Δίκτυα Υπολογιστών – Άσκηση 1

Ξεκινώντας με τον serv1 έχουμε ένα σχετικά μικρής πολυπλοκότητας μοντέλο iterative server. Στους επόμενους servers δε θα αναλυθούν τόσο ενδελεχώς ίδιες εντολές με τον serv1.

```
#include "libf.h"
#include "keyvalue.h"
```

Αρχικά περιλαμβάνονται με εντολές προεπεξεργαστή δύο προσαρμοσμένες βιβλιοθήκες:

Η πρώτη είναι η libf.h που περιέχει:

- Βιβλιοθήκες απαραίτητες για την υποστήριξη επικοινωνίας μέσω δικτύου, εισόδου/εξόδου, χειρισμό σημάτων, κ.ά.
- Ρουτίνες απαραίτητες για την ανάγνωση και εγγραφή σε sockets καθώς και άλλες ρουτίνες που επιστρέφουν το μέγιστο μέγεθος του buffer αποστολής/λήψης.

Η δεύτερη είναι η keyvalue.h η οποία όπως ζητήθηκε περιέχει ρουτίνες τοποθέτησης και αφαίρεσης των ζευγών key – value από τη μνήμη του server, καθώς και τη δομή αποθήκευσης των ζευγών σε μορφή διασυνδεδεμένης λίστας.

```
extern struct data *pHead;
```

Έπειτα δηλώνεται η αρχή της διασυνδεδεμένης λίστας.

```
int main(int argc, char **argv) {
char request[65535], *ch, *ch2, *fin;
```

Στη συνέχεια, στη main, ορίζουμε τις μεταβλητές char που θα χρειαστούμε: Η request είναι στην πραγματικότητα ο buffer για τη λήψη και αποστολή δεδομένων από και προς το socket. Οι δε ch, ch2 αφορούν στο χειρισμό, τη μορφοποίηση των συμβολοσειρών και ως character iterators για την ανάγνωση των ληφθέντων εντολών από τον client. `int sParent, sChild;`

Οι sParent και sChild είναι socket descriptors, όπου ο πρώτος χειρίζεται τις εισερχόμενες αιτήσεις για σύνδεση και ο δεύτερος αναλαμβάνει την επικοινωνία του server με τον client.

```
socklen_t clients_queue; struct
sockaddr_in cliaddr, servaddr;
```

Οι δομές τύπου sockaddr_in και socklen_t ορίζονται με σκοπό τη μετέπειτα δημιουργία του socket και τις απαραίτητες αρχικοποιήσεις για την επικοινωνία των server και client.

```
 pHead = NULL;
sParent = socket(AF_INET, SOCK_STREAM, 0); servaddr.sin_family
= AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(atoi(argv[1]));
```

Ο pHead αρχικοποείται και στον sParent εκχωρείται ο αριθμός του socket που θέλουμε να δημιουργήσουμε. Στις επόμενες εκχωρήσεις ορίζονται η οικογένεια του πρωτοκόλλου επικοινωνίας, οι διευθύνσεις και το port που ακούει ο server σε network byte ordering.

```
bind(sParent, (struct sockaddr *) &servaddr, sizeof(struct sockaddr_in)); listen(sParent,
5);
```

Η bind δεσμεύει το socket του server και η listen σηματοδοτεί την έναρξη αποδοχής εισερχόμενων συνδέσεων.

```
while (1) {
clients_queue = sizeof(cliaddr); sChild = accept(sParent, (struct
sockaddr *) &cliaddr, &clients_queue);
```

Όσο είναι ενεργός ο server: στο sChild εκχωρείται η πρώτη στην ουρά εισερχόμενη σύνδεση. `while (readKey(sChild, request) > 0) { fin = request;`

Όσο η ανάγνωση είναι δυνατή στο socket: Η fin παίρνει το offset των ληφθέντων δεδομένων στο request.

```
do { ch = fin + 1; if
(*fin == 'p') { ch2 = ch +
strlen(ch) + 1; put(ch,
ch2);
fin = ch2 + strlen(ch2) + 1; }
```

Επανάλαβε τα εξής: Η ch δείχνει στο key της εντολής.
Αν το opcode της εντολής είναι ‘p’ τότε: η ch2 δείχνει στο επόμενο string (το value) της εντολής.
Τοποθέτησε τα ch και ch2 στη μνήμη του server.
Θέσε στη fin το offset για την επόμενη προς ανάγνωση εντολή.

```
else if (*fin == 'g') {
ch2 = get(ch); if (ch2
!= NULL) { *(--ch2) =
'f';
writeKey(sChild, ch2, strlen(ch2)); }
```

Αν το opcode της εντολής είναι ‘g’:

Εκχώρησε στη ch2 το offset του value που επιστρέφει η get(ch), όπου ch είναι το ζητούμενο κλειδί.
Αν η ch2 δεν είναι κενή τότε:
Τοποθέτησε ένα χαρακτήρα ‘f’ μπροστά από τη συμβολοσειρά που δείχνει η ch2.
Στείλε στον client τη νέα συμβολοσειρά.

```
else {
writeChar(sChild, 'n'); }
```

Διαφορετικά απόστειλε στον client το χαρακτήρα ‘f’.

```
fin = ch + strlen(ch) + 1; }
```

Θέσε στη fin το offset για την επόμενη προς ανάγνωση εντολή.

```
else {
close(sChild); }
```

Αν ληφθεί κάτι που δε συμπεριλαμβάνεται στα γνωστά opcode τερμάτισε τη σύνδεση.

```
} while (*fin == 'p' || *fin == 'g');
}
```

Η επανάληψη γίνεται όσο το opcode της επόμενης προς εκτέλεση εντολής είναι ‘p’ ή ‘g’.
Τέλος επανάληψης της συνθήκης με το readKey.

```
close(sChild); memset(request, 0, 65535);
```

Κλείσιμο σύνδεσης εφ’όσων δεν υπάρχει κάτι άλλο προς ανάγνωση από το socket.
Μηδενισμός του buffer.

```
close(sParent);
delKeys(); return
0; }
```

Όταν βγούμε από τη συνθήκη while(1) (!):
Κλείσιμο του αρχικού socket του server.
Αποδέσμευση της μνήμης.

Για τον serv2 προστίθενται τα εξής:

Γραμμή 6:

```
void sig_chld(int signo) {
       while (waitpid(-1, NULL, WNOHANG) > 0) {}
}
```

Γραμμή 26:

```
signal(SIGCHLD, sig_chld);
```

Διαχείριση από το γονέα σήματος τερματισμού διεργασίας παιδιού.

Γραμμή 30:

```
pid = fork();
              if (pid == 0) {
```

Παραγωγή διεργασίας παιδιού.

Γραμμή 62: `exit(0);`

Τερματισμός διεργασίας παιδιού.

Γενικά, server σε κάθε νέα σύνδεση αναθέτει μία νέα διεργασία.

Στον serv3 προστίθενται σημαφόροι για την ανάθεση εισερχομένων συνδέσεων σε preforked processes μόνον όταν αυτές οι διεργασίες βρίσκονται στην κρίσιμη περιοχή. Στη συνέχεια η κάθε διεργασία αναλαμβάνει την επικοινωνία με τον client.

Στον serv4 για κάθε αποδοχή νέας εισερχόμενης σύνδεσης από το πρόγραμμα δημιουργείται ένα thread το οποίο αναλαμβάνει να φέρει σε πέρας τις εντολές που αποστέλει ο client.