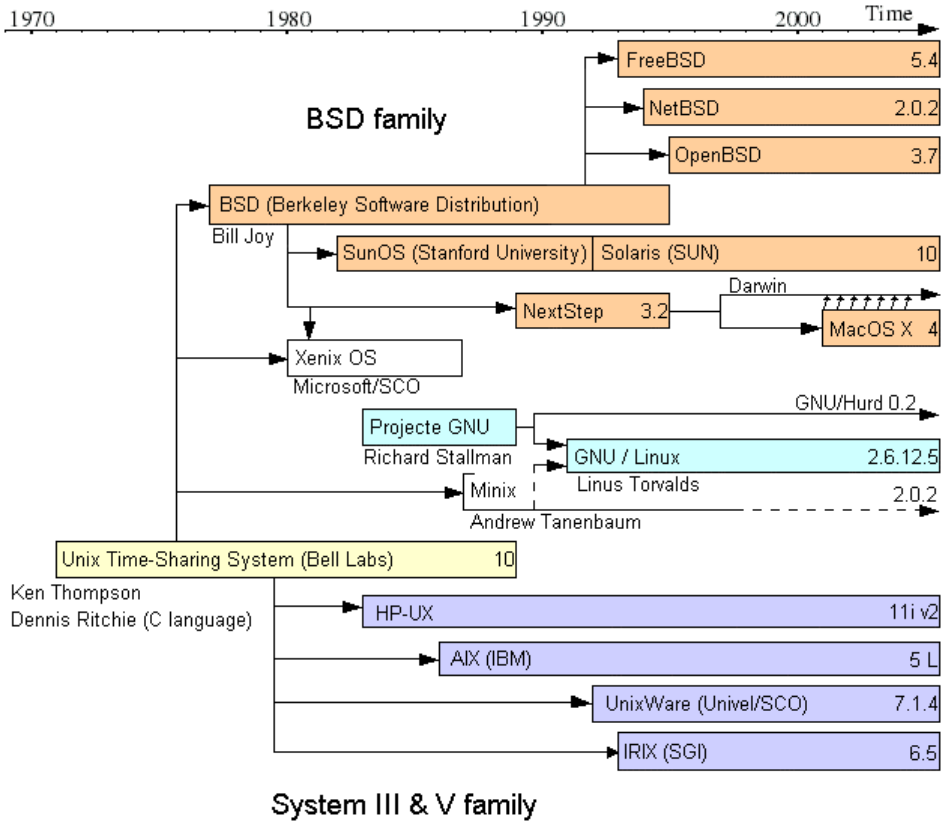
Το Unix ή UNIX είναι λειτουργικό σύστημα Ηλεκτρονικών Υπολογιστών, το οποίο αναπτύχθηκε κατά τις δεκαετίες του 1960 και του 1970 από ομάδα εργαζομένων των εργαστηρίων Μπελ (Bell Labs) της εταιρείας AT&T, στην οποία συμμετείχαν, μεταξύ άλλων, οι Κεν Τόμσον (Ken Thompson), Ντένις Ρίτσι (Dennis Ritchie) και Ντάγκλας Μακιλρόι (Douglas McIlroy). Σήμερα τα συστήματα Unix έχουν χωριστεί σε πολλούς κλάδους και αναπτύσσονται τόσο από την AT&T όσο και από άλλους εμπορικούς παράγοντες, όπως και από αρκετούς μη κερδοσκοπικούς οργανισμούς, όπως το πρόγραμμα GNU.

Ο νυν ιδιοκτήτης του εμπορικού συμβόλου Unix είναι μια ομάδα, η οποία ονομάζεται The Open Group, ενώ οι κάτοχοι των πνευματικών δικαιωμάτων του πηγαίου κώδικα του Unix είναι η ομάδα SCO Group και η εταιρία Νοβέλ (Novell). Μόνο συστήματα πλήρως συμβατά και πιστοποιημένα με το πρωτόκολλο Single UNIX Specification χαρακτηρίζονται ως «Unix» (τα υπόλοιπα χαρακτηρίζονται ως «παρόμοια με το Unix» ή, στην καθομιλούμενη των υπολογιστών, «Unix-οειδή», «Unix-like»).

Κατά τη διάρκεια των τελευταίων χρόνων της δεκαετίας του 1970 και των αρχών της δεκαετίας του 1980 η επιρροή του Unix στους ακαδημαϊκούς

κύκλους οδήγησε στην μαζική αποδοχή του (ειδικά από την παραλλαγή του BSD, προερχόμενη από το Πανεπιστήμιο του Μπέρκλεϊ) από εμπορικά προγράμματα Unix, με πιο αξιοσημείωτο αυτό της εταιρείας Sun Microsystems.

Μερικές φορές ο όρος Παραδοσιακό Unix χρησιμοποιείται για να περιγράψει ένα λειτουργικό σύστημα Unix ή GNU το οποίο έχει τα χαρακτηριστικά είτε της έκδοσης 7 του Unix είτε του UNIX System V.



Εικόνα 7 - Κλάδοι του Unix και παρόμοιων συστημάτων

Τα λειτουργικά συστήματα Unix χρησιμοποιούνται ευρέως και σε εξυπηρετητές και σε σταθμούς εργασίας. Το περιβάλλον Unix και το μοντέλο πελάτη - εξυπηρετητή ήταν απαραίτητα στοιχεία στην ανάπτυξη του

Διαδικτύου και τον αναπροσανατολισμό των υπολογιστών προς την δημιουργία και χρήση δικτύων αντί για ξεχωριστούς υπολογιστές.

Τόσο το Unix όσο και η γλώσσα προγραμματισμού C αναπτύχθηκαν από την εταιρία AT&T και διανεμήθηκαν σε κρατικά και σε ακαδημαϊκά ιδρύματα, με αποτέλεσμα να μεταφερθούν και να προσαρμοστούν σε κατά πολύ ευρύτερο φάσμα υπολογιστών από οποιοδήποτε άλλο λειτουργικό σύστημα. Συνεπώς, το Unix έγινε ταυτόσημο με την έννοια του ανοιχτού συστήματος (Open System).

Το Unix σχεδιάστηκε για να μεταφέρεται εύκολα σε άλλες πλατφόρμες και να υποστηρίζει πολλαπλές ταυτόχρονες εργασίες παράλληλα με την ταυτόχρονη χρήση του από πολλούς χρήστες, σε διάταξη χρονομερισμού. Τα συστήματα Unix χαρακτηρίζονται συνήθως από τις εξής ιδιότητες:

- χρήση απλού κειμένου για την αποθήκευση των δεδομένων,
- ιεραρχικό σύστημα αρχείων,
- η αντιμετώπιση συσκευών αλλά και κάποιων μορφών διαδιεργασιακής επικοινωνίας ως αρχεία και η χρήση ενός μεγάλου αριθμού εργαλείων, μικρές εφαρμογές που μπορούν να συνδυαστούν με ένα διερμηνέα γραμμής εντολών με χρήση σωληνώσεων (pipes), αντί για ένα μονολιθικό πρόγραμμα που θα υλοποιούσε την ίδια λειτουργία. Οι αρχές αυτές είναι γνωστές και ως η φιλοσοφία Unix.

Στο Unix το "λειτουργικό σύστημα" αποτελείται από πολλά τέτοια εργαλεία μαζί με το κύριο πρόγραμμα ελέγχου, τον πυρήνα. Ο πυρήνας παρέχει υπηρεσίες για την εκκίνηση και τερματισμό προγραμμάτων, χειρίζεται το σύστημα αρχείων και άλλες λειτουργίες "χαμηλού επιπέδου", τις οποίες μοιράζονται τα περισσότερα προγράμματα. Επιπλέον, ίσως πιο σημαντικό, σχεδιάζει την πρόσβαση στο υλικό ώστε να αποφύγει συγκρούσεις αν δυο

προγράμματα προσπαθούν ταυτόχρονα να έχουν πρόσβαση στον ίδιο πόρο ή συσκευή. Για να μεσολαβήσει σε τέτοιες προσβάσεις, ο πυρήνας έχει ειδικά δικαιώματα πάνω στο σύστημα, γεγονός που οδηγεί στη διαφοροποίηση: "χώρος πυρήνα" και "χώρος χρήστη".

Η αρχή του μικροπυρήνα (microkernel) αναπτύχθηκε σε μια προσπάθεια να αντιστραφεί η τάση για μεγαλύτερους πυρήνες, και για επιστροφή σε ένα σύστημα όπου οι περισσότερες λειτουργίες εκτελούνται από μικρά εργαλεία. Σε μια περίοδο που ο "κανονικός" υπολογιστής συμπεριλάμβανε σκληρό δίσκο για την αποθήκευση δεδομένων, και τερματικό για είσοδο και έξοδο, το μοντέλο αρχείων του Unix δούλευε αρκετά ικανοποιητικά, μιας και η είσοδος και έξοδος ήταν κυρίως "γραμμική". Όμως, τα σύγχρονα συστήματα περιλαμβάνουν δικτύωση, και άλλες νέες συσκευές. Με την ανάπτυξη των γραφικών διεπαφών, το μοντέλο αρχείων αποδείχθηκε ανεπαρκές για τη διαχείριση ασύγχρονων γεγονότων, όπως αυτά που προκαλούνται από ένα ποντίκι. Έτσι, τη δεκαετία 1980, η ασύγχρονη είσοδος/έξοδος και η διαδιεργασιακή επικοινωνία επεκτάθηκαν με sockets, κοινή μνήμη, ουρές μηνυμάτων, σημαφόρους, και άλλες λειτουργίες, καθώς τα πρωτόκολλα δικτύου μετακινήθηκαν εκτός του πυρήνα.

3.3 Γλώσσα προγραμματισμού C

Η C (προφέρεται "σι") είναι μια διαδικαστική γλώσσα προγραμματισμού γενικής χρήσης, η οποία αναπτύχθηκε αρχικά, μεταξύ του 1969 και του 1973, από τον Ντένις Ρίτσι στα εργαστήρια AT&T Bell Labs για να χρησιμοποιηθεί για την ανάπτυξη του λειτουργικού συστήματος UNIX. Όπως οι περισσότερες διαδικαστικές γλώσσες προγραμματισμού που ακολουθούν την παράδοση της ALGOL, η C έχει δυνατότητες δομημένου προγραμματισμού και επιτρέπει τη χρήση αναδρομής (αλλά όχι και εμφωλευμένων συναρτήσεων), ενώ, ο στατικός ορισμός του τύπου των μεταβλητών που

επιβάλλει, προλαμβάνει πολλά σφάλματα κατά την χρήση τους. Ο σχεδιασμός της περιλαμβάνει δομές που μεταφράζονται αποδοτικά σε τυπικές εντολές μηχανής (machine instructions) και εξ αιτίας αυτού χρησιμοποιείται συχνά σε εφαρμογές που παλιότερα γράφονταν σε συμβολική γλώσσα (assembly language). Αυτό ακριβώς το χαρακτηριστικό της, που έχει σαν συνέπεια και την αυξημένη ταχύτητα εκτέλεσης των εφαρμογών που γράφονται σε αυτή, καθώς και το γεγονός ότι είναι διαθέσιμη στα περισσότερα σημερινά λειτουργικά συστήματα, συνέβαλε κατά πολύ στην καθιέρωση της και την χρήση της για ανάπτυξη λειτουργικών συστημάτων και λοιπών προγραμμάτων συστήματος (system software), αλλά και απλών εφαρμογών.

Η C συγκαταλέγεται πλέον στις πιο ευρέως χρησιμοποιούμενες γλώσσες προγραμματισμού όλων των εποχών και πολλές νεώτερες γλώσσες έχουν επηρεαστεί άμεσα ή έμμεσα από αυτήν, συμπεριλαμβανομένων των C++, C#, D, Go, Java, JavaScript, Limbo, LPC, Perl, PHP, Python, καθώς και του κελύφους C (C shell) του Unix. Κάποιες από αυτές τις γλώσσες έχουν επηρεαστεί κυρίως στη σύνταξη τους, με το σύστημα τύπων, τα μοντέλα δεδομένων και το νόημα των εκφράσεων τους να διαφέρουν σημαντικά από την C. Η C++, ειδικά, ξεκίνησε σαν προεπεξεργαστής της C, αλλά έχει εξελιχθεί πλέον σε μια αντικειμενοστραφή γλώσσα, που αποτελεί υπερσύνολο της C.

3.3.1 **Φιλοσοφία**

Η C είναι μια σχετικά μινιμαλιστική γλώσσα προγραμματισμού. Ανάμεσα στους σχεδιαστικούς στόχους που έπρεπε να καλύψει η γλώσσα περιλαμβανόταν το ότι θα μπορούσε να μεταγλωττιστεί άμεσα με τη χρήση μεταγλωττιστή ενός περάσματος (single-pass compiler) — με άλλα λόγια, ότι θα απαιτούνταν μόνο ένας μικρός αριθμός από εντολές σε γλώσσα μηχανής για κάθε βασικό στοιχείο της, χωρίς εκτεταμένη υποστήριξη στον χρόνο

εκτέλεσης. Ως αποτέλεσμα, είναι δυνατό να γραφτεί κώδικας σε C σε χαμηλό επίπεδο προγραμματισμού με ακρίβεια ανάλογη της συμβολικής γλώσσας, στην πραγματικότητα η C ορισμένες φορές αποκαλείται (και χωρίς να υπάρχει πάντα αντιπαράθεση) «συμβολική γλώσσα υψηλού επιπέδου» («high-level assembly») ή «φορητή συμβολική γλώσσα» («portable assembly»). Επίσης, γίνονται αναφορές στη C ως γλώσσα προγραμματισμού μεσαίου επιπέδου.

3.3.2 **Χαρακτηριστικά**

Στη C δεν επιβάλλεται κάποια συγκεκριμένη μορφή στον πηγαίο κώδικα (όπως, για παράδειγμα, συνέβαινε στις αρχικές εκδόσεις της Fortran). Ο προγραμματιστής, χωρίς να αγνοεί φυσικά το συντακτικό της γλώσσας, είναι ελεύθερος να δώσει όποια μορφή θέλει στον κώδικα που γράφει (free-format source). Το ελληνικό ερωτηματικό (;) χρησιμοποιείται ως τερματιστής εντολών (και όχι ως διαχωριστής, όπως στην Pascal, παραδείγματος χάριν) και τα άγκιστρα ({}) χρησιμοποιούνται για την ομαδοποίηση εντολών (όπως τα begin/end στην Pascal).

Ακόμα, στη C όλος ο εκτελέσιμος κώδικας περιέχεται σε υπορουτίνες οι οποίες ονομάζονται «συναρτήσεις» (όχι με την αυστηρή έννοια του συναρτησιακού προγραμματισμού). Οι παράμετροι περνιούνται στις συναρτήσεις πάντα με τιμή (pass-by-value). Το πέρασμα με αναφορά (pass-by-reference) γίνεται έμμεσα στην ουσία, περνώντας, ως παραμέτρους των συναρτήσεων, δείκτες στις μεταβλητές των οποίων θέλουμε να αλλάζουμε τις τιμές μέσα από τις συναρτήσεις.

Η C έχει ακόμα τα εξής χαρακτηριστικά:

- Έχει ένα πολύ μικρό σταθερό πλήθος λέξεων-κλειδιών (keywords), το οποίο περιλαμβάνει ένα πλήρες σύνολο δομών/εντολών ελέγχου ροής: for, if/else, while, switch, και do/while, goto.

- Υπάρχει μόνο ένας χώρος ονομάτων (namespace) και τα ονόματα (μεταβλητών, συναρτήσεων, κ.τ.λ.) που ορίζονται από το χρήστη δεν διακρίνονται με κάποιο τρόπο από τις λέξεις-κλειδιά της γλώσσας.

- Υπάρχει ένα μεγάλο πλήθος αριθμητικών και λογικών τελεστών, όπως οι: +, +=, ++, -, -=, --, *, *=, /, /=, ==, &, &&, |, ||, ~, κ.ά.

- Σε μία εντολή μπορεί να γίνουν παραπάνω από μια εκχωρήσεις τιμών.

- Η τιμή που επιστρέφει μια συνάρτηση, μπορεί να αγνοηθεί εάν δεν χρειάζεται.

- Ο ορισμός των τύπων των μεταβλητών είναι στατικός και απαραίτητος, αλλά γίνονται έμμεσες μετατροπές από τη γλώσσα. Για παράδειγμα, παραστάσεις με τύπο χαρακτήρα μπορούν να χρησιμοποιηθούν σε σημεία που απαιτείται ακέραιος.

- Η σύνταξη των δηλώσεων ονομάτων προσομοιάζει την χρήση αυτών μέσα στον εκτελέσιμο κώδικα. Η C δεν έχει ειδική λέξη-κλειδί για τον ορισμό ονομάτων (όπως είναι η "var" στην Pascal, για παράδειγμα) ή συναρτήσεων (όπως η "function", πάλι στην Pascal). Μια γραμμή που ξεκινάει με το όνομα ενός τύπου, εκλαμβάνεται σαν ορισμός μεταβλητής ή συνάρτησης, ανάλογα με τον αν υπάρχουν, ή όχι, παρενθέσεις που περικλείουν (τυπικές) παραμέτρους συνάρτησης.

- Ο χρήστης μπορεί να ορίσει δικούς του τύπους (και σύνθετους), εάν το επιθυμεί. Μπορεί επίσης να ορίσει και τύπους εγγραφών (structs στη C, records σε άλλες γλώσσες).

- Μπορούν να οριστούν πίνακες, αν και δεν υπάρχει ειδική λέξη-κλειδί για τον ορισμό τους (όπως το "array" στην Pascal). Η δεικτοδότηση τους γίνεται με χρήση αγκυλών ([]), αν και πολύ συχνά γίνεται χρήση

αριθμητικής δεικτών. Το πρώτο στοιχείο κάθε πίνακα δεικτοδοτείται πάντα από το μηδέν (0). Π.χ.: Το στοιχείο month[0] είναι το πρώτο στοιχείο του πίνακα month. Τέλος, δεν υπάρχουν τελεστές για την σύγκριση ή εκχώρηση πινάκων.

- Είναι δυνατή η δημιουργία απαριθμήσιμων τύπων με τη χρήση της λέξης-κλειδί "enum", οι οποίοι μπορούν να χρησιμοποιηθούν όπου οι ακέραιοι και αντίστροφα.

- Δεν υπάρχει ιδιαίτερος τύπος για αλφαριθμητικά, τα οποία παραδοσιακά υλοποιούνται και αντιμετωπίζονται σαν πίνακες από χαρακτήρες, και έχουν έναν μηδενικό χαρακτήρα να σημαδεύει το τέλος τους (null-terminated arrays of characters).

- Είναι δυνατή η άμεση προσπέλαση χαμηλού επιπέδου στη μνήμη του υπολογιστή με τη χρήση δεικτών.

- Οι υπορουτίνες που δεν επιστρέφουν τιμή ("procedures" σε άλλες γλώσσες) είναι συναρτήσεις που ορίζονται να είναι τύπου "void" (ψευδοτύπος που δείχνει την απουσία επιστρεφόμενης τιμής, αλλά χρησιμοποιείται και για δείκτες που δεν δείχνουν σε αντικείμενο συγκεκριμένου τύπου).

- Δεν μπορούν να οριστούν συναρτήσεις μέσα σε άλλες συναρτήσεις (εμφωλιασμένες).

- Οι δείκτες σε συναρτήσεις και δεδομένα επιτρέπουν την υλοποίηση πολυμορφισμού στην πράξη.

- Ο προεπεξεργαστής της γλώσσας επιτρέπει τον ορισμό μακροεντολών, την συγχώνευση αρχείων πηγαίου κώδικα, καθώς και την μεταγλώττιση υπό συνθήκες.

- Αρχεία πηγαίου κώδικα μπορούν να μεταγλωττιστούν χωριστά και να συνδεθούν μαζί, ενώ υπάρχει η δυνατότητα ελέγχου της ορατότητας συναρτήσεων και μεταβλητών στα άλλα αρχεία (πέρα από αυτό στο οποίο ορίζονται) με το χαρακτηρισμό τους ως "static" ή "extern".

- Οι πολύπλοκες λειτουργίες, όπως οι λειτουργίες εισόδου/εξόδου, ο χειρισμός των αλφαριθμητικών, καθώς και οι μαθηματικές συναρτήσεις, έχουν ανατεθεί, με συνεπή τρόπο, στις αντίστοιχες βιβλιοθήκες.

Η C δεν διαθέτει κάποιες από τις δυνατότητες νεώτερων γλωσσών, όπως τον προσανατολισμό στα αντικείμενα και την συλλογή απορριμάτων (garbage collection).

Πιο κάτω είναι ένα μικρό παράδειγμα ενός προγράμματος σε C που εκτυπώνει στο τερματικό "Hello world":

```
#include <stdio.h>
int main( int argc, char **argv )
{
printf( "Hello world!" );
return 0;
}
```

Για να μετατραπεί ο πηγαίος κώδικας αυτός όμως σε μορφή εξόδου προς τον τελικό χρήστη πρέπει να χρησιμοποιηθεί ένας μεταγλωττιστής.

3.3.3 **Μεταγλωττιστής**

Μεταγλωττιστής ή μεταφραστής (στα αγγλικά compiler) ονομάζεται ένα πρόγραμμα που μετατρέπει/μεταφράζει κείμενο γραμμένο σε μια γλώσσα προγραμματισμού (πηγαία γλώσσα) σε μια άλλη γλώσσα προγραμματισμού (τη γλώσσα στόχο). Το κείμενο της εισόδου ονομάζεται πηγαίος κώδικας (source code) και η έξοδος του προγράμματος αντικειμενικός κώδικας (object code).

Ο όρος «μεταγλωττιστής» χρησιμοποιείται κυρίως για προγράμματα που μεταφράζουν μια γλώσσα προγραμματισμού υψηλού επιπέδου σε μια γλώσσα χαμηλότερου επιπέδου (όπως η συμβολική γλώσσα ή η γλώσσα μηχανής). Αν το μεταγλωττισμένο πρόγραμμα πρόκειται να εκτελεστεί σε έναν υπολογιστή που έχει διαφορετικό επεξεργαστή ή λειτουργικό σύστημα σε σχέση με την πλατφόρμα που εκτελείται ο μεταγλωττιστής, ο τελευταίος τότε ονομάζεται cross-compiler. Ένα πρόγραμμα που μεταφράζει από μια γλώσσα χαμηλού επιπέδου σε μια υψηλότερου επιπέδου ονομάζεται decompiler. Ένα πρόγραμμα που μεταφράζει από μια γλώσσα υψηλού επιπέδου σε μια άλλη, επίσης υψηλού επιπέδου, ονομάζεται συνήθως γλωσσικός μεταφραστής, μεταφραστής από πηγαίο κώδικα σε πηγαίο κώδικα (source to source translator) ή μετατροπέας γλωσσών. Ένα πρόγραμμα που μεταφράζει τη μορφή εκφράσεων σε άλλη μορφή, διατηρώντας την ίδια γλώσσα, ονομάζεται language rewriter.

Ένας μεταγλωττιστής μπορεί να περιλαμβάνει οποιαδήποτε από τις εξής λειτουργίες: λεκτική ανάλυση, προεπεξεργασία, συντακτική ανάλυση, σημασιολογική ανάλυση (μετάφραση καθοδηγούμενη από τη σύνταξη), παραγωγή κώδικα και βελτιστοποίηση κώδικα.

Τα σφάλματα προγραμμάτων που προκύπτουν από λανθασμένη μεταγλώττιση είναι πολύ δύσκολο να εντοπιστούν και να αντιμετωπιστούν. Για αυτόν τον λόγο οι κατασκευαστές μεταγλωττιστών κάνουν σημαντικές προσπάθειες για να βεβαιώσουν την ορθότητα λειτουργίας του λογισμικού τους.

Ο όρος μεταγλωττιστής μεταγλωττιστών (compiler-compiler) χρησιμοποιείται συχνά για τις γεννήτριες συντακτικών αναλυτών, που είναι εργαλεία που βοηθούν στην κατασκευή λεκτικών και συντακτικών αναλυτών.

3.4 Συμπεράσματα

Τόσο το Unix όσο και η γλώσσα προγραμματισμού C αναπτύχθηκαν από την εταιρία AT&T και διανεμήθηκαν σε κρατικά και σε ακαδημαϊκά ιδρύματα αρχικά. Η C διακρίνεται για την αυξημένη ταχύτητα της στην εκτέλεση εφαρμογών και το Unix για την «ευκολία» του στην διαδιεργασιακή επικοινωνία μέσω σωληνώσεων(pipes). Γι'αυτό και εμείς επιλέξαμε αυτά τα δυο για να υλοποιήσουμε την εφαρμογή μας που θα δούμε πιο κάτω αναλυτικότερα.

4 ο ΚΕΦΑΛΑΙΟ

ΤΟ ΠΡΟΓΡΑΜΜΑ

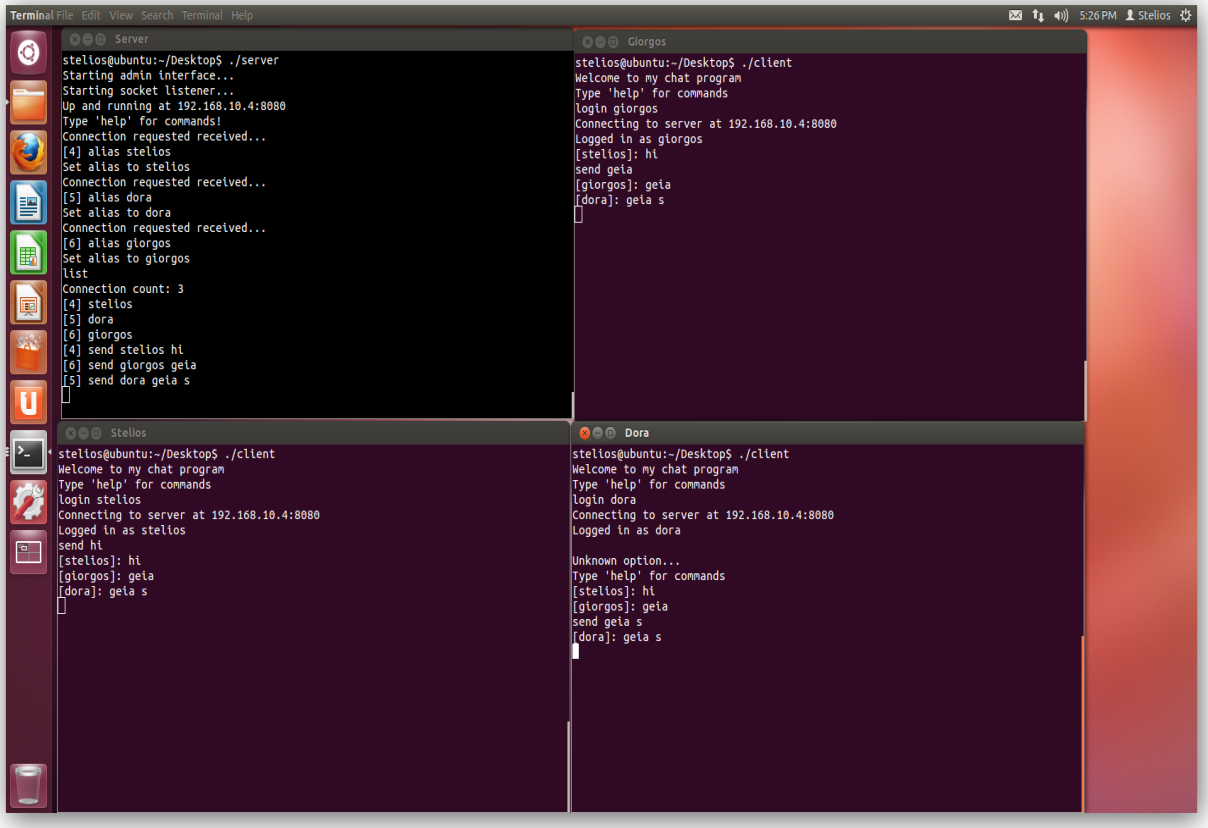
4.1 Εισαγωγή

Το πρόγραμμα μας υλοποιήθηκε σε γλώσσα προγραμματισμού C και δοκιμάστηκε στο Cygwin(Windows) και στα Ubuntu 12.04 LTS(Linux).

Σκοπός του προγράμματος ήταν να δείξουμε πως γίνεται η επικοινωνία μεταξύ διαφόρων client με τον server, πως δημιουργούμε sockets καθώς και πως τα χειριζόμαστε αυτά (δημιουργία, διαγράφη, διευθύνσεις,I/O). Επίσης χρησιμοποιήθηκε κώδικας για ανάγνωση αρχείων για προβολή πληροφοριών εφαρμογής και βοηθητικές εντολές του χρήστη.

Γι`αυτό το λόγο δημιουργήσαμε ένα απλό πρόγραμμα ανταλλαγής μηνυμάτων (chat) στο οποίο μπορεί να μπει ένας προκαθορισμένος αριθμός χρηστών (που ορίζουμε εμείς μέσω μεταβλητής στον κώδικα) και να ανταλλάσσουν δημόσια (public messages) μηνύματα ή ακόμη και μεταξύ τους(private messages).

Ο συνολικός πηγαίος κώδικας του προγράμματος είναι γύρω στις 800 γραμμές.



Εικόνα 8 - 3 Clients και ο Server

4.2 Οι βιβλιοθήκες ή κεφαλίδες (Library header)

Στην γλώσσα προγραμματισμού C η C πρότυπη βιβλιοθήκη είναι μια πρότυπη (στάνταρ) συλλογή από αρχεία επικεφαλίδες και συναρτήσεις βιβλιοθήκης που υλοποιούν κάποιες κοινές λειτουργίες, όπως είσοδος/έξοδος και χειρισμό αλφαριθμητικών, στη C. Σε αντίθεση με άλλες γλώσσες όπως η COBOL, η Fortran και η PL/I, η C δεν περιλαμβάνει ενσωματωμένες δεσμευμένες λέξεις για αυτές τις λειτουργίες , και γι' αυτό σχεδόν όλα τα προγράμματα γραμμένα στη C βασίζονται στην πρότυπη βιβλιοθήκη για να λειτουργήσουν. Οπότε για να μπορέσεις να χρησιμοποιήσεις συγκεκριμένες εντολές πρέπει να «φορτώσεις» την ανάλογη βιβλιοθήκη που περιέχει αυτές

τις εντολές για να τες καταλάβει ο μεταγλωττιστής. Αυτό επιτυγχάνεται γράφοντας στην αρχή του προγράμματος όλα τα ονόματα των βιβλιοθηκών που θέλεις να φορτώσεις μαζί με το #include πριν το όνομα της βιβλιοθήκης.

Στο πρόγραμμα μας χρησιμοποιήσαμε τις εξής βιβλιοθήκες :

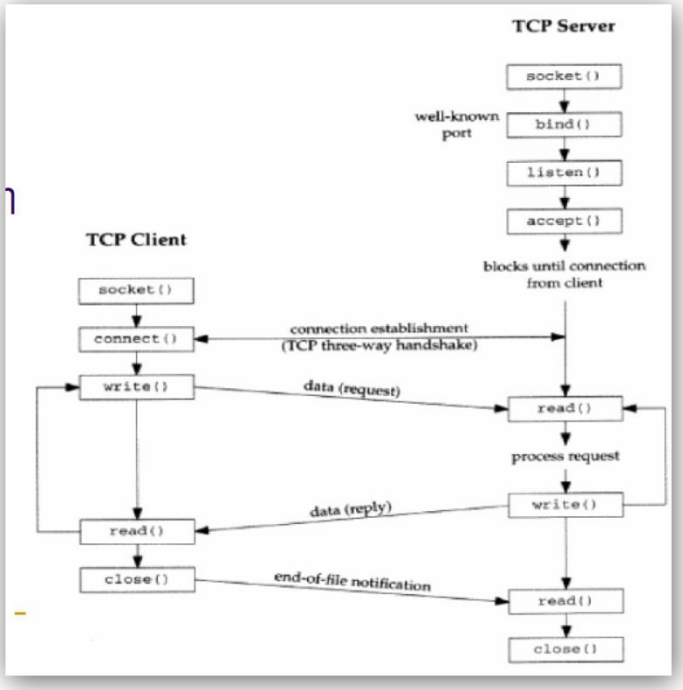
- <stdio.h> Παρέχει δυνατότητες εισόδου εξόδου. Περιέχει και την ξακουστή συνάρτηση printf .
- <stdlib.h> Για εκτέλεση διαφόρων λειτουργιών όπως μετατροπές, ψευδό-τυχαίους αριθμούς, δέσμευση μνήμης, έλεγχος διεργασιών, περιβάλλον, αναζήτηση , ταξινόμηση και σήματα.
- <string.h> Για χειρισμό αλφαριθμητικών διαφόρων ειδών.
- <errno.h> Για έλεγχο κωδικών λαθών που αναφέρονται από τις συναρτήσεις των βιβλιοθηκών.
- <netdb.h > Για τους ορισμούς για τις εργασίες στην βάση δεδομένων του δικτύου
- <*unistd.h*> Για ορισμό διάφορων συμβολικών σταθερών και τύπων και δηλώνει διάφορες λειτουργίες (π.χ pipe(),fork())
- <pthread.h> Νήματα (threads).
- <arpa/inet.h> Για διάφορους ορισμούς του διαδικτύου
- <netinet/in.h> Οικογένεια διευθύνσεων του Διαδικτύου (Internet address family). Για το port number, IP.
- <sys/types.h> Τύποι δεδομένων (data types,π.χ mutex)
- <sys/socket.h> Κύρια κεφαλίδα για δημιουργία και χειρισμό των sockets

4.3 Λειτουργία προγράμματος

Ξεκινώντας με την σύνδεση του server και του client χρησιμοποιήσαμε socket στην προκειμένη περίπτωση TCP πρωτόκολλο με IP:127.0.0.1 (local) και port 8080.

Ο server είναι ρυθμισμένος έτσι ώστε να δέχεται μέχρι 10 πελάτες ταυτόχρονα και άλλους 10 στην ουρά να περιμένουν. Οι τιμές αυτές μπορούν να αλλάξουν στον πηγαίο κώδικα πολύ εύκολα.

Η αλληλεπίδραση γενικά μεταξύ client-server με TCP είναι όπως φαίνεται πιο κάτω.



Εικόνα 9 – Αλληλεπίδραση Πελάτη–Εξυπηρετητή στο TCP μέσω Υποδοχών

Συνοπτικά η λειτουργία του προγράμματος μας έχει ως εξής:

Ο εξυπηρετητής (server):

- Δημιουργεί μια υποδοχή (socket())

  sockfd = socket(AF_INET, SOCK_STREAM, 0);

- δεσμεύει την υποδοχή στην επιθυμητή τοπική δικτυακή διεύθυνση τελικού σημείου (bind())

  *bind(sockfd, (struct sockaddr *)&serv_addr, sizeof(struct sockaddr));*

- ζητά από το ΛΣ να "ακούει" για συνδέσεις (listen())

  listen(sockfd, BACKLOG);

- εισέρχεται σε κατάσταση όπου αναμένει διαρκώς να του παραδώσει μια σύνδεση το ΛΣ (accept())

  accept(sockfd,(struct sockaddr)&client_addr,(socklen_t*)&sin_size));*

- όταν λαμβάνει εντολές από τους clients στέλλει τις πληροφορίες που ζήτησαν (recv(),send())

  *bytes = recv(threadinfo.sockfd, (void *)&packet, sizeof(struct PACKET), 0);*

  *sent = send(curr->threadinfo.sockfd, (void *)&spacket, sizeof(struct PACKET), 0);*

- Στην περίπτωση που ο χρήστης γράψει την εντολή exit κλείνει τα sockets , αποδεσμεύει τους πόρους του συστήματος και τερματίζει την λειτουργία του. (close(),free())

Ο πελάτης (client):

- Δημιουργεί μια υποδοχή δεσμεύει την υποδοχή στην επιθυμητή τοπική δικτυακή διεύθυνση (socket())

  newfd = socket(AF_INET, SOCK_STREAM, 0));

- Ζητά από το ΛΣ να συνδέσει την υποδοχή με κάποια υποδοχή στη διεύθυνση αποδέκτη (connect())

  *connect(newfd, (struct sockaddr *)&serv_addr, sizeof(struct sockaddr));*

- Μεταφέρει τις εντολές στον εξυπηρετητή και λαμβάνει τις πληροφορίες και τις εμφανίζει στον πελάτη (recv(),send())

  *recvd = recv(sockfd, (void *)&packet, sizeof(struct PACKET), 0);*

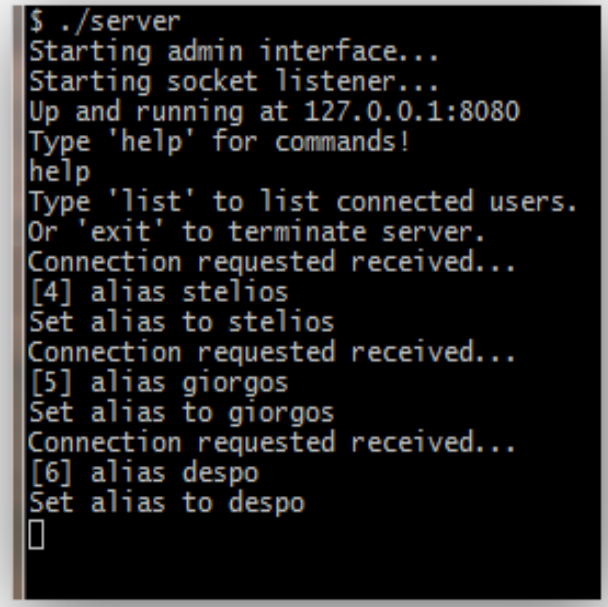
  *sent = send(sockfd, (void *)&packet, sizeof(struct PACKET), 0);*

4.3.1 **Ο εξυπηρετητής (Server)**

Ο πηγαίος κώδικας του server του προγράμματος μας αποτελείται από 400 γραμμές πηγαίου κώδικα περίπου όπως φαίνονται στο παράρτημα.

Ο εξυπηρετητής αφού ξεκινήσει και συνδεθεί μπορεί να δεχτεί 3 εντολές από τον χρήστη:

- Exit – Για τερματισμό του εξυπηρετητή και αποσύνδεση πελατών
- Help – Για προβολή των εντολών που μπορεί να χρησιμοποιήσει ο χρήστης
- List - Δείχνει όλους τους συνδεδεμένους χρήστες



Εικόνα 10 - Εξυπηρετητής στην γραμμή εντολών

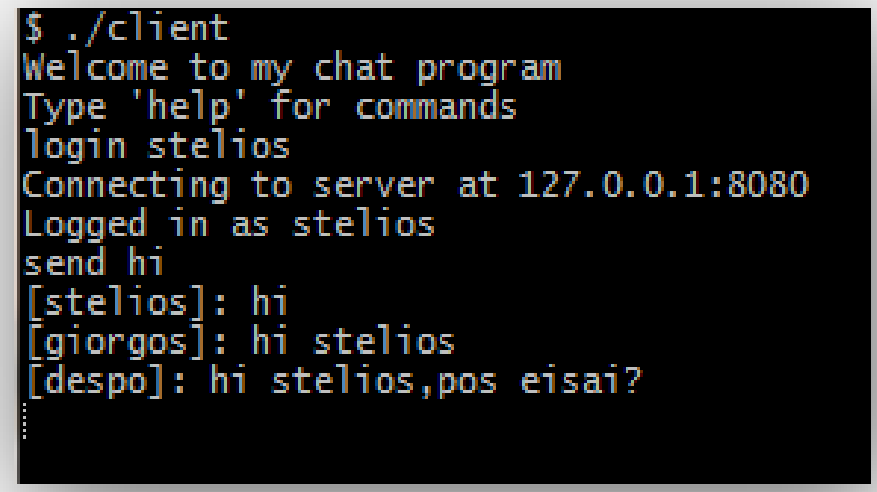
Δηλαδή με λίγα λόγια ο χρήστης-διαχειριστής που χρησιμοποιεί το πρόγραμμα μας (server) μπορεί να δει πόσοι συνδεδεμένοι χρήστες υπάρχουν , τα ονόματα αυτών, τις πληροφορίες που ανταλλάσουν καθώς και να ζητήσει βοήθεια για τις εντολές που μπορεί να χρησιμοποιήσει ανά πάσα στιγμή.

4.3.2 **Ο πελάτης (client)**

Ο πηγαίος κώδικας του πελάτη του προγράμματος μας αποτελείται από 400 γραμμές πηγαίου κώδικα περίπου όπως φαίνονται στο παράρτημα.

Ο πελάτης αφού ξεκινήσει και συνδεθεί μπορεί να δεχτεί 7 εντολές από τον χρήστη:

- login [alias] – Για να συνδεθεί με τον εξυπηρετητή αρχικά. Στην περίπτωση που δεν οριστεί όνομα(alias) ορίζεται ως Anonymous.
- alias [new_name] - Για να αλλάξει το όνομα του ο χρήστης
- send [message] – Για να στείλει ένα δημόσιο μήνυμα
- whisper [user_alias] [message] - Για να στείλει ένα προσωπικό μήνυμα σε συγκεκριμένο χρήστη
- logout – Για να αποσυνδεθεί από τον εξυπηρετητή
- info – Για να δει τις πληροφορίες της εφαρμογής
- help - Προβολή όλων των εντολών που μπορεί να χρησιμοποιήσει ο χρήστης
- exit – Για την έξοδο του από το πρόγραμμα και αποσύνδεση από τον εξυπηρετητή



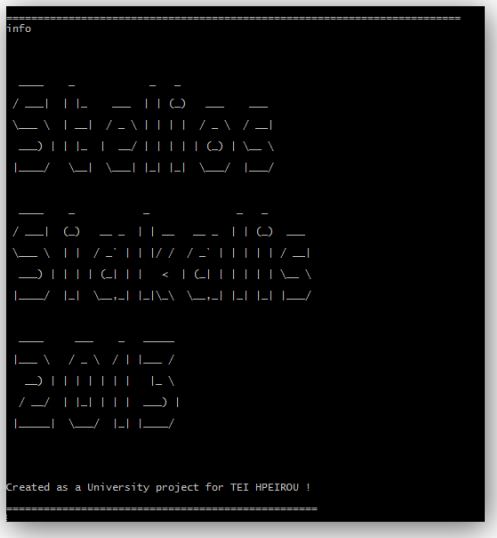
Εικόνα 11 - Πελάτης στην γραμμή εντολών

Δηλαδή με λίγα λόγια ένας χρήστης που χρησιμοποιεί το πρόγραμμα μας (client) μπορεί να συνδεθεί στον εξυπηρετητή, να αποσυνδεθεί, να αλλάξει το όνομα του ανά πάσα στιγμή, να στείλει μηνύματα σε όλους ή σε ένα άτομο συγκεκριμένα καθώς και να ζητήσει από το σύστημα να του πει τις εντολές που μπορεί να χρησιμοποιήσει.

4.4 Συμπεράσματα και μελλοντική εξέλιξη εφαρμογής

Η υλοποίηση της πτυχιακής ήταν αρκετά ενδιαφέρουσα και αρκετά μεγάλη πρόκληση για εμένα. Μέσα από την πτυχιακή έμαθα αρκετά ενδιαφέροντα πράγματα που θα μου χρησιμευόσουν στο μέλλον, όπως το να δημιουργώ συνδέσεις τύπου client-server, να χειρίζομαι τα νήματα (threads) για καλύτερη και γρηγορότερη λειτουργία των προγραμμάτων, πώς να προγραμματίζω σε γλώσσα προγραμματισμού C αλλά και πώς να χρησιμοποιώ καλύτερα τα τύπου Unix λειτουργικά συστήματα.

Στο μέλλον θα μπορούσα να εξελίξω την εφαρμογή μου σε αρκετά σημεία. Καταρχήν θα ήταν να γίνει ο server κάπου online με σταθερό IP,port και να συνδέονται περισσότεροι χρήστες ταυτόχρονα. Επίσης θα της έκανα ένα πιο ωραίο γραφικό περιβάλλον στον χρήστη (UI), με περισσότερα παράθυρα, μενού, κουμπιά, χρώματα, φόντα, πεδία εισαγωγής κτλ. Ακόμη θα μπορούσα να προσθέσω περισσότερες επιλογές εντολών στους χρήστες, πχ να μπορούν να στέλλουν αρχεία στους άλλους χρήστες ή και ακόμη φωνητικά μηνύματα.



Εικόνα 12 - Εντολή info

ΒΙΒΛΙΟΓΡΑΦΙΑ

Βιβλίο

1. BlackU., Internet Architecture: An Introduction to IP Protocols, Prentice Hall, 2000.
2. Gralla P., How the Internet Works, Que/Sams Publishing, 2004.
3. Δικτυακός Προγραμματισμός, Douglas E. Comer, David L. Stevens, Εκδόσεις ΙΩΝ, 2005
4. Unix Network Programming, W. Richard Stevens, Prentice-Hall, 1994
5. W. Richard Stevens, Unix Network Programming, Prentice-Hall, 1990
6. W. Richard Stevens, TCP/IP Illustrated, Volume 1: The Protocols, Addison-Wesley, 1994
7. W. Richard Stevens, TCP/IP Illustrated, Volume 3: TCP for Transactions,
8. HTTP, NNTP, and the UNIX Domain Protocols, Addison-Wesley, 1996
9. Gary R. Wright and W. Richard Stevens,TCP/IP Illustrated, Volume 2: The Implementation, Addison-Wesley, 1995
10. Marshall McKusick, Keith Bostic, Michael J. Karels, John S. Quarterman
11. T, Design and Implementation of the 4.4 BSD Operating System, Addison-Wesley, 1996
12. Brian Kernighan and Dennis Ritchie, The C Programming Language,Prentice-Hall, 1989(Second Edition)

13. Samuel Harbison and Guy Steele, Jr.C: A Reference Manual (4th edition),Prentice-Hall, 1995
14. P. J. Plauger,The Standard C Library,Prentice-Hall, 1992
15. Michael J. Donahoo, Kenneth L. Calvert,TCP/IP Sockets in C: Practical Guide for Programmers, Second Edition,Morgan Kaufmann,2009

Διαδίκτυο

1. http://www.imada.sdu.dk/~svalle/courses/dm14-2005/mirror/c/
2. http://en.wikipedia.org/wiki/Computer_network_programming
3. http://el.wikipedia.org/wiki/Domain_Name_System
4. http://www.garykessler.net/library/tcpip.html
5. http://el.wikipedia.org/wiki/POSIX
6. http://cs.ecs.baylor.edu/~donahoo/practical/CSockets2/textcode.html
7. http://pubs.opengroup.org/onlinepubs/009695399/
8. http://el.wikipedia.org/wiki/UNIX
9. http://el.wikipedia.org/wiki/%CE%94%CE%B9%CE%B5%CF%8D%CE%B8%CF%85%CE%BD%CF%83%CE%B7_IP
10. http://el.wikipedia.org/wiki/%CE%94%CE%B9%CE%B1%CE%BA%CE%BF%CE%BC%CE%B9%CF%83%CF%84%CE%AE%CF%82
11. http://msc.ds.unipi.gr/network-programming-next-year/
12. http://www.it.uom.gr/teaching/c_sys/node28.html

ΠΑΡΑΡΤΗΜΑΤΑ

CLIENT.C

```
/*
** Author: Stelios Siakalis
** Date: 26th November, 2013
** Copyright: NO COPYRIGHT ISSUES.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

#include <netdb.h>
#include <unistd.h>
#include <pthread.h>

#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>

#define SERVERIP "127.0.0.1" //server`s IP
#define SERVERPORT 8080 //server`s Port

#define BUFFSIZE 1024 //buffer size in KB
#define ALIASLEN 32 //alias length in KB
#define OPTLEN 16 //option length in KB
#define LINEBUFF 2048 //file`s line length in KB

struct PACKET
{
   char option[OPTLEN]; // instruction
   char alias[ALIASLEN]; // client's alias
   char buff[BUFFSIZE]; // payload
};

struct USER
{
   int sockfd; // user's socket descriptor
   char alias[ALIASLEN]; // user's name
```

```
};

struct THREADINFO
{
   pthread_t thread_ID; // thread's pointer
   int sockfd; // socket file descriptor
};

int isconnected, sockfd;
char option[LINEBUFF];
struct USER me;

int connect_with_server();
void setalias(struct USER *me);
void logout(struct USER *me);
void login(struct USER *me);
void *receiver(void *param);
void sendtoall(struct USER *me, char *msg);
void sendtoalias(struct USER *me, char * target, char *msg);

int main(int argc, char **argv)
{
   int sockfd, aliaslen;
   memset(&me, 0, sizeof(struct USER));
/* welcome user */
printf("Welcome to my chat program\n");
printf("Type 'help' for commands\n");
/* waiting for commands */
   while(gets(option))
   {
/* exit command */
      if(!strncmp(option, "exit", 4))
      {
         logout(&me);
         break;
      }
/* info command */
      if(!strncmp(option, "info", 4))
      {
         FILE *fin = fopen("info.txt", "r");
         if(fin != NULL)
         {
            while(fgets(option, LINEBUFF-1, fin)) puts(option);
            fclose(fin);
         }
         else
```

```
            {
               fprintf(stderr, "Information file not found...\n");
            }
         }
/* help command */
         else if(!strncmp(option, "help", 4))
         {
            FILE *fin = fopen("help.txt", "r");
            if(fin != NULL)
            {
               while(fgets(option, LINEBUFF-1, fin)) puts(option);
               fclose(fin);
            }
            else
            {
               fprintf(stderr, "Help file not found...\n");
            }
         }
/* login command */
         else if(!strncmp(option, "login", 5))
         {
            char *ptr = strtok(option, " ");
            ptr = strtok(0, " ");
            memset(me.alias, 0, sizeof(char) * ALIASLEN);
            if(ptr != NULL)
            {
               aliaslen = strlen(ptr);
               if(aliaslen > ALIASLEN) ptr[ALIASLEN] = 0;
               strcpy(me.alias, ptr);
            }
            else
            {
               strcpy(me.alias, "Anonymous");
            }
            login(&me);
         }
/* alias command */
         else if(!strncmp(option, "alias", 5))
         {
            char *ptr = strtok(option, " ");
            ptr = strtok(0, " ");
            memset(me.alias, 0, sizeof(char) * ALIASLEN);
            if(ptr != NULL)
            {
               aliaslen = strlen(ptr);
               if(aliaslen > ALIASLEN) ptr[ALIASLEN] = 0;
```

```
                strcpy(me.alias, ptr);
                setalias(&me);
            }
        }
/* whisper command */
        else if(!strncmp(option, "whisper", 7))
        {
            char *ptr = strtok(option, " ");
            char temp[ALIASLEN];
            ptr = strtok(0, " ");
            memset(temp, 0, sizeof(char) * ALIASLEN);
            if(ptr != NULL)
            {
                aliaslen = strlen(ptr);
                if(aliaslen > ALIASLEN) ptr[ALIASLEN] = 0;
                strcpy(temp, ptr);
                while(*ptr) ptr++;
                ptr++;
                while(*ptr <= ' ') ptr++;
                sendtoalias(&me, temp, ptr);
            }
        }
/* send command */
        else if(!strncmp(option, "send", 4))
        {
            sendtoall(&me, &option[5]);
        }
/* logout command */
        else if(!strncmp(option, "logout", 6))
        {
            logout(&me);
        }
/* unknown command */
        else
{
fprintf(stderr, "Unknown option...\n");
printf("Type 'help' for commands\n");
}
    }
    return 0;
}

/**
*Logins user to server
*@param User structure with user information
**/
```

```
void login(struct USER *me)
{
   int recvd;

/* user already connected */
   if(isconnected)
   {
      fprintf(stderr, "You are already connected to server at %s:%d\n", SERVERIP,
SERVERPORT);
      return;
   }
/* connect user */
   sockfd=-1;//reset value
   sockfd = connect_with_server();
   if(sockfd >= 0)
   {
      isconnected = 1;
      me->sockfd = sockfd;
      if(strcmp(me->alias, "Anonymous")) setalias(me);
      printf("Logged in as %s\n", me->alias);
      //printf("Receiver started [%d]...\n", sockfd);
      struct THREADINFO threadinfo;
      pthread_create(&threadinfo.thread_ID, NULL, receiver, (void *)&threadinfo);

   }
   else
   {
      fprintf(stderr, "Connection rejected...\n");
   }
}

/**
*Connecting user to server using sockets
**/
int connect_with_server()
{
   int newfd=0;
   int err_ret=0;
   struct sockaddr_in serv_addr;
   struct hostent *to;

   /* generate address */
   if((to = gethostbyname(SERVERIP))==NULL)
   {
      err_ret = errno;
      fprintf(stderr, "gethostbyname() error...\n");
```

```
        return err_ret;
    }

    /* open a socket */
    if((newfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
    {
        err_ret = errno;
        fprintf(stderr, "socket() error...\n");
        return err_ret;
    }

    /* set initial values */
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(SERVERPORT);
    serv_addr.sin_addr = *((struct in_addr *)to->h_addr);
    memset(&(serv_addr.sin_zero), 0, 8);

    /* try to connect with server */
    if(connect(newfd, (struct sockaddr *)&serv_addr, sizeof(struct sockaddr)) < 0)
    {
        err_ret = errno;
        fprintf(stderr, "connect() error...\n");
        isconnected = 0;
        return err_ret;
    }
    else
    {
        printf("Connecting to server at %s:%d\n", SERVERIP, SERVERPORT);
        return newfd;
    }
}

/**
*Logout user from server
*@param User pointer structure with user information
**/
void logout(struct USER *me)
{
    int sent;
    struct PACKET packet;

/* user not connected */
    if(!isconnected)
    {
        fprintf(stderr, "You are not connected...\n");
        return;
```

```
    }
    memset(&packet, 0, sizeof(struct PACKET));
    strcpy(packet.option, "exit");
    strcpy(packet.alias, me->alias);
    /* send request to close this connetion */
    sent = send(sockfd, (void *)&packet, sizeof(struct PACKET), 0);
    isconnected = 0;
}

/**
*Changing user alias
*@param User pointer structure with user information
**/
void setalias(struct USER *me)
{
    int sent;
    struct PACKET packet;
    if(!isconnected)
    {
        fprintf(stderr, "You are not connected...\n");
        return;
    }
    memset(&packet, 0, sizeof(struct PACKET));
    strcpy(packet.option, "alias");
    strcpy(packet.alias, me->alias);
    /* send request to close this connetion */
    sent = send(sockfd, (void *)&packet, sizeof(struct PACKET), 0);
}

/**
*Receiving packets from server(messages)
**/
void *receiver(void *param)
{
    int recvd;
    struct PACKET packet;
    //printf("Waiting here [%d]...\n", sockfd);

/* receive packets from server */
    while(isconnected)
    {
        recvd = recv(sockfd, (void *)&packet, sizeof(struct PACKET), 0);
        if(!recvd)//no connection to server
        {
            fprintf(stderr, "Connection lost from server...\n");
            isconnected = 0;
```

```
            close(sockfd);
            break;
        }
        if(recvd > 0)//print messages
        {
            printf("[%os]: %os\n", packet.alias, packet.buff);
        }
        memset(&packet, 0, sizeof(struct PACKET));
    }
    return NULL;
}

/**
*Sending messages to all users
*@param *me pointer of structure with user information
*@param *msg pointer of char array with message to sent
**/
void sendtoall(struct USER *me, char *msg)
{
    int sent;
    struct PACKET packet;

/* user not connected */
    if(!isconnected)
    {
        fprintf(stderr, "You are not connected...\n");
        return;
    }
    msg[BUFFSIZE] = 0;
    memset(&packet, 0, sizeof(struct PACKET));
    strcpy(packet.option, "send");
    strcpy(packet.alias, me->alias);
    strcpy(packet.buff, msg);

    /* send request to close this connetion */
    sent = send(sockfd, (void *)&packet, sizeof(struct PACKET), 0);
}

/**
*Sending private message to specific user alias
*@param *me pointer of structure with user information
*@param *targer pointer of target user alias to sent private message
*@param *msg pointer of char array with message to sent
**/
void sendtoalias(struct USER *me, char *target, char *msg)
{
```

```
    int sent, targetlen;
    struct PACKET packet;
    if(target == NULL)//no target
    {
        return;
    }
    if(msg == NULL)//no message
    {
        return;
    }
    if(!isconnected)//not connected
    {
        fprintf(stderr, "You are not connected...\n");
        return;
    }
    msg[BUFFSIZE] = 0;
    targetlen = strlen(target);
    memset(&packet, 0, sizeof(struct PACKET));
    strcpy(packet.option, "whisper");
    strcpy(packet.alias, me->alias);
    strcpy(packet.buff, target);
    strcpy(&packet.buff[targetlen], " ");
    strcpy(&packet.buff[targetlen+1], msg);

    /* send request to close this connetion */
    sent = send(sockfd, (void *)&packet, sizeof(struct PACKET), 0);
}
```

SERVER.C

```
/*
** Author: Stelios Siakalis
** Date: 26th November, 2013
** Copyright: NO COPYRIGHT ISSUES.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

#include <netdb.h>
#include <unistd.h>
#include <pthread.h>

#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>

#define IP "127.0.0.1" //server IP
#define PORT 8080 //server port
#define BACKLOG 10 //the maximum length the queue of pending connections
#define CLIENTS 10 //the maximum length the queue of client connections

#define BUFFSIZE 1024 //buffer size in kilobytes
#define ALIASLEN 32 //user alias length in kilobytes
#define OPTLEN 16 //user`s command-option length in kilobytes

struct PACKET
{
   char option[OPTLEN]; // instruction
   char alias[ALIASLEN]; // client's alias
   char buff[BUFFSIZE]; // payload
};

struct THREADINFO
{
   pthread_t thread_ID; // thread's pointer
   int sockfd; // socket file descriptor
   char alias[ALIASLEN]; // client's alias
};

struct LLNODE
{
```

```
    struct THREADINFO threadinfo;
    struct LLNODE *next;
};

struct LLIST
{
    struct LLNODE *head, *tail;
    int size;
};

/**
*Helper method for comparing client`s sockets
*@param *a structure pointer of 1st client thread
*@param *b structure pointer of 2nd client thread
**/
int compare(struct THREADINFO *a, struct THREADINFO *b)
{
    return a->sockfd - b->sockfd;
}

/**
*Initialize client list
*@param *l1 pointer of list to init
**/
void list_init(struct LLIST *ll)
{
    ll->head = ll->tail = NULL;
    ll->size = 0;
}

/**
*Inserts user to client list
*@param *l1 pointer of list to edit
*@param *thr_info the thread to work with
**/
int list_insert(struct LLIST *ll, struct THREADINFO *thr_info)
{
    if(ll->size == CLIENTS) return -1;//empty list-no clients to insert
    if(ll->head == NULL)
    {
        ll->head = (struct LLNODE *)malloc(sizeof(struct LLNODE));
        ll->head->threadinfo = *thr_info;
        ll->head->next = NULL;
        ll->tail = ll->head;
    }
    else
```

```
    {
        ll->tail->next = (struct LLNODE *)malloc(sizeof(struct LLNODE));
        ll->tail->next->threadinfo = *thr_info;
        ll->tail->next->next = NULL;
        ll->tail = ll->tail->next;
    }
    ll->size++;
    return 0;
}

/**
*Delete user from client list
*@param *ll pointer of list to work with
*@param *thr_info the thread to work with
**/
int list_delete(struct LLIST *ll, struct THREADINFO *thr_info)
{
    struct LLNODE *curr, *temp;
    if(ll->head == NULL) return -1;//empty list-no client to delete
    if(compare(thr_info, &ll->head->threadinfo) == 0)
    {
        temp = ll->head;
        ll->head = ll->head->next;
        if(ll->head == NULL) ll->tail = ll->head;
//free memory
        free(temp);
        ll->size--;
        return 0;
    }
    for(curr = ll->head; curr->next != NULL; curr = curr->next)
    {
        if(compare(thr_info, &curr->next->threadinfo) == 0)
        {
            temp = curr->next;
            if(temp == ll->tail) ll->tail = curr;
            curr->next = curr->next->next;
//free memory
            free(temp);
            ll->size--;
            return 0;
        }
    }
    return -1;
}

/**
```

```
*Prints client list
*@param *l1 pointer of list to print
**/
void list_dump(struct LLIST *ll)
{
   struct LLNODE *curr;
   struct THREADINFO *thr_info;
   printf("Connection count: %d\n", ll->size);
   for(curr = ll->head; curr != NULL; curr = curr->next)
   {
      thr_info = &curr->threadinfo;
      printf("[%d] %s\n", thr_info->sockfd, thr_info->alias);
   }
}

int sockfd, newfd;
struct THREADINFO thread_info[CLIENTS];
struct LLIST client_list;
pthread_mutex_t clientlist_mutex;

void *io_handler(void *param);
void *client_handler(void *fd);

/**
*Main method
**/
int main(int argc, char **argv)
{
   int err_ret, sin_size;
   struct sockaddr_in serv_addr, client_addr;
   pthread_t interrupt;

   /* initialize linked list */
   list_init(&client_list);

   /* initiate mutex */
   pthread_mutex_init(&clientlist_mutex, NULL);

   /* open a socket */
   if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
   {
      err_ret = errno;
      fprintf(stderr, "socket() failed...\n");
      return err_ret;
   }
```

```
   /* set initial values */
   serv_addr.sin_family = AF_INET;
   serv_addr.sin_port = htons(PORT);
   serv_addr.sin_addr.s_addr = inet_addr(IP);
   memset(&(serv_addr.sin_zero), 0, 8);

   /* bind address with socket */
   if(bind(sockfd, (struct sockaddr *)&serv_addr, sizeof(struct sockaddr)) == -1)
   {
      err_ret = errno;
      fprintf(stderr, "bind() failed...\n");
      return err_ret;
   }

   /* start listening for connection */
   if(listen(sockfd, BACKLOG) == -1)
   {
      err_ret = errno;
      fprintf(stderr, "listen() failed...\n");
      return err_ret;
   }

   /* initiate interrupt handler in a new thread for IO controlling */
   printf("Starting admin interface...\n");
   if(pthread_create(&interrupt, NULL, io_handler, NULL) != 0)
   {
      err_ret = errno;
      fprintf(stderr, "pthread_create() failed...\n");
      return err_ret;
   }

   /* keep accepting connections */
   printf("Starting socket listener...\n");
   printf("Up and running at %s:%d\n",IP,PORT);
   printf("Type 'help' for commands!\n");
   while(1)
   {
      sin_size = sizeof(struct sockaddr_in);
      if((newfd = accept(sockfd, (struct sockaddr *)&client_addr, (socklen_t*)&sin_size))
== -1)
      {
         err_ret = errno;
         fprintf(stderr, "accept() failed...\n");
         return err_ret;
      }
      else
```

```
        {
//  check client counter
            if(client_list.size == CLIENTS)
            {
               fprintf(stderr, "Connection full, request rejected...\n");
               continue;
            }
//init new client
            printf("Connection requested received...\n");
            struct THREADINFO threadinfo;
            threadinfo.sockfd = newfd;
            strcpy(threadinfo.alias, "Anonymous");
//acquire a lock on the specified mutex variable
            pthread_mutex_lock(&clientlist_mutex);
//insert Anonymous client to list
            list_insert(&client_list, &threadinfo);
//unlock mutex variable
            pthread_mutex_unlock(&clientlist_mutex);
//create client thread
            pthread_create(&threadinfo.thread_ID,      NULL,      client_handler,      (void
*)&threadinfo);
        }
    }

    return 0;
}

/**
*Handles server`s io(commands)
*@param *param
**/
void *io_handler(void *param)
{
    char option[OPTLEN];
    while(scanf("%s", option)==1)
    {
/* exit command (server) */
        if(!strcmp(option, "exit"))
        {
            /* clean up */
            printf("Terminating server...\n");
            pthread_mutex_destroy(&clientlist_mutex);
            close(sockfd);
            exit(0);
        }
/* list command (server) */
```

```
        else if(!strcmp(option, "list"))
        {
           pthread_mutex_lock(&clientlist_mutex);
           list_dump(&client_list);
           pthread_mutex_unlock(&clientlist_mutex);
        }
/* help command (server) */
else if(!strcmp(option, "help"))
{
printf("Type 'list' to list connected users.\n");
printf("Or 'exit' to terminate server.\n");
}
/* unknown command (server) */
        else
        {
           fprintf(stderr, "Unknown command: '%s'\n", option);
          printf("Type 'help' for commands");
        }
    }
    return NULL;
}

/**
*Handles client`s commands and sends back to clients via sockets
*@param *fd
**/
void *client_handler(void *fd)
{
    struct THREADINFO threadinfo = *(struct THREADINFO *)fd;
    struct PACKET packet;
    struct LLNODE *curr;
    int bytes, sent;
    while(1)
    {
        bytes = recv(threadinfo.sockfd, (void *)&packet, sizeof(struct PACKET), 0);
        if(!bytes)
        {
           fprintf(stderr, "Connection lost from [%d] %s...\n", threadinfo.sockfd,
threadinfo.alias);
           pthread_mutex_lock(&clientlist_mutex);
//delete client
           list_delete(&client_list, &threadinfo);
           pthread_mutex_unlock(&clientlist_mutex);
           break;
        }
       printf("[%d] %s %s %s\n", threadinfo.sockfd, packet.option, packet.alias, packet.buff);
```

```
/* Alias command */
        if(!strcmp(packet.option, "alias"))
        {
            printf("Set alias to %s\n", packet.alias);
            pthread_mutex_lock(&clientlist_mutex);
            for(curr = client_list.head; curr != NULL; curr = curr->next)
            {
                if(compare(&curr->threadinfo, &threadinfo) == 0)
                {
                    strcpy(curr->threadinfo.alias, packet.alias);
                    strcpy(threadinfo.alias, packet.alias);
                    break;
                }
            }
            pthread_mutex_unlock(&clientlist_mutex);
        }
/* Whisper command */
        else if(!strcmp(packet.option, "whisper"))
        {
            int i;
            char target[ALIASLEN];
            for(i = 0; packet.buff[i] != ' '; i++);
            packet.buff[i++] = 0;
            strcpy(target, packet.buff);
            pthread_mutex_lock(&clientlist_mutex);
            for(curr = client_list.head; curr != NULL; curr = curr->next)
            {
                if(strcmp(target, curr->threadinfo.alias) == 0)
                {
                    struct PACKET spacket;
                    memset(&spacket, 0, sizeof(struct PACKET));
                    if(!compare(&curr->threadinfo, &threadinfo)) continue;
                    strcpy(spacket.option, "msg");
                    strcpy(spacket.alias, packet.alias);
                    strcpy(spacket.buff, &packet.buff[i]);
//send message
                    sent = send(curr->threadinfo.sockfd, (void *)&spacket, sizeof(struct
PACKET), 0);
                }
            }
            pthread_mutex_unlock(&clientlist_mutex);
        }
/* Send command */
        else if(!strcmp(packet.option, "send"))
        {
```

```
            pthread_mutex_lock(&clientlist_mutex);
            for(curr = client_list.head; curr != NULL; curr = curr->next)
            {
               struct PACKET spacket;
               memset(&spacket, 0, sizeof(struct PACKET));
               // if(!compare(&curr->threadinfo, &threadinfo)) continue;
               strcpy(spacket.option, "msg");
               strcpy(spacket.alias, packet.alias);
               strcpy(spacket.buff, packet.buff);
//sent message
               sent = send(curr->threadinfo.sockfd, (void *)&spacket, sizeof(struct PACKET),
0);
            }
            pthread_mutex_unlock(&clientlist_mutex);
         }
/* Exit command */
         else if(!strcmp(packet.option, "exit"))
         {
            printf("[%d] %s has disconnected...\n", threadinfo.sockfd, threadinfo.alias);
            pthread_mutex_lock(&clientlist_mutex);
//delete client
            list_delete(&client_list, &threadinfo);
            pthread_mutex_unlock(&clientlist_mutex);
            break;
         }
         else
         {//problem,clean socket and thread
            fprintf(stderr, "Garbage data from [%d] %s...\n", threadinfo.sockfd,
threadinfo.alias);
         }
      }

      /* clean up socket */
      close(threadinfo.sockfd);

      return NULL;
}
```