Πανεπιστήμιο Πατρών
Τμήμα Μηχανικών Η/Υ & Πληροφορικής
Ακαδημαϊκό έτος 2012-2013

(ΒΑΘΜΟΣ ΆΣΚΗΣΗΣ: 10)

Εργαστήριο Δικτύων Υπολογιστών
Αναφορά 2ης Άσκησης

Αφροδίτη Αλεβιζοπούλου ΑΜ: 3879

Ομάδα NETLAB 39

1 Ανάλυση κώδικα – Σχεδιαστικές αποφάσεις

Ο ρόλος του server είναι να εκτελείται στο παρασκήνιο και να εξυπηρετεί τους clients που συνδέονται σ'αυτόν. Στην εφαρμογή μας ο server δημιουργεί ένα νέο socket με τη χρήση της *socket()* και το συνδέει με μια τοπική θύρα μέσω της *bind()*. Έπειτα περιμένει για νέες συνδέσεις μέσω της *listen()* και ξεκινάει έναν ατέρμον βρόγχο στον οποίο μέσω της κλήσης συστήματος *accept*() δέχεται συνεχώς νέες συνδέσεις. Σε κάθε νέα σύνδεση που πραγματοποιείται καλείται η *fork()*, η οποία δημιουργεί μια νέα διεργασία ή αλλιώς ένα πιστό αντίγραφο του server που αναλαμβάνει να εξυπηρετήσει τον client που συνδέθηκε. Έτσι, είναι δυνατό να εξυπηρετούνται πολλοί clients ταυτόχρονα. Στη συνέχεια μπαίνει σε ένα νέο ατέρμον loop στο οποίο μέσω της *read()* δέχεται συνεχώς εντολές από τον client. Ανάλογα με την εντολή εκτελείται η ανάλογη if η οποία ξεκινά την ανάλογη συνάρτηση που υλοποιεί τη λειτουργικότητα της εντολής ή εμφανίζει κάποιο μήνυμα λάθους αν κάτι δεν έγινε σωστά.

Στην άσκησή μας ως εφαρμογή που θα πραγματοποιεί τις αιτήσεις προς τον εξυπηρετητή χρησιμοποιείται η υπηρεσία *telnet*. Γίνεται telnet στη θύρα που ακούει ο εξυπηρετητής και στη συνέχεια υποβάλλεται η αίτηση. Τα δεδομένα της απάντησης επιστρέφονται και εμφανίζονται στο telnet παράθυρο του client.

Για όλη την επικοινωνία μεταξύ server και client (telnet) μέσω μηνυμάτων χρησιμοποιήθηκαν οι συναρτήσεις *read() – write()*.

Να σημειώσουμε πως η υπηρεσία telnet προσθέτει \r \n στο τέλος κάθε string. Αφού το μήνυμα του client έχει διαβαστεί, πάμε στο τέλος του string και όσο έχει \r \n τα αντικαθιστούμε με \0. Το string είναι πλέον έτοιμο να δοθεί σαν είσοδος στη συνάρτηση strtok(), η οποία θα το χωρίσει σε δύο tokens βάσει του κενού χαρακτήρα. Το $1^{ο}$ token θα είναι το *Service (res_name/ res_ip/ disc_serv)* και το $2^{ο}$ token θα είναι αντίστοιχα το *hostname/ hostipaddress/ knownservice*. Το $1^{ο}$ token θα καθορίσει ποια συνάρτηση απ΄τις παρακάτω θα κληθεί:

- *void parse_name(char *name, int newsockfd)*
- *void parse_ip(char *ip, int newsockfd)*
- *void parse_service(char *servname, int newsockfd)*

Εντός των συναρτήσεων αυτών καλούνται αντίστοιχα οι gethostbyname()/ gethostbyaddr()/ getservbyname(), μέσω των οποίων βρίσκουμε και επιστρέφουμε όλα τα σχετικά στοιχεία.

Για περισσότερες λεπτομέρειες σχετικά με τις συναρτήσεις μπορείτε να δείτε τα σχόλια που περιέχονται στον κώδικα.

Όπως αναφέραμε, είναι σημαντικό να γίνονται οι κατάλληλοι έλεγχοι για την εντιμετώπιση τόσο των «συντακτικών» όσο και των «λειτουργικών» λαθών στα μηνύματα που δέχεται ο εξυπηρετητής:

- Σε όλες τις περιπτώσεις *«συντακτικού»* λάθους (π.χ. να δοθεί μια εντολή της μορφής *res_ipp diogenis.ceid.upatras.gr*) ο εξυπηρετητής ενημερώνει τον client με το μήνυμα λάθους:

    - *"Give the correct parameters!"*

- Για την περίπτωση όπου ο client δώσει μόνο το όνομα του Service (π.χ. *res_name/ res_ip/ disc_serv*) χωρίς να δώσει αντίστοιχα κάποιο *hostname/ hostipaddress/ knownservice*, ο εξυπηρετητής τον ενημερώνει με το αντίστοιχο μήνυμα λάθους:

    - *"You forgot to give a hostname. Try again!"* ή
    - *"You forgot to give an ip address. Try again!"* ή
    - "*You forgot to give a service name. Try again!"*

- Για την περίπτωση όπου ο client πατάει απλώς enter, χωρίς να στέλνει κάποια εντολή στον εξυπηρετητή, ο εξυπηρετητής διαβάζει το \r και κάνει *continue*, επιστρέφει δηλαδή κατευθείαν στη read περιμένοντας την επόμενη εντολή.

- Όλες οι περιπτώσεις *«λειτουργικού»* λάθους (π.χ. να μην υπάρχει ο ζητούμενος hostname) αντιμετωπίζονται εντός των συναρτήσεων που υλοποιούν τη λειτουργικότητα της εκάστοτε εντολής (*parse_name()*, *parse_ip(), parse_service()*). Τα αντίστοιχα μηνύματα λάθους είναι:

    - *"gethostbyname() failed. Error message: ……."* ή
    - *"gethostbyaddr() failed. Error message: ……."* ή
    - *"getservbyname() failed. Unknown application: ……."*

Ο πελάτης μπορεί να διακόψει τη σύνδεση είτε πληκτρολογώντας την εντολή *exit* είτε πληκτρολογώντας τον escape character του telnet και μετά quit. Ο server από την πλευρά του τελειώνει την εξυπηρέτηση του πελάτη κάνοντας *close(newsockfd) και exit(0)*. Με το signal(SIGCHLD, SIG_IGN); αγνοούμε τα signals από τις διεργασίες-παιδιά αφού στο τέλος τερματίζουμε έτσι κι αλλιώς όλες τις διεργασίες ώστε να μην υπάρξουν zombie child processes.

Επίσης, έχουμε ορίσει και ως handler για τα σήματα SIGTERM και SIGINT τη συνάρτηση:
void toDie(int signal). Πληκτρολογώντας Ctrl+C στο terminal του server τυπώνεται το μήνυμα “Die!” και το πρόγραμμα τερματίζει κλείνοντας το socket που μας επέστρεψε η socket().

2 Οδηγίες για τη μεταγλώτισση του κώδικα

Για να κάνουμε compile στο πρόγραμμά μας κάνουμε τα παρακάτω:

1. Ανοίγουμε το terminal.
2. Μεταβαίνουμε μέσω της εντολής cd στο φάκελο server.
3. Γράφουμε make στο terminal *ή* πληκτρολογούμε gcc –g –Wall server.c –o server

Αφού γίνει το compile τρέχουμε τον server γράφοντας στο terminal:

./server –p <port>

, όπου port είναι η θύρα στην οποία θέλουμε να δέχεται συνδέσεις ο server. Η ομάδα μας είναι η 39, οπότε οι θύρες που έχουμε στη διάθεσή μας είναι δέκα, ξεκινώντας από την [9000 + (39-1) * 10] = 9380, οπότε μπορούμε να χρησιμοποιήσουμε τις 9380-9389. Συνεπώς, μια πιθανή εκτέλεση θα ήταν:

./server –p 9385

Να σημειωθέι εδώ πως αν δε δώσουμε παράμετρο στην εντολή εκτέλεσης (./server), ο server θα εκτελεστεί κανονικά και θα δέχεται συνδέσεις σε μια προκαθορισμένη θύρα, την 9380.

Αν ο server εκτελεστεί κανονικά, θα εμφανιστεί στο terminal του server το μήνυμα:

"To socket dhmiourgh8hke epityxws!"

Σε ένα δεύτερο terminal γίνεται telnet στη θύρα που ακούει ο εξυπηρετητής (π.χ. στο διογένη: telnet 150.140.141.181 9385). Στο terminal του server θα εμφανιστεί το μήνυμα:

"Connected! New socket: ……"

Στη συνέχεια υποβάλλεται η αίτηση του client με τη μορφή που μας έχει ζητηθεί. Τα δεδομένα της απάντησης επιστρέφονται και εμφανίζονται στο telnet παράθυρο του client.