Προγραμματισμός Δικτύων (Ε-01)

1η Προγραμματιστική Εργασία

Δημιουργία βασικής βιβλιοθήκης δικτυακού προγραμματισμού και χρήση της για την υλοποίηση απλού πελάτη και εξυπηρετητή ανταλλαγής αρχείων

Διδάσκων: Νίκος Ντάρμος
Σχολή Πληροφορικής, Πανεπιστήμιο Ιωαννίνων
Ιωάννινα, Νοέμβριος 2010

1 Γενικά

Η εργασία αυτή έχει ως στόχο την εισαγωγή των φοιτητών σε έννοιες δικτυακού προγραμματισμού και στην προγραμματιστική διεπαφή των BSD sockets. Στην εργασία αυτή καλείσθε να υλοποιήσετε:

1. Μία βασική προγραμματιστική διεπαφή (API) συναρτήσεων δικτυακού προγραμματισμού.
2. Έναν βασικό εξυπηρετητή αποστολής/λήψης αρχείων.
3. Ένα ζεύγος βασικών πελατών αποστολής/λήψης αρχείων.

Βαρύτητα επίσης δίνεται στη μεταφερσιμότητα του κώδικα και στη δυνατότητα λειτουργίας του σε διαφορετικά περιβάλλοντα (αρχιτεκτονικές, λειτουργικά συστήματα, κτλ.) και με διαφορετικά πρωτόκολλα (TCP, UDP).

1.1 Παράδοση

Ως καταληκτική ημερομηνία παράδοσης της εργασίας σας ορίζεται η Κυριακή, 5 Δεκεμβρίου 2010 (23:59:59), ενώ η παράδοση θα είναι μέσω e-mail στο `ntarmos@cs.uoi.gr`. Όπως συζητήθηκε και στο πρώτο μάθημα[1], θα πρέπει να παραδώσετε:

1. Τα αρχεία πηγαίου κώδικα.
2. Ένα απλό Makefile.
3. Αναφορά/σχολιασμό (σε μορφή txt/ps/pdf).

Για οποιαδήποτε απορία ή πρόβλημα μπορείτε να επικοινωνείτε μαζί μου στο παραπάνω e-mail ή στο τηλέφωνο 2651008866 ή κατ' ιδίαν τις ώρες γραφείου που έχουμε συζητήσει στο μάθημα. Υπάρχει ακόμα και η δυνατότητα επικοινωνίας μέσω προγραμμάτων IM· ρωτήστε με για λεπτομέρειες.

2 Βασική βιβλιοθήκη

Η βασική σας βιβλιοθήκη θα αποτελείται από συναρτήσεις οι οποίες θα «κρύβουν» τις λεπτομέρειες υλοποίησης και τις κλήσεις συστήματος και θα προσφέρει στον χρήστη της μία ενιαία πλατφόρμα ανάπτυξης εφαρμογών. Το API που θα υλοποιεί η βιβλιοθήκη σας αποτελείται από δομές δεδομένων και συναρτήσεις και περιγράφεται παρακάτω. Για τον τρόπο με τον οποίο μπορείτε να δημιουργήσετε μια βιβλιοθήκη λογισμικού σε σύστημα UNIX την οποία να μπορείτε να συνδέετε (link) με τον κωδικά σας κατά την μετάφρασή του, δείτε το παράρτημα Α'.

Το API ορίζει συναρτήσεις για δημιουργία, σύνδεση και καταστροφή άκρων επικοινωνίας, για αποδοχή αιτήσεων σύνδεσης, καθώς και για ανταλλαγή δεδομένων. Επίσης, για την αποθήκευση και διαχείριση των πληροφοριών των sockets, το API χρησιμοποιεί μία δική του δομή δεδομένων. Στο σχήμα 1 φαίνεται συνολικά το header file το οποίο ορίζει το API αυτό. Τα ορίσματα και η λειτουργικότητα των συναρτήσεων καθορίζονται παρακάτω.

Στη φάση αυτή απαιτείται η υποστήριξη των πρωτοκόλλων TCP και UDP στο επίπεδο μεταφοράς και των πρωτοκόλλων IPv4 και IPv6 στο επίπεδο διαδικτύου. Όπως φαίνεται και από το σχήμα 1, ούτε οι δομές δεδομένων αλλά ούτε και οι συναρτήσεις έχουν κάποιο πεδίο/όρισμα για το αν η υλοποίηση είναι σε IPv4 ή IPv6, οπότε θα πρέπει να λειτουργούν πάνω από οποιοδήποτε δίκτυο αυτών των τύπων. Η περιγραφή των δομών δεδομένων και των συναρτήσεων που ακολουθεί καθορίζει την ελάχιστη λειτουργικότητα του API· είστε ελεύθεροι να προσθέσετε και

[1] `http://www.cs.uoi.gr/~ntarmos/Courses/NetworkProgramming/Lectures/Lecture01.pdf`

Σχήμα 1: myNetLib.h

```
1  #ifndef __MY_NET_LIB_H__
2  #define __MY_NET_LIB_H__
3
4  typedef enum {
5     TCPEndpoint,          // TCP endpoint.
6     UDPEndpoint,          // UDP endpoint.
7  } EndpointType;
8
9  typedef struct {
10    EndpointType       type;     // Type of the endpoint.
11    int                backlog;  // For TCP server sockets.
12    int*               sd;       // Table of socket descriptors.
13    int                sdlen;    // # elements in sd.
14    struct addrinfo*   addr;     // Info for the above descriptors.
15 } EndpointInfo;
16
17 int createServerEndpoint(const char *host, const char *service,
      EndpointInfo *info);
18 int createClientEndpoint(const char *host, const char *service,
      EndpointInfo *info);
19 int closeEndpoint(EndpointInfo *info);
20 int getNextClientFromEndpoint(const EndpointInfo *serverInfo,
      EndpointInfo *clientInfo);
21 int sendDataToEndpoint(const EndpointInfo *info, const void *data, size_t
       datalen);
22 int recvDataFromEndpoint(const EndpointInfo *info, void *data, size_t
      datalen);
23
24 #endif
```

δικές σας βοηθητικές/εσωτερικές συναρτήσεις ή/και επιπλέον πεδία στις δομές δεδομένων, αν θεωρείτε ότι αυτό είναι απαραίτητο, κάτι που θα πρέπει και να τεκμηριώσετε στην αναφορά σας.

2.1 `createServerEndpoint(...)`

Η συνάρτηση αυτή θα δημιουργεί τα άκρα επικοινωνίας στα οποία η διεργασία του εξυπηρετητή θα δέχεται εισερχόμενες αιτήσεις σύνδεσης.

Τα ορίσματα είναι ως εξής:

1. `host`: Η διεύθυνση IP ή το συμβολικό όνομα στο οποίο θα αναμένουμε εισερχόμενες συνδέσεις, σε ASCII αναπαράσταση (π.χ. «zeus.cs.uoi.gr», «195.130.121.11» κτλ). Αν το πεδίο αυτό έχει την τιμή `NULL` τότε θα πρέπει η αντιστοίχηση να γίνεται με όλες τις διευθύνσεις του κόμβου στον οποίο εκτελείται η διεργασία του εξυπηρετητή («passive socket»).
2. `service`: Το συμβολικό όνομα υπηρεσίας ή ο αριθμός θύρας στην οποία θα αναμένουμε εισερχόμενες συνδέσεις, σε ASCII αναπαράσταση (π.χ. «http», «80» κτλ.). Το πεδίο αυτό δεν μπορεί να έχει την τιμή `NULL`.
3. `info`: Δείκτης σε δομή τύπου `EndpointInfo`. Όταν κληθεί η συνάρτηση θα πρέπει

Σχήμα 2: TCP createServerEndpoint example

```
1  int err;
2  EndpointInfo tcpInfo;
3  // ...
4  tcpInfo.type = TCPEndpoint;
5  tcpInfo.backlog = 5;
6  err = createServerEndpoint("zeus.cs.uoi.gr", "http", &tcpInfo);
```

Σχήμα 3: UDP createServerEndpoint example

```
1  int err;
2  EndpointInfo udpInfo;
3  // ...
4  udpInfo.type = UDPEndpoint;
5  err = createServerEndpoint("gaia.cs.uoi.gr", "domain", &udpInfo);
```

το πεδίο `type` να έχει τον τύπο του άκρου επικοινωνίας που θέλουμε να δημιουργηθεί. Όταν η συνάρτηση επιστρέψει, στο πεδίο `addr` θα έχουν αποθηκευθεί οι πληροφορίες για το άκρο επικοινωνίας που θα δημιουργήσει η συνάρτηση αυτή. Η συνάρτηση απαιτεί ο δείκτης αυτός να δείχνει σε ισχύουσα θέση μνήμης και δε θα δεσμεύει αυτή το χώρο μνήμης για τη δομή αυτή. Ωστόσο θα είναι υπεύθυνη να δεσμεύει χώρο για τον πίνακα `sd` καθώς και την δομή `addr`. Ο χώρος αυτός θα πρέπει να αποδεσμεύεται όταν κληθεί η `closeEndpoint(...)`.

Η συνάρτηση αυτή θα πρέπει:

- να μεταφράζει τα ορίσματα `host` και `service` σε δυαδική μορφή, κατάλληλη για χρήση από τις κλήσεις συστήματος του API των BSD sockets.
- να δεσμεύει τον απαραίτητο χώρο για τα πεδία `sd` και `addr` της δομής στην οποία δείχνει ο δείκτης `info`.
- να εκτελεί κατάλληλες κλήσεις στις συναρτήσεις `getaddrinfo(3)`, `socket(2)`, `bind(2)` και `listen(2)`, ανάλογα με τον ζητούμενο τύπο άκρου επικοινωνίας. Στην περίπτωση που πρέπει να κληθεί και η `listen(2)`, το μέγεθος της ουράς αναμονής θα βρίσκεται στο πεδίο `backlog` της δομής στην οποία δείχνει ο `info`, αλλιώς το πεδίο αυτό θα αγνοείται.

Σε αυτό το στάδιο της υλοποίησης, στην περίπτωση που ο κόμβος στον οποίο θα εκτελεστεί η συνάρτηση αυτή έχει παραπάνω από μία διευθύνσεις ή/και υποστηρίζει και IPv4 και IPv6, θα επιλέγεται μόνο μία εκ των διευθύνσεων (δηλ. ο πίνακας `sd` θα έχει το πολύ μία θέση)· συγκεκριμένα, θα επιλέγεται η πρώτη διεύθυνση στην λίστα που επιστρέφει η `getaddrinfo(3)`.

Η συνάρτηση επιστρέφει 0 όταν πέτυχε να δημιουργήσει έστω ένα άκρο επικοινωνίας ή -1 για αποτυχία.

Ο κώδικας στα σχήματα 2 και 3 δίνει ένα παράδειγμα χρήσης της συνάρτησης αυτής για συνδέσεις πρωτοκόλλου TCP και UDP αντίστοιχα.

2.2 `createClientEndpoint(...)`

Η συνάρτηση αυτή θα δημιουργεί μία σύνδεση ανάμεσα στον πελάτη που την καλεί και τον εξυπηρετητή που της ορίζεται, μέσω της οποίας ο πελάτης θα μπορεί να ανταλλάξει κατόπιν

Σχήμα 4: TCP createClientEndpoint example

```
1  int err;
2  EndpointInfo tcpInfo;
3  // ...
4  tcpInfo.type = TCPEndpoint;
5  err = createClientEndpoint("zeus.cs.uoi.gr", "http", &tcpInfo);
```

Σχήμα 5: UDP createClientEndpoint example

```
1  int err;
2  EndpointInfo udpInfo;
3  // ...
4  udpInfo.type = UDPEndpoint;
5  err = createClientEndpoint("gaia.cs.uoi.gr", "domain", &udpInfo);
```

δεδομένα με τον απομακρυσμένο εξυπηρετητή.

Τα ορίσματα είναι ως εξής:

1. `host`: Η διεύθυνση IP ή το συμβολικό όνομα στο οποίο θα συνδεθεί ο πελάτης, σε ASCII αναπαράσταση (π.χ. «zeus.cs.uoi.gr», «195.130.121.11» κτλ). Το πεδίο αυτό δε μπορεί να έχει την τιμή `NULL`.
2. `service`: Το συμβολικό όνομα υπηρεσίας ή ο αριθμός θύρας στην οποία θα συνδεθεί ο πελάτης, σε ASCII αναπαράσταση (π.χ. «http», «80» κτλ.). Το πεδίο αυτό δεν μπορεί να έχει την τιμή `NULL`.
3. `info`: Δείκτης σε δομή τύπου `EndpointInfo`. Όταν κληθεί η συνάρτηση θα πρέπει το πεδίο `type` να έχει τον τύπο του άκρου επικοινωνίας που θέλουμε να δημιουργηθεί. Όταν η συνάρτηση επιστρέψει, στο πεδίο `addr` θα έχουν αποθηκευθεί οι πληροφορίες για την σύνδεση που δημιούργησε η συνάρτηση αυτή. Η συνάρτηση απαιτεί ο δείκτης αυτός να δείχνει σε ισχύουσα θέση μνήμης και δε θα δεσμεύει αυτή το χώρο μνήμης για τη δομή αυτή. Ωστόσο θα είναι υπεύθυνη να δεσμεύει χώρο για τον πίνακα `sd` καθώς και την δομή `addr`. Ο χώρος αυτός θα πρέπει να αποδεσμεύεται όταν κληθεί η `closeEndpoint(...)`.

Η συνάρτηση αυτή θα πρέπει:

- να μεταφράζει κατάλληλα τα ορίσματα `host` και `service` σε δυαδική μορφή, κατάλληλη για χρήση από τις κλήσεις συστήματος του API των BSD sockets.
- να δεσμεύει τον απαραίτητο χώρο για τα πεδία `sd` και `addr` της δομής στην οποία δείχνει ο δείκτης `info`.
- ανάλογα με τον ζητούμενο τύπο άκρου επικοινωνίας, να εκτελεί κατάλληλες κλήσεις στις συναρτήσεις `getaddrinfo(3)`, `socket(2)` και `connect(2)`. Το πεδίο `backlog` της δομής στην οποία δείχνει ο `info` αγνοείται.

Στην περίπτωση που ο εξυπηρετητής έχει παραπάνω από μία διευθύνσεις ή/και υποστηρίζει και IPv4 και IPv6, η συνάρτηση θα πρέπει να προσπαθήσει να δημιουργήσει το πολύ μία σύνδεση (δηλ. ο πίνακας `sd` θα έχει το πολύ μία θέση).

Η συνάρτηση επιστρέφει 0 όταν πέτυχε να δημιουργήσει έστω μία σύνδεση με την απομακρυσμένη διεργασία ή -1 για αποτυχία.

Ο κώδικας στα σχήματα 4 και 5 δίνει ένα παράδειγμα χρήσης της συνάρτησης αυτής για συνδέσεις πρωτοκόλλου TCP και UDP αντίστοιχα.

Σχήμα 6: TCP getNextClientFromEndpoint example

```
1  int err;
2  EndpointInfo serverInfo, clientInfo;
3  // ...
4  serverInfo.type = TCPEndpoint;
5  serverInfo.backlog = 5;
6  err = createServerEndpoint(NULL, "http", &serverInfo);
7  // ...
8  err = getNextClientFromEndpoint(&serverInfo, &clientInfo);
```

2.3 `closeEndpoint(...)`

Η συνάρτηση αυτή θα κλείνει το άκρο επικοινωνίας που δημιουργήθηκε από αντίστοιχη κλήση μίας εκ των `createServerEndpoint(...)` και `createClientEndpoint(...)` και θα απελευθερώνει τους πόρους που αυτή καταλάμβανε.

Τα ορίσματα είναι ως εξής:

1. `info`: Δείκτης σε δομή τύπου `EndpointInfo`. Η συνάρτηση απαιτεί ο δείκτης αυτός να δείχνει σε ισχύουσα θέση μνήμης.

Η συνάρτηση αυτή θα πρέπει:

- να εκτελεί κατάλληλες κλήσεις στις συναρτήσεις `close(2)` και `freeaddrinfo(3)`.
- να απελευθερώνει την μνήμη που καταλαμβάνουν τα πεδία `sd` και `addr` της δομής δεδομένων στην οποία δείχνει ο `info`.

Η συνάρτηση επιστρέφει 0 για επιτυχία και -1 για αποτυχία.

2.4 `getNextClientFromEndpoint(...)`

Η συνάρτηση αυτή θα καλείται από εξυπηρετητές τόσο συνδεσμοστραφούς όσο και ασυνδεσμικής επικοινωνίας. Με την συνάρτηση αυτή οι πρώτοι θα αποδέχονται την αίτηση σύνδεσης από τον επόμενο πελάτη στη λίστα αναμονής του άκρου επικοινωνίας τους. Για τους δεύτερους, η συνάρτηση αυτή θα μπλοκάρει μέχρι να έρθουν δεδομένα στο άκρο επικοινωνίας του εξυπηρετητή.

Τα ορίσματα είναι ως εξής:

1. `server`: Δείκτης σε δομή τύπου `EndpointInfo`. Η συνάρτηση απαιτεί ο δείκτης αυτός να δείχνει σε ισχύουσα θέση μνήμης. Η δομή αυτή αντιστοιχεί στην πληροφορία του άκρου επικοινωνίας του εξυπηρετητή και θα πρέπει να έχει προηγουμένως αρχικοποιηθεί με κατάλληλη κλήση της `createServerEndpoint(...)`.
2. `client`: Δείκτης σε δομή τύπου `EndpointInfo`. Η συνάρτηση απαιτεί ο δείκτης αυτός να δείχνει σε ισχύουσα θέση μνήμης. Η δομή αυτή αντιστοιχεί στην πληροφορία του νέου socket που θα δημιουργηθεί για να συνδέσει τον εξυπηρετητή με τον επόμενο πελάτη. Όταν η συνάρτηση επιστρέψει, η δομή αυτή θα πρέπει να έχει όλη την απαραίτητη πληροφορία για να μπορεί κατόπιν ο εξυπηρετητής να ανταλλάξει δεδομένα με τον επόμενο πελάτη.

Η συνάρτηση αυτή θα πρέπει:

- να δεσμεύει τον απαραίτητο χώρο για τα πεδία `sd` και `addr` της δομής στην οποία δείχνει ο δείκτης `client`.

- να εκτελεί κατάλληλες κλήσεις στη συνάρτηση `accept(2)` ή `recvfrom(2)`, ανάλογα με το ζητούμενο τύπο άκρου επικοινωνίας.

Η συνάρτηση θα επιστρέφει 0 για επιτυχία και -1 για αποτυχία.

Ο κώδικας στο σχήμα 6 δίνει ένα παράδειγμα χρήσης της συνάρτησης αυτής για συνδέσεις πρωτοκόλλου TCP.

2.5 `sendDataToEndpoint(...)` και `recvDataFromEndpoint(...)`

Οι συναρτήσεις αυτές θα χρησιμοποιούνται τόσο από εξυπηρετητές όσο και από πελάτες για να ανταλλάξουν μεταξύ τους δεδομένα.

Τα ορίσματα είναι ως εξής:

1. `info`: Δείκτης σε δομή τύπου `EndpointInfo`. Η συνάρτηση απαιτεί ο δείκτης αυτός να δείχνει σε ισχύουσα θέση μνήμης. Η δομή αυτή αντιστοιχεί στην πληροφορία του άκρου επικοινωνίας στο οποίο θα στείλουμε ή από το οποίο θα λάβουμε δεδομένα καλώντας την `sendDataToEndpoint(...)` ή την `recvDataFromEndpoint(...)` αντίστοιχα.
2. `data`: Δείκτης σε θέση μνήμης όπου βρίσκονται τα δεδομένα που θα στείλουμε ή στον οποίο θα γραφούν τα δεδομένα που θα λάβουμε. Οι συναρτήσεις αυτές απαιτούν ο δείκτης αυτός να δείχνει σε ισχύουσα θέση μνήμης και δε θα δεσμεύουν αυτές τον αντίστοιχο χώρο.
3. `datalen`: Μέγεθος σε bytes του χώρου μνήμης στον οποίο δείχνει ο δείκτης `data`.

Οι συναρτήσεις αυτές θα πρέπει:

- να μετατρέπουν τα δεδομένα από και πρός κάποια ενδιάμεση μορφή της επιλογής σας (σχολιάστε την σχεδίασή σας στην αναφορά σας), ώστε να επιτρέπουν την επικοινωνία ανάμεσα σε υπολογιστές με διαφορετική δυαδική αναπαράσταση (big-endian vs. little-endian). Προφανώς η `sendDataToEndpoint(...)` θα μετατρέπει τα δεδομένα στην ενδιάμεση μορφή, ενώ η `recvDataToEndpoint(...)` θα αντιστρέφει την παραπάνω μετατροπή.
- να εκτελεί κατάλληλες κλήσεις στις συναρτήσεις `read(2)`, `write(2)`, `send(2)`, `recv(2)`, `sendto(2)` και `recvfrom(2)`, ανάλογα με τον τύπο του άκρου επικοινωνίας που περιγράφει η δομή στην οποία δείχνει ο `info` και με το αν στέλνουμε ή λαμβάνουμε δεδομένα.

Οι συναρτήσεις θα επιστρέφουν -1 για αποτυχία ή τον αριθμό των bytes που στάλθηκαν ή λήφθηκαν προς/από το άκρο επικοινωνίας. Για την περίπτωση σύνδεσης UDP, θα πρέπει οι συναρτήσεις αυτές να κατακερματίζουν τα δεδομένα που τους δίνονται σε πακέτα κατάλληλου μεγέθους, ενώ δε θα πρέπει να σας απασχολεί η απώλεια πακέτων κατά τη μεταφορά μεγάλου όγκου δεδομένων καθώς κάτι τέτοιο είναι αποδεκτό για το πρωτόκολλο αυτό.

2.6 Συνολική εικόνα

Στα σχήματα 7 και 8 φαίνεται ένα πλήρες παράδειγμα κώδικα πελάτη και εξυπηρετητή αντίστοιχα οι οποίοι χρησιμοποιούν το παραπάνω API. Ο πελάτης δέχεται ως ορίσματα γραμμής εντολών το συμβολικό όνομα ή την IP διεύθυνση του εξυπηρετητή και το συμβολικό όνομα υπηρεσίας ή τον αριθμό θύρας στα οποία θα συνδεθεί, καθώς και ποιο πρωτόκολλο («`tcp`» ή «`udp`») θα χρησιμοποιήσει. Αντίστοιχα, ο εξυπηρετητής δέχεται ως ορίσματα γραμμής εντολών μόνο το συμβολικό όνομα υπηρεσίας ή τον αριθμό θύρας όπου και θα περιμένει για νέους πελάτες, καθώς και ποιο πρωτόκολλο («`tcp`» ή «`udp`») θα χρησιμοποιήσει.

Παρατηρήστε ότι τόσο ο κώδικας του πελάτη όσο και αυτός του εξυπηρετητή είναι πανομοιότυποι ανεξάρτητα απ' το αν η επικοινωνία γίνεται πάνω από TCP ή UDP και από το αν στο επίπεδο διαδικτύου έχουμε IPv4 ή IPv6 (το μόνο που αλλάζει είναι το πεδίο `server.type`). Ο

κώδικάς σας θα πρέπει να παρέχει όλη την απαιτούμενη λειτουργικότητα ώστε αυτό να είναι εφικτό.

Σχήμα 7: Client example

```c
#include <stdio.h>
#include <string.h>
#include "myNetLib.h"

int main(int argc, char **argv) {
   EndpointInfo server;
   const char data[11] = "0123456789";

   if (argc != 4 || (strcmp(argv[3], "tcp") && strcmp(argv[3], "udp"))) {
      fprintf(stderr, "Usage: %s <host> <service> <tcp|udp>\n", argv[0]);
      return 1;
   }

   server.type = strcmp(argv[3], "tcp") ? UDPEndpoint : TCPEndpoint;
   if (createClientEndpoint(argv[1], argv[2], &server))
      return 1;
   if (sendDataToEndpoint(&server, data, sizeof(data)) != sizeof(data))
      return 1;
   if (closeEndpoint(&server))
      return 1;
   return 0;
}
```

Σχήμα 8: Server example

```c
#include <stdio.h>
#include <string.h>
#include "myNetLib.h"

int main(int argc, char **argv) {
   char buf[11];
   EndpointInfo server, client;

   if (argc != 3 || (strcmp(argv[2], "tcp") && strcmp(argv[2], "udp"))) {
      fprintf(stderr, "Usage: %s <service> <tcp|udp>\n", argv[0]);
      return 1;
   }

   server.type = strcmp(argv[2], "tcp") ? UDPEndpoint : TCPEndpoint;
   server.backlog = 5;
   if (createServerEndpoint(NULL, argv[1], &server))
      return 1;
   while (1) {
      if (getNextClientFromEndpoint(&server, &client))
        return 1;
      memset(buf, 0, sizeof(buf));
      if (recvDataFromEndpoint(&client, buf, sizeof(buf)) < 0)
        return 1;
```

```
24        printf("%s\n", buf);
25        if (closeEndpoint(&client) < 0)
26          return 1;
27      }
28      if (closeEndpoint(&server) < 0)
29        return 1;
30      return 0;
31  }
```

3 Εφαρμογή ανταλλαγής αρχείων

Στο δεύτερο κομμάτι της εργασίας αυτής σας ζητείται να υλοποιήσετε έναν απλό εξυπηρετητή και ένα ζεύγος απλών πελατών ανταλλαγής αρχείων, χρησιμοποιώντας τη βιβλιοθήκη που φτιάξατε προηγουμένως. Η ζητούμενη λειτουργικότητα των προγραμμάτων αυτών περιγράφεται παρακάτω.

3.1 Πελάτες

Θα πρέπει να υλοποιήσετε δύο πελάτες: `download` και `upload`. Ο πρώτος θα μεταφορτώνει ένα αρχείο από τον εξυπηρετητή στον τρέχοντα κατάλογο ενώ ο δεύτερος θα μεταφορτών ένα αρχείο από τον τρέχοντα κατάλογο στον εξυπηρετητή. Και στις δύο περιπτώσεις, το πρόγραμμά σας θα πρέπει να δέχεται ως ορίσματα γραμμής εντολών ένα όνομα κόμβου (συμβολικό ή αριθμητικό), ένα όνομα υπηρεσίας (συμβολικό ή αριθμητικό), προαιρετικά το πρωτόκολλο επιπέδου μεταφοράς («`tcp`» ή «`udp`») που θα χρησιμοποιήσει κατά τη σύνδεσή του με τον απομακρυσμένο εξυπηρετητή, και το όνομα του αρχείου που θα μεταφορτώσει. Αν δεν ορίζεται πρωτόκολλο επιπέδου μεταφοράς, θα εννοείται ως όρισμα το «tcp».

Κατά την εκτέλεσή του, θα πρέπει:

1. Να συνδέεται με τον εξυπηρετητή που ορίζεται από το όνομα κόμβου και το όνομα υπηρεσίας της γραμμής εντολών,
2. Να δημιουργεί ή/και να ανοίγει το αρχείο που ορίστηκε στη γραμμή εντολών,
3. Στην περίπτωση του `upload`, να στέλνει την null-terminated (`'\0'`) συμβολοσειρά «`PUT`» και τα δεδομένα του αρχείου στον εξυπηρετητή.
4. Στην περίπτωση του `download`, να στέλνει την null-terminated συμβολοσειρά «`GET filename`» στον εξυπηρετητή (όπου filename το όνομα του αρχείου που θέλει να μεταφορτώσει), να λαμβάνει τα δεδομένα που του στέλνει ο εξυπηρετητής και να τα γράφει στο τοπικό αρχείο,
5. Να τερματίζει τη σύνδεση και να επιστρέφει:
   (α′) 0, αν η μεταφόρτωση τερματίσει ομαλά.
   (β′) 1, αν συμβεί κάποιο λάθος στην επικοινωνία, τυπώνοντας στο standard error του το κατάλληλο μήνυμα (δείτε τις `gai_strerror(3)` και `perror(3)`), ή αν δεν μετέφερε καθόλου δεδομένα (π.χ. δεν υπήρχε το ζητούμενο αρχείο στον εξυπηρετητή).

3.2 Εξυπηρετητής

Το πρόγραμμά σας θα πρέπει να δέχεται ως ορίσματα γραμμής εντολών το συμβολικό όνομα υπηρεσίας ή τον αριθμό θύρας όπου και θα περιμένει αιτήσεις από πελάτες και προαιρετικά το

πρωτόκολλο επιπέδου μεταφοράς («tcp» ή «udp») μέσω του οποίου θα γίνει η επικοινωνία. Αν δεν ορίζεται πρωτόκολλο θα εννοείται ως όρισμα το «tcp».

Κατά την εκτέλεσή του, ο εξυπηρετητής σας θα πρέπει:

1. Να δεσμεύει την ζητούμενη θύρα σε όλες τις διευθύνσεις του κόμβου στον οποίο εκτελείται,
2. Να περιμένει εισερχόμενες αιτήσεις από πελάτες,
3. Για κάθε τέτοια αίτηση, να διαβάζει τη συμβολοσειρά που του στέλνει ο πελάτης, μέχρι να βρει χαρακτήρα '\0',
4. Για την εντολή «PUT», να δημιουργεί/μηδενίζει/ανοίγει το ζητούμενο αρχείο και να γράφει τα δεδομένα που του στέλνει ο πελάτης σε αυτό.
5. Για την εντολή «GET», να ανοίγει το ζητούμενο αρχείο και να στέλνει τα δεδομένα του στον πελάτη.
6. Αν υπάρξει κάποιο πρόβλημα με το αρχείο (π.χ. δεν υπάρχει, δεν μπορεί να αναγνωστεί, δε μπορεί να δημιουργηθεί, κτλ.), θα πρέπει να κλείνει απευθείας την σύνδεση χωρίς να στέλνει δεδομένα στον πελάτη.
7. Να επανέρχεται στο βήμα 2 σε αναμονή για νέο πελάτη.

Στη φάση αυτή δεν απαιτείται ο εξυπηρετητής σας να έχει τη δυνατότητα παράλληλης επεξεργασίας πολλαπλών αιτήσεων ή την εκτέλεσή του στο παρασκήνιο. Σε περίπτωση που συμβεί κάποιο σφάλμα κατά την επικοινωνία του εξυπηρετητή με κάποιον πελάτη, θα πρέπει να τυπώνει κατάλληλο μήνυμα λάθους στο standard error του και να συνεχίζει με τον επόμενο πελάτη στη λίστα, αφού πρώτα κλείσει τη σύνδεση με τον προηγούμενο πελάτη.

Α΄ Δημιουργία βιβλιοθήκης

Σε περιβάλλον UNIX μπορείτε να «συλλέξετε» διάφορα αρχεία ενδιάμεσου κώδικα (`.o`) και να δημιουργήσετε μία βιβλιοθήκη, την οποία κατόπιν μπορείτε να συνδέετε με τον κώδικά σας κατά τη μετάφρασή του (linking). Αυτό επιτυγχάνεται ως εξής:

1. Οργανώνετε τον κώδικά σας όπως θέλετε σε αρχεία `.c` και `.h`. Έστω ότι η βιβλιοθήκη που σας ζητείται στο πρώτο τμήμα της εργασίας αυτής είναι υλοποιημένη στα αρχεία `createServerEndpoint.c`, `createClientEndpoint.c`, `closeEndpoint.c`, `getNextClientFromEndpoint.c`, `sendDataToEndpoint.c` και `recvDataFromEndpoint.c` και ότι ο πελάτης και ο εξυπηρετητής είναι υλοποιημένοι στα αρχεία `myhttpclient.c` και `myhttpserver.c` αντίστοιχα[2].
2. Μεταφράζουμε τα αρχεία `.c` της υλοποίησης του API σε αντίστοιχα αρχεία ενδιάμεσου κώδικα `.o`, εκτελώντας την εντολή `gcc -c <filename>`.
3. Κατόπιν συλλέγουμε όλα τα αρχεία ενδιάμεσου κώδικα σε ένα «archive» (`lib*.a`), με την εντολή `ar rc <archive> <filenames>`.
4. Δημιουργούμε δομή δεικτοδότησης στο αρχείο που προκύπτει από το προηγούμενο βήμα εκτελώντας την εντολή `ranlib <archive>`.

Η βιβλιοθήκη είναι τώρα έτοιμη προς χρήση. Μπορούμε στη συνέχεια να μεταφράσουμε τα αρχεία του πελάτη και του εξυπηρετητή και να συνδέσουμε τη βιβλιοθήκη, προσθέτοντας στη γραμμή εντολών του μεταφραστή τα ορίσματα `-L. -l<archive>`, όπου από το όνομα του archive έχουμε αφαιρέσει το πρόθεμα `lib`.

Το σχήμα 9 δίνει ένα παράδειγμα των παραπάνω σε περιβάλλον κελύφους. Στη γραμμή 7 μεταφράζουμε όλα τα αρχεία της υλοποίησης του API σε αρχεία ενδιάμεσου κώδικα, τα

[2] Προφανώς κάτι τέτοιο ούτε απαιτείται ούτε ενδείκνυται· χρησιμοποιούμε τη δομή αυτή εδώ χάριν παραδείγματος.

Σχήμα 9: Library creation

```
1  user@host:~/myproject1$ ls
2  MyNetLib.h                    myclient.c
3  closeEndpoint.c               myserver.c
4  createClientEndpoint.c        recvDataFromEndpoint.c
5  createServerEndpoint.c        sendDataToEndpoint.c
6  getNextClientFromEndpoint.c
7  user@host:~/myproject1$ gcc -c *Endpoint.c
8  user@host:~/myproject1$ ls
9  MyNetLib.h                    getNextClientFromEndpoint.o
10 closeEndpoint.c               myclient.c
11 closeEndpoint.o               myserver.c
12 createClientEndpoint.c        recvDataFromEndpoint.c
13 createClientEndpoint.o        recvDataFromEndpoint.o
14 createServerEndpoint.c        sendDataToEndpoint.c
15 createServerEndpoint.o        sendDataToEndpoint.o
16 getNextClientFromEndpoint.c
17 user@host:~/myproject1$ ar rc libmynet.a *.o
18 user@host:~/myproject1$ rm *.o
19 user@host:~/myproject1$ ranlib libmynet.a
20 user@host:~/myproject1$ gcc myclient.c -L. -lmynet -o myclient
21 user@host:~/myproject1$ gcc myserver.c -L. -lmynet -o myserver
22 user@host:~/myproject1$
```

οποία και βλέπουμε με την εντολή της γραμμής 8. Δημιουργούμε το archive με την εντολή της γραμμής 17 και διαγράφουμε τα αρχεία ενδιάμεσου κώδικα με την εντολή της γραμμής 18 αφού δεν τα χρειαζόμαστε άλλο. Με την εντολή στη γραμμή 19 δημιουργούμε μία δομή δεικτοδότησης στο archive η οποία απαιτείται για τη χρήση του από τον μεταφραστή, οπότε και η βιβλιοθήκη είναι πλέον έτοιμη. Τέλος, στις γραμμές 20 και 21 βλέπουμε πως τη χρησιμοποιούμε κατά την μετάφραση των αρχείων του πελάτη και του εξυπηρετητή.