1 RPC Client - Server

1.1 rpcprog.x file

Ένα πολύ βασικό κομμάτι του προγράμματος είναι η επικοινωνία μεταξύ RPC client και server. Το κομμάτι αυτό δημιουργείται αυτόματα μέσω του rpcgen compiler με βάση το αρχείο rpcprog.x, το οποίο περιέχει πληροφορίες για το πόσες συναρτήσεις πρόκειται να υλοποιηθούν και μία ή περισσότερες δομές δεδομένων για την μεταφορά της πληροφορίας. Χρησιμοποιώντας τον rpcgen παράγονται αρχεία που αφορούν το client / server stub module (το οποίο είναι υπεύθυνο για την επικοινωνία) και το client / server application module (το οποίο είναι υπεύθυνο για την υλοποίηση των διεργασιών). Το stub module περιέχει έτοιμο κώδικα και δεν χρειάζεται να κάνουμε καμία αλλαγή πάνω του, ενώ το application module περιέχει έτοιμα templates για τις συναρτήσεις που δηλώσαμε στο ".x" αρχείο. Αυτό που χρειάζεται να προγραμματίσουμε λοιπόν, είναι το είδος της πληροφορίας και πώς αυτή θα επεξεργάζεται από τους client και server. Με βάση τα ζητούμενα της άσκησης χρησιμοποιήθηκαν δύο δομές δεδομένων και τρεις συναρτήσεις (βλ. rpcgen.x). Για την αποστολή των δεδομένων χρησιμοποιείται η δομή "send_data" και για την λήψη τους η "receive_data". Υπάρχουν δύο τρόποι δήλωσης πίνακα στην RPCL (Remote Procedure Call Language). Αυτός που χρησιμοποιήσαμε (Υ<>) μεταφράζει τον πίνακα σε δομή με δύο πεδία.

- **Y_len**: Ακέραιος αριθμός όπου δηλώνει το μέγεθος του διανύσματος.
- **Y_val**: Pointer ακεραίου τύπου.

Κάθε συνάρτηση έχει τον δικό της ξεχωριστό αναγνωριστικό αριθμό (1, 2 ή 3), έκδοση και θεωρείται ξεχωριστό πρόγραμμα.

1.2 rpcprog_server.c

Το αρχείο του server μας περιέχει την υλοποίηση κάθε μίας από τις ζητούμενες συναρτήσεις. Αυτό σημαίνει πως όταν ο RPC client στείλει κάποια δεδομένα και ζητήσει μία από αυτές, τότε ο server επεξεργάζεται κατάλληλα τα δεδομένα, βρίσκει το αποτέλεσμα και το επιστρέφει σε αυτόν. Λίγο πιο αναλυτικά:

- **avg_1_svc(...)**: Λαμβάνει έναν πίνακα ακεραίων (Y_val) μέσω της δομής Υ, τους αθροίζει στην τοπική μεταβλητή result, διαιρεί το άθροισμα τους με το πλήθος των στοιχείων του (Y_len) και αποθηκεύει το αποτέλεσμα πάλι στην result. Τελικά επιστρέφει στον client τη μεταβλητή αυτή.

  Στις επόμενες συναρτήσεις χρησιμοποιείται και η δεύτερη δομή (receive_data) που έχουμε φτιάξει έτσι ώστε να επιστρέψουμε τα δεδομένα στον client. Παρατηρούμε ότι η μεταβλητή result έχει αλλάξει τύπο σχετικά με πριν, λογικό αφού τώρα θέλουμε να επιστρέψουμε δομή και όχι μια απλή μεταβλητή.

- **min_max_1_svc(...)**: Λαμβάνει έναν πίνακα ακεραίων (Y_val) μέσω της δομής Υ, κάνει δυναμική δέσμευση μνήμης (malloc) δύο στοιχείων, επειδή γνωρίζουμε από πριν ότι το αποτέλεσμα θα είναι διάνυσμα δύο ακεραίων, στον πίνακα της δομής int_arr (int_arr_val) και αρχικοποιεί τις δύο προσωρινές μεταβλητές (min, max) με την πρώτη τιμή του πίνακα. Μέσα σε μία for loop η οποία "τρέχει" για όσο ο μετρητής i είναι μικρότερος του μεγέθους του πίνακα (Y_len), ελέγχουμε εάν το επόμενο στοιχείο του διανύσματος είναι μικρότερο ή μεγαλύτερο από αυτό που υπάρχει στην μεταβλητή, αν

είναι, τότε το αποθηκεύουμε στην αντίστοιχη μεταβλητή και συνεχίζουμε τον έλεγχο. Τέλος, δίνουμε αυτές τις τιμές στον πίνακα (int_arr_val) που θα επιστραφεί στον client και στέλνουμε πίσω τα αποτελέσματα.

- **mul_1_svc(...)**: Λαμβάνει έναν πίνακα ακεραίων (Y_val) μέσω της δομής Y και έναν δεκαδικό αριθμό μέσω της μεταβλητής a, επίσης της δομής Y, κάνει δυναμική δέσμευση μνήμης (malloc) τόσων στοιχείων όσων λέει η μεταβλητή Y_len στον πίνακα της δομής double_arr (double_arr_val) και μετά πολλαπλασιάζει όλα τα στοιχεία του διανύσματος με τον δεκαδικό αριθμό, αποθηκεύοντας τα αποτελέσματα στον πίνακα όπου θα επιστραφεί (double_arr_val).

1.3 rpcprog_client.c

Ο RPC client έχει δύο λειτουργίες. Ως socket server για τον socket client και ως RPC client για τον RPC server. Συνεπώς, δέχεται τα δεδομένα από τον socket client και τα μεταβιβάζει στον RPC server χωρίς να τα επηρεάσει ιδιαίτερα. Αρχικά ξεκινάμε με κάποιους ελέγχους και πραγματοποιούμε την δημιουργία του socket στο οποίο θα "ακούει" ο server:

- **argc != 2**: Έλεγχος αν ο χρήστης έχει δώσει λιγότερα ή παραπάνω στοιχεία από τα απαιτούμενα για την χρήση του προγράμματος. Αν ναι, τυπώνει μήνυμα λάθους και τερματίζει.

- **socket(...)**: Δημιουργεί ένα socket και επιστρέφει τον περιγραφέα αρχείου (file descriptor) που αντιστοιχεί σ' αυτό ή -1 σε περίπτωση λάθους. Αν υπάρξει λάθος τυπώνεται σχετικό μήνυμα στην τυπική έξοδο.

- **setsockopt(...)**: Χρησιμοποιείται σε συνδυασμό με την παράμετρο SO_REUSEPORT για την επαναχρησιμοποίηση του port.

- **bind(...)**: "Δένει" μια διεύθυνση και μία πόρτα στο socket που έχουμε πρωτύτερα δημιουργήσει και επιστρέφει "0" σε περίπτωση επιτυχίας και "-1" σε περίπτωση αποτυχίας. Αν αποτύχει τυπώνεται μήνυμα λάθους στην τυπική έξοδο. Η πόρτα που έχουμε ορίσει είναι η "43862".

- **listen(...)**: "Ακούει" τις εισερχόμενες συνδέσεις και επιστρέφει "0" σε περίπτωση επιτυχίας και "-1" σε περίπτωση αποτυχίας. Αν αποτύχει τυπώνεται μήνυμα λάθους στην τυπική έξοδο.

- **accept(...)**: Μπλοκάρει προσωρινά την παρούσα διεργασία μέχρι να υπάρξει κάποια σύνδεση στην ουρά ολοκληρωμένων συνδέσεων. Αμέσως μετά επεξεργάζεται την πρώτη σύνδεση στην ουρά, δημιουργώντας ένα καινούριο συνδεδεμένο socket και επιστρέφει τον περιγραφέα αρχείου αυτού του νέου socket ή "-1" σε περίπτωση αποτυχίας.

  Όπως διαβάσαμε προηγουμένως η accept() μπλοκάρει το κύριο νήμα, για τον λόγο αυτό χρησιμοποιήσαμε threads. Κάθε φορά που συνδέεται κάποιος client, δημιουργείται ένα thread και αποδίδεται σε αυτό η εξυπηρέτηση του. Βάλαμε την accept() μέσα σε έναν ατέρμονο βρόχο έτσι ώστε μετά από κάθε client που εξυπηρετείται, η κύρια διεργασία να περιμένει τον επόμενο. Έχουμε φτιάξει μια δομή (parameters) με τέσσερα πεδία, η οποία χρησιμοποιείται για να περάσουμε παραμέτρους στην συνάρτηση που εκτελούν τα threads. Τα πεδία αυτά αναλύονται παρακάτω:

- **int client**: File descriptor του client. Χρησιμοποιείται για την socket client / server επικοινωνία.
- **int id**: Ξεκινάει από το 0 και αυξάνεται για κάθε thread που δημιουργείται.
- **int *available_IDs**: Δυναμικός πίνακας που περιέχει όλα τα διαθέσιμα IDs των threads που έχουν τερματίσει. Αρχικοποιείται με 10 θέσεις και αν χρειαστεί αυξάνεται.
- **char *host**: IP διεύθυνση του RPC server. Χρησιμοποιείται για την RPC επικοινωνία.

1.3.1 Συνάρτηση proc_call()

Η proc_call είναι η συνάρτηση που εκτελεί κάθε thread το οποίο καλείται να εξυπηρετήσει έναν client. Έχει ως μοναδικό όρισμα έναν void pointer ο οποίος μπορεί να δείξει/κρατήσει κάθε είδους δεδομένα. Για να χρησιμοποιήσουμε αυτά τα δεδομένα όμως, πρέπει να κάνουμε cast τον pointer σε έναν έγκυρο τύπο δεδομένων ή σε μια struct (βλ. γραμμή 195) η οποία μπορεί να περιέχει πάνω από έναν τύπους. Εκτός από αυτή την δομή, χρησιμοποιούνται αρκετές μεταβλητές όπου αναφέρονται παρακάτω.

- **char *Data**: Δυναμικός πίνακας στον οποίο θα αποθηκευτούν τα απεσταλμένα δεδομένα του client.
- **char *multiplier**: Δυναμικός πίνακας στον οποίο αποθηκεύεται ο πολλαπλασιαστής που χρησιμοποιείται στην συνάρτηση mul_prog_1.
- **int client_status**: Χρησιμοποιείται για έλεγχο αποσύνδεσης του client.
- **int id_pos**: Παίρνει τιμή από την μεταβλητή id και δείχνει στον πίνακα available_IDs που να αποθηκευτεί το επόμενο id.
- **int thrd_id**: Μοναδικό αναγνωριστικό κάθε thread.
- **int clnt_fd**: "Κρατάει" τον file descriptor του πελάτη που εξυπηρετεί. Χρησιμοποιείται για την επικοινωνία με αυτόν.
- **int data_arr_size**: Είναι το πλήθος των στοιχείων που θα περιέχει ο πίνακας Data.

Στην συνέχεια, αποδίδεται αναγνωριστικό ID στο thread είτε από τον πίνακα, είτε από τον αύξοντα αριθμό id. Καθώς το thread προχωράει σε μία ατέρμονη while loop χρησιμοποιεί την συνάρτηση recv() για να πάρει τα δεδομένα που του έστειλε ο client. Την πρώτη φορά που θα εκτελέσει την συνάρτηση αυτή (βλ. γραμμή 220) λαμβάνει το μέγεθος του πίνακα Data τον οποίο θα δημιουργήσει αμέσως μετά και την δεύτερη φορά λαμβάνει τα δεδομένα που θα αποθηκεύσει σε αυτόν. Ανάλογα με την τιμή της πρώτης θέσης του πίνακα, καλείται η ανάλογη συνάρτηση.

- **recv(...)**: Η κλήση συστήματος recv() λαμβάνει ένα μήνυμα μέσω ενός socket και επιστρέφει τον αριθμό των χαρακτήρων που λήφθηκαν ή "-1" σε περίπτωση λάθους.

1.3.2 Συναρτήσεις avg_prog_1() / min_max_prog_1() / mul_prog_1()

Οι τρεις αυτές συναρτήσεις έχουν ως σκοπό την αποστολή των δεδομένων στον RPC server, την παραλαβή των αποτελεσμάτων από αυτόν και την αποστολή τους πίσω στον client. Όπως έχουμε προαναφέρει, το κομμάτι της RPC επικοινωνίας είναι έτοιμο, οπότε εμείς προσθέσαμε δύο συναρτήσεις, την send() για να στέλνει τα αποτελέσματα στον client και την str2int() η οποία με την βοήθεια της strtok() και της atoi() μετατρέπει τα στοιχεία του πίνακα data από χαρακτήρες σε ακέραιους αριθμούς.

- **send(...)**: Η κλήση συστήματος send() στέλνει ένα μήνυμα μέσω ενός socket και επιστρέφει τον αριθμό των χαρακτήρων που στάλθηκαν ή "-1" σε περίπτωση λάθους.

- **strtok(...)**: Η strtok(char * str, const char * delim) διασπά τη συμβολοσειρά str σε μια σειρά από tokens χρησιμοποιώντας το delimiter delim. Επιστρέφει έναν pointer στο πρώτο token που βρίσκεται στη συμβολοσειρά ή έναν NULL pointer, εάν δεν υπάρχουν άλλα tokens για ανάκτηση.

- **atoi(...)**: Η atoi(const char * str) μετατρέπει τη συμβολοσειρά str σε ακέραιο αριθμό. Επιστρέφει τον μετατρεπόμενο ακέραιο αριθμό ως τιμή int ή μηδέν εάν αποτύχει η μετατροπή.

- **str2int(...)**: Δεσμεύει χώρο στη μνήμη για τον πίνακα (Y_val) της δομής (Y) στον οποίο θα αποθηκευτούν οι ακέραιες τιμές. Κάθε token που δημιουργεί η strtok() κρατείται από τον pointer num, μετατρέπεται σε ακέραιο από την atoi() και αποθηκεύεται στον πίνακα ακεραίων. Ο πίνακας αυτός αρχικοποιείται με 10 θέσεις μνήμης, αλλά μεγαλώνει κατά 10 όσες φορές χρειαστεί, σε περίπτωση που το πλήθος των αριθμών είναι μεγαλύτερο του μεγέθους του. Επιστρέφει το μέγεθος του πίνακα.

Αφού γίνει η μετατροπή των δεδομένων από αλφαριθμητικό σε ακεραίους, στέλνονται στον RPC server και ο RPC client περιμένει να αποθηκεύσει τα αποτελέσματα στην μεταβλητή result_1. Όταν τα λάβει, χρησιμοποιεί την send() για να τα στείλει στον socket client, ελευθερώνει τον χώρο που δέσμευσε για τον πίνακα των ακεραίων και επιστρέφει στην while της συνάρτησης proc_call() για να λάβει τα επόμενα δεδομένα.

2 C socket client

2.1 client.c

Ο socket client είναι, η διεργασία του τελικού χρήστη ο οποίος θα χρησιμοποιήσει αυτά που φτιάξαμε. Ακολουθούμε την ίδια διαδικασία με τον socket server για την σύνδεση, η μόνη διαφορά είναι ότι τώρα χρησιμοποιούμε την connect() αντί της accept(). Αφού ολοκληρωθεί το connection μεταξύ socket client / server, εμφανίζει, στον χρήστη, ένα menu επιλογών και περιμένει να επιλέξει. Η επιλογή αποθηκεύεται στον πίνακα answer και με βάση αυτήν συνεχίζει ή όχι το πρόγραμμα. Εάν συνεχίσει, καλείται η συνάρτηση functions() με παραμέτρους τον file descriptor του sever και την απάντηση του χρήστη. Η συνάρτηση αυτή είναι υπεύθυνη για την αποστολή και την παραλαβή των δεδομένων.

- **connect(...)**: Συνδέει τον client στο listening socket του TCP server και επιστρέφει "0" σε περίπτωση επιτυχίας και "-1" σε περίπτωση αποτυχίας.

2.1.1 Συνάρτηση functions()

Η συνάρτηση αυτή δεσμεύει μνήμη με την χρήση της malloc() για τον πίνακα (Data) και καλεί την συνάρτηση getData() για την οποία θα μιλήσουμε παρακάτω. Στη συνέχεια, στέλνει στον server τον αριθμό των bytes που πρέπει να διαβάσει στην επόμενη recv() έτσι ώστε να διαβάσει όλα τα δεδομένα, του πίνακα Data, που θα σταλθούν. Ανάλογα την επιλογή του χρήστη θα μπει σε μία από τις επόμενες if statements για να στείλει, αλλά και να λάβει, τα σωστά δεδομένα. Συγκεκριμένα στις δύο πρώτες if, στέλνει τα δεδομένα του πίνακα Data και περιμένει τα αποτελέσματα. Αν η επιλογή του χρήστη είναι η τρίτη όμως, τότε δεσμεύει μνήμη για έναν πίνακα μεγέθους ίσο με του Data - 2 στον οποίο θα αποθηκεύσει τα αποτελέσματα. Αυτό γίνεται γιατί ο πίνακας αυτός δεν θα περιέχει τα δύο επιπλέον στοιχεία που περιέχει ο Data. Συνεχίζοντας την εκτέλεση του το πρόγραμμα στέλνει τα δεδομένα και τον πολλαπλασιαστή του διανύσματος που έχει επιλέξει ο χρήστης, περιμένοντας τα αποτελέσματα. Σε κάθε περίπτωση καλείται η απαραίτητη συνάρτηση για την εμφάνιση των αποτελεσμάτων και η free() για την αποδέσμευση της μνήμης των δυναμικών πινάκων.

2.1.2 Συνάρτηση getData()

Η getData() είναι η συνάρτηση με την οποία ο χρήστης γεμίζει με δεδομένα τους πίνακες Data και multiplier. Τα δύο πρώτα σοιχεία του Data δεν τα ορίζει ο χρήστης, παίρνουν συγκεκριμένες τιμές κατά την εκτέλεση του προγράμματος. Το πρώτο στοιχείο είναι η απάντηση του χρήστη και το επόμενο είναι το "new line" (\n) έτσι ώστε να μην συμπεριληφθεί στο convertion της str2int().

Παράμετροι της getData():

- **int srv**: Server file descriptor, χρησιμοποιείται για την επικοινωνία με τον socket server.
- **char *data**: Pointer στον πίνακα Data για την διαχείριση του μέσα στην συνάρτηση.
- **int start_point**: Σηματοδοτεί από ποιο σημείο, και μετά, του πίνακα θα ξεκινήσουν να αποθηκεύονται τα δεδομένα. Χρησιμοποιείται και ως μετρητής μεγέθους του πίνακα.
- **char *ans**: Pointer στον πίνακα answer. Χρησιμοποιείται για να ξέρει το πρόγραμμα αν θα χρειαστεί να δώσει ο χρήστης τιμή στον πολλαπλασιαστή.
- **char multiplier[]**: Pointer στον πίνακα multiplier για την διαχείριση του μέσα στην συνάρτηση.

Αναλυτικά οι μεταβλητές που χρησιμοποιούνται στην getData():

- **int arr_size**: Χρησιμοποιείται για να ξέρουμε τι είναι αυτό που επιστρέφει η συνάρτηση getData(). Δηλαδή το μέγεθος του πίνακα Data.
- **char arr_elem**: Είναι το στοιχείο που δίνει ο χρήστης και πάνω σε αυτό γίνονται όλοι οι έλεγχοι για την ορθή εισαγωγή των στοιχείων στον πίνακα Data.
- **int minus**: Χρησιμοποιείται ως flag (0 ή 1) για να ξέρουμε αν ο χρήστης έχει δώσει αρνητικό αριθμό.

- **int numCount**: Μετράει το πλήθος των ψηφίων του αριθμού που έδωσε ο χρήστης.
- **int cmp**: Παίρνει την ακέραια τιμή του arr_elem. Χρησιμοποιείται για να δούμε αν ο χρήστης έδωσε αριθμό ή οτιδήποτε άλλο (εκτός από "\n").
- **int numbers[10]**: Πίνακας αριθμών (0-9), συγκρίνεται με την μεταβλητή cmp για την εξακρίβωση της εισόδου.
- **int valid_char**: Χρησιμοποιείται ως flag (0 ή 1) για να ξέρει το πρόγραμμα αν θα δεχτεί τον χαρακτήρα ή όχι.