



ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
UNIVERSITY OF WEST ATTICA

RPC

RPC (Server/Client)
Socket (Server/Client)
Threads

**Κατανεμημένα Συστήματα
(Εργαστήριο)**

Παπαντωνάκης Σταυρός CSE45227
(ΣΤ12)

15/05/2020

Remote Procedure Calls (RPC) (Unix/Sun)

Απομεμακρυσμένη Κλήση Διαδικασιών

Παπαντωνάκης Σταύρος
CSE45227

Αναφορά πρώτης εργαστηριακής άσκησης
Εργαστήριο κατανεμημένων
ΣΤ12



Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών
Πανεπιστήμιο Δυτικής Αττικής
Αθηνά
15/05/2020

Copyright ©2020 Παπαντώνακης Σταύρος
Το Παρόν Έργο παρέχεται υπό τους όρους της Άδειας:



Αναφορά Δημιουργού-Μη Εμπορική Χρήση-Παρόμοια Διανομή 4.0 Διεθνής

Το πλήρες κείμενο αυτής της άδειας είναι διαθέσιμο εδώ:
<http://creativecommons.org/licenses/by-nc-sa/4.0/>

Είστε ελεύθερος να:

Διαμοιραστείτε – να αντιγράψετε και αναδιανείμετε το υλικό με οποιοδήποτε μέσο και μορφή.

Προσαρμόσετε – να αναμείξετε, μετασχηματίσετε και να επεκτείνετε το υλικό.

Ο αδειοδότης δεν μπορεί να σας αφαιρέσει αυτές τις ελευθερίες όσο ακολουθείτε τους όρους της παρούσας άδειας.

Υπο τους ακόλουθους όρους:



Αναφορά Δημιουργού – Θα πρέπει να αναφέρετε τον δημιουργό του έργου, να παρέχετε σύνδεσμο προς αυτή την άδεια, και να υποδείξετε τυχόν αλλαγές. Μπορείτε να το κάνετε με οποιοδήποτε εύλογο μέσο, αλλά όχι με τρόπο που να υπονοεί ότι ο αδειοδότης επικροτεί εσάς ή τη χρήση του έργου από εσάς.



Μη Εμπορική Χρήση – Δεν μπορείτε να χρησιμοποιήσετε το υλικό για εμπορικούς σκοπούς.



Παρόμοια Διανομή – Αν αναμείξετε, μετασχηματίσετε ή επεκτείνετε το υλικό, θα πρέπει να διανείμετε τις αλλαγές σας υπό την ίδια άδεια με το πρωτότυπο έργο.

Όχι επιπλέον περιορισμοί – Δεν μπορείτε να εφαρμόσετε νομικούς όρους ή **τεχνικά μέσα** που να περιορίζουν νομικά τους άλλους να πράξουν σύμφωνα με τις ελευθερίες αυτής της άδειας.

Σημειώσεις:

Δεν χρειάζεται να ακολουθήσετε την άδεια για τμήματα του υλικού που θεωρούνται δημόσια γνώση (public domain) ή όπου η χρήση τους επιτρέπεται εξαιτίας μιας **εξαίρεσης ή περιορισμού**.

Δεν δίνονται εγγυήσεις. Η άδεια ίσως να μη σας δίνει όλα τα δικαιώματα για την επιδιωκόμενη χρήση. Για παράδειγμα, επιπλέον δικαιώματα όπως **δημοσιότητα, ιδιωτικότητα, ή ηθικά δικαιώματα** μπορεί να επιβάλλουν περιορισμούς στη χρήση του υλικού.

Το παρόν έργο στοιχειοθετήθηκε σε \LaTeX . Ο πηγαίος κώδικας του είναι διαθέσιμος στην παρακάτω τοποθεσία:

<https://github.com/lardianos/Distributed>

Περιεχόμενα

1	Απάντηση	5
1.1	Client.x	5
1.2	Calculator (RPC Server)	6
1.3	Calculator (RPC Client/Socket Server)	8
1.4	Socket Client	11

Γενικό Πλαίσιο

Σας ζητείται να φτιάξετε σε C έναν concurrent server (διεργασία εξυπηρετητή) ο οποίος ως έργο εξυπηρέτησης θα επιτελεί τους ακόλουθους υπολογισμούς (λαμβάνοντας ως εισόδους έναν πραγματικό αριθμό a και ένα διάνυσμα ακεραίων της μορφής $Y(y_1, y_2, \dots, y_n)$ μήκους n όπου το n θα το ορίζει ο χρήστης, και τις οποίες θα μπορούν να στέλνουν επαναληπτικά ένας ή περισσότεροι clients / διεργασίες πελατών):

1. Τη μέση τιμή του διανύσματος Y (επιστροφή: ένας πραγματικός αριθμός)
2. Τη μέγιστη και την ελάχιστη τιμή του Y (επιστροφή: ένας πίνακας μήκους 2 ακεραίων)
3. Το γινόμενο $a * Y$ (επιστροφή: ένα διάνυσμα πραγματικών αριθμών μήκους n)

Η επικοινωνία θα πρέπει να γίνεται μέσω TCP AF_INET (Internet Domain) sockets. Η κάθε διεργασία socket-Client θα διαβάζει από το πληκτρολόγιο (επαναληπτικά, μέχρι να δηλώσει ο χρήστης ότι δεν επιθυμεί να συνεχίσει).

- (α) την επιλογή του υπολογισμού που επιθυμεί ο χρήστης να γίνει (1,2,3)
- (β) τα αντίστοιχα-απαραίτητα κατά περίπτωση δεδομένα (n, Y, a) , θα τα διοχετεύει στη διεργασία socket-Server και θα περιμένει να λάβει από αυτήν το αποτέλεσμα για να το τυπώσει στην οθόνη.

Η διεργασία socket-Server θα δέχεται τα δεδομένα προς επεξεργασία από τις διεργασίες socket-Clients, και θα παράγει το αντίστοιχο αποτέλεσμα ΟΧΙ μέσω δικιάς του (τοπικής) συνάρτησης-υπολογισμού ΑΛΛΑ μέσω κατάλληλου Remote Procedure Call που θα υλοποιήσετε με τη βοήθεια του ONC RPC implementation. Θα πρέπει δηλαδή η διεργασία socket-Server (λειτουργώντας παράλληλα και ως RPC-Client) να καλεί (ανάλογα με την τιμή υπολογισμού που έστειλε ο χρήστης - 1,2,3) την αντίστοιχη ρουτίνα από έναν RPC-Server και να περιμένει το αντίστοιχο αποτέλεσμα από αυτόν (προκειμένου να το διοχετεύσει στη συνέχεια στον αντίστοιχο socket-Client).

Όσον αφορά το RPC-based μέρος της επικοινωνίας, θα πρέπει πρώτα να ορίσετε σωστά το απαιτούμενο ('.x') interface file (ορίζοντας μέσα σε αυτό τρεις ξεχωριστές συναρτήσεις-διαδικασίες (μία για κάθε έναν από τους τρεις υπολογισμούς που ζητούνται παραπάνω), στη συνέχεια

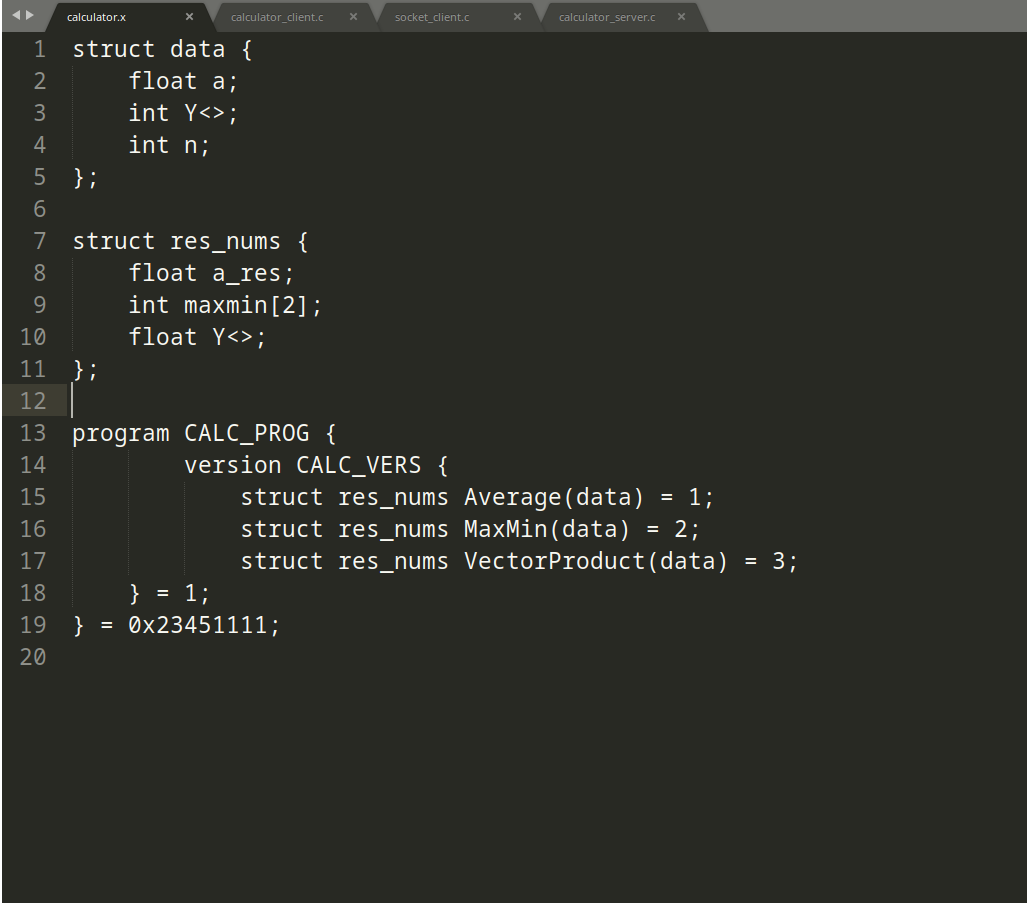
να παράγετε αυτοματοποιημένα μέσω του rpcgen utility (και με βάση τα όσα διδαχθήκατε στο εργαστήριο) τόσο τα απαιτούμενα system modules (RPC-server-stub module και RPC-client-stub module) για την υλοποίηση των ζητούμενων RPCs, όσο και τα έτοιμα templates για τα δύο application modules της εφαρμογής σας (RPC-server-application module και RPC-client-application module), και ακολούθως:

- (α) να ολοκληρώσετε κατάλληλα το RPC-server-application module (το οποίο θα επιτελεί τις βασικές εργασίες εξυπηρέτησης πάνω στα δεδομένα-παραμέτρους που θα στέλνει απομεμακρυσμένη ο RPC-Client/socket-Server).
- (β) να ολοκληρώσετε κατάλληλα επίσης το RPC-client-application module (μέσω του οποίου θα επιτελείται επί της ουσίας η κλήση της εκάστοτε απομεμακρυσμένης διαδικασίας) ενσωματώνοντας/συγχωνεύοντας το μεταξύ άλλων με τη βασική διεργασία socket-Server που περιγράφηκε παραπάνω.

1 Απάντηση

1.1 Client.x

Αρχικά δημιουργούμε στο αρχείο .x μια struct (data) για τα δεδομένα που θα στέλνουμε στην RPC Procedure και μια struct (res_num) για τα δεδομένα που θα λαμβάνουμε πίσω. Στη struct data δηλώνουμε μια μεταβλητή τύπου float a στη οποία θα τοποθετούμε την τιμή του a, ένα πίνακα int Y<> ο οποίος θα αρχικοποιείται και θα δεσμεύει μνήμη ανάλογα με την τιμή n που θα δίνει ο χρήστης, και μια τιμή int n στην οποία θα αποθηκεύεται το μέγεθος του πίνακα Y. Για την struct res_num θα χρειαστούμε μια float a_res στην οποία θα αποθηκεύεται το αποτέλεσμα του μέσου ορού. Έναν int maxmin[2] στον οποίο θα αποθηκεύεται το max και το min, και έναν float Y<> στον οποίο θα αποθηκεύεται το διάνυσμα αφότου έχει πολλαπλασιαστεί με το a. Τέλος θα χρειαστούμε 3 procedures ένα για κάθε λειτουργία, τα οποία επιστρέφουν ένα struct res_num για με τα αντίστοιχα αποτελέσματα και ως είσοδο μια struct data με τα δεδομένα προς υπολογισμό.



```
1 struct data {
2     float a;
3     int Y<>;
4     int n;
5 };
6
7 struct res_nums {
8     float a_res;
9     int maxmin[2];
10    float Y<>;
11 };
12
13 program CALC_PROG {
14     version CALC_VERS {
15         struct res_nums Average(data) = 1;
16         struct res_nums MaxMin(data) = 2;
17         struct res_nums VectorProduct(data) = 3;
18     } = 1;
19 } = 0x23451111;
```

Αμέσως μετά τρέχουμε την `rpcgen` και μας παράγει όλα τα απαραίτητα αρχεία.

1.2 Calculator (RPC Server)

Ανοίγουμε το αρχείο του RPC server και γράφουμε τον κώδικα που απαιτείτε για την υλοποίηση της `average`, `maxmin` και `vectorproduct`. Αρχικά για την `average` δηλώνουμε μεταβλητές και θέτουμε τιμές σε αυτές. τρέχουμε ένα `for loop` και αθροίζουμε της τιμές του `Y` και τέλος υπολογίζουμε των μεσώ όρο, τον αναθέτουμε στον `result.a_res` και επιστρέφουμε το `result`.


```
9 struct res_nums * average_1_svc(data *argp, struct svc_req *rqstp){
10     static struct res_nums result;
11     float average=0;
12     int n = argp->Y.Y_len;
13     for (int i = 0; i < n; i++) {
14         average += argp->Y.Y_val[i];
15     }
16     average /= n;
17     printf("%f\n",average);
18     printf("Server func 1\n");
19     result.a_res=average;
20     return &result;
21 }
```

Για την maxmin αναθέτουμε τιμές σε μεταβλητές, τρέχουμε ένα for loop και υπολογίζουμε το max και το min του πίνακα Y, το max το αναθέτουμε στο result.maxmin[0] και το min στο result.maxmin[1], τελειώνοντας επιστρέφουμε το result.

```
22 struct res_nums * maxmin_1_svc(data *argp, struct svc_req *rqstp){
23     static struct res_nums result;
24     int n = argp->Y.Y_len;
25     int max=argp->Y.Y_val[0],min=argp->Y.Y_val[0];
26     for (int i = 0; i < n; i++){
27         if (argp->Y.Y_val[i] > max) {
28             max = argp->Y.Y_val[i];
29         }else if (argp->Y.Y_val[i] < min){
30             min = argp->Y.Y_val[i];
31         }
32     }
33     printf("Server func 2\n");
34     printf("max %d min %d\n",max,min);
35     result.maxmin[0]=max;
36     result.maxmin[1]=min;
37     return &result;
38 }
```

Τέλος για την vectorproduct ακολουθούμε την ίδια διαδικασία δηλώνοντας μεταβλητές και αναθέτοντας τιμές. Σε αυτήν τη περίπτωση για την αποθήκευση των αποτελεσμάτων χρησιμοποιήσουμε τον πίνακα Y της struct για αυτόν τον λόγο δεσμεύουμε η θέση στην μνήμη, υπολογίζουμε της τιμές και επιστρέφουμε το result.

```

39 struct res_nums * vectorproduct_1_svc(data *argp, struct svc_req *rqstp){
40     static struct res_nums result;
41     int n = argp->Y._len;
42     float a = argp->a;
43     result.Y._len = n;
44     result.Y._val = (float *) malloc(n*sizeof(float));
45     for (int i = 0; i < n; i++){
46         result.Y._val[i] = (float) a * (float) argp->Y._val[i];
47         printf("%f\n",result.Y._val[i] );
48     }
49     printf("Server func 3\n");
50     return &result;
51 }
52

```

1.3 Calculator (RPC Client/Socket Server)

Σε αυτό και στο επόμενο section δεν έχω χρησιμοποίηση screenshots λόγω της μεγάλης έκτασης του κώδικα, θα προσπαθήσω να το εξηγήσω πως λειτουργεί μόνο με λογία.

Αρχικά ο έτυμος κώδικας που παράχθηκε απο το rpcgen για των RPC Client, όταν καλείτε η calc_prog_1 καλούνται και η τρεις RPC procedures. επομένως προσθέτουμε μια λογική τριών if, όπου η κάθε μια ελέγχει μια μεταβλητή select για το αν είναι 1,2 η 3 το όπιο select το προσθέτουμε στα inputs της calc_prog_1 για να το λαμβάνουμε όταν την καλούμε. Σε κάθε μια από της if κάνουμε της απαραίτητες δεσμεύσεις χορού στην μνήμη και ανάθεσης τιμών στις μεταβλητές της struct data.

Στην συνέχεια δημιουργούμε 3 struct που θα μας βοηθήσουν να αποθηκεύουμε δεδομένα και να τα περνάμε ανάμεσα στις συναρτήσεις μας.

```

struct data_save{
    float a;
    int * Y;
    int n;
};

struct addres_data{
    struct sockaddr_in addr;
    int len;
    int *newsockfd;
    char * s_host;
};

```

```
typedef struct {
    float a_res;
    int maxmin[2];
    float *Y;
    int Y_len;
} res_nums_cl;
```

Παρατηρούμε ότι ο έτυμος κώδικας που έχει παραχθεί, ακολουθεί ένα μοτίβο. Αν στο αρχείο .x είχαμε δηλώσει περισσότερα programs με procedures θα μας είχε εμφάνιση περισσότερες functions της μορφής calc_prog_1, calc_prog_2, calc_prog_3 κλπ. Παρόμοιο μοτίβο βλέπουμε και στη struct μεταβλητές εισόδου και επιστροφής που δηλώνει πχ result_1, result_2, result_3, average_1_arg, maxmin_1_arg, vectorproduct_1_arg. Κοιτάζοντας στο αρχείο .h παρατηρούμε ότι οι πινάκες της μορφής Y<> δηλώνονται ως μια struct που περιεχί έναν pointer αντιστοίχου τύπου καθώς και μια μεταβλητή για την αποθηκεύσει του μήκους.

Για την αρχικοποίηση των δεδομένων μέσα στην calc_prog_1 προσθέτουμε στην είσοδο της μερικά πράγματα ακόμα όπως την float n_num, την int n_num και την int * y_table. της οποίες θα της χρησιμοποιήσουμε για να περάσουμε τα δεδομένα μας σε αυτήν. Επίσης αλλάζουμε τον τύπο επιστροφής από void σε res_nums_cl (σχόλιο: εδώ θα απορείτε βεβαία γιατί φτιάξαμε της struct data_save και res_num_cl ενώ θα μπορούσαμε να χρησιμοποιήσουμε της ήδη υπάρχουσες του αρχείου .h καθώς είναι όμοιες, η απάντηση είναι ότι κάλλιστα θα μπορούσαμε να το κάνουμε άλλα υπάρχει ένα βασικό πρόβλημα, δεν το σκεφτήκαμε όταν γράφαμε των κώδικα...).

Τέλος αφότου καλέσουμε την κάθε μια από αυτές της συναρτήσεις και μας επιστρέψουν τα αποτελέσματα, τα αναθέτουμε σε μια μεταβλητή struct res_num_cl ret1 και τα επιστρέφουμε μέσω του return.

Στην συνέχεια φτιάχνουμε μια νέα συνάρτηση void * reciveMessage(void * cli_data) όπου θα υλοποιεί την βασική λειτουργία του προγράμματος όπου την εκτελούν threads μέσω της pthread_create() και παίρνει ως είσοδο έναν void pointer, εμείς στον void pointer αναθέτουμε έναν struct pointer της δομής addres_data όπου περιεχί κάποια βασικά δεδομένα για την επικοινωνία με των σωστό client καθώς και το socket, οπός την διεύθυνση του client τον file descriptore του socket και την διεύθυνση του RPC server. αρχικά δηλώνουμε κάποιες μεταβλητές, flags κλπ. Δεν έχει νόημα να της αναλύσουμε όλες μια μια, η περισσότερες εξάλλου

φαίνεται τη κάνουν από το όνομα τους. Αυτές που έχει νόημα να αναφέρουμε είναι η struct data_save data; στην όπια αποθηκεύουμε τα δεδομένα αφού τα λάβουμε από των socket client, την struct addres_data * local_address_data = cli_data; οπού αποθηκεύουμε τα δεδομένα επικοινωνίας του client. των char buffer[BUF_SIZE]; που θα χρησιμοποιήσουμε για να αποθηκεύσουμε τα δεδομένα που μας στέλνει ο χρήστης και την res_nums_cl ret1; οπού θα πάρουμε τα δεδομένα άπω τα return της calc_prog_1.

ο Βασικός κώδικας αυτής της συνάρτησης τρέχει σε έναν ατέρμον βρόχο οπού τερματίζει μόνο όταν πάρει από τον socket client ως επιλογή λειτουργίας των χαρακτήρα '0'. Στην ουσία αυτό που κάνει είναι να εκτελεί τη recvfrom() οπού περιμένει μέχρις ότου ο χρήστης του στήλη δεδομένα. μόλις τα λάβει ελέγχει αν ο πρώτος χαρακτήρας είναι ο χαρακτήρας '0', εάν είναι εκτελεί return(void *)"END"; και τερματίζει την συνάρτηση. Σε διαφορετική περίπτωση έχουμε φτιάξει έναν κώδικα από μερικά for loops τα οποία κάνουν extract τα δεδομένα από τον buffer, τα μετατρέπουν από string σε διαχειρίσιμα δεδομένα(int, float κλπ.) έτσι ώστε να μπορούμε να τα αποστείλουμε στην calc_prog_1. αμέσως μετά έχουμε 3 if οπού επιλεγούν ανάλογα με το select τη δεδομένα θα στείλουν στην calc_prog_1. Αφότου εκτελεστή η calc_prog_1 επιστρέφει την ανάλογη struct με τα αποτελέσματα στο ret1. Τέλος μετατρέπουμε τα αποτελέσματα ξανά σε string με την sprintf() και τα τοποθετούμε σε έναν buffer με την strcat() και τα αποστέλλουμε στον socket client με την sendto(). Τέλος ξαναγυρίζουμε στην αρχή και περιμένουμε να μας ξαναστείλει ο χρήστης δεδομένα.

Στην main ο έτυμος κώδικας χρησιμοποιεί argv είσοδο για την διεύθυνση του RPC Server. Για να φτιάξουμε την επικοινωνία με τα Socket δηλώνουμε 2 ακεραίους οπού θα αποθηκευτούν τα file descriptors που θα δημιουργηθούν. Εκτελούμε την sockfd=socket(AF_INET,SOCK_STREAM,0); και δημιουργούμε ένα socket, άμεσα μετά τρέχουμε την setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR,&yes, sizeof(int)) για να σετάρουμε το socket. Αναθέτουμε τιμές στο struct του socket.

```
memset(&addr,0 , sizeof (addr));
addr.sin_family =AF_INET;
addr.sin_addr.s_addr=INADDR_ANY;
addr.sin_port=PORT;
```

Στην συνεχεια κανουμε bind() και listen() και περιμενουμε για connect με την newsockfd=accept(sockfd, (struct sockaddr*) &cl_addr, (socklen_t*)

&len); μόλις γίνει το accept αποθηκεύουμε την διεύθυνση του του client και το fd του socket του στην δομή struct addres_data cl_data και δημιουργούμε ένα thread με την pthread_create το οποίο καλή την reciveMessage και της περνάει την δομή cl_data και ξαναπαίρνουμε για connect από άλλο client.

1.4 Socket Client

Στον client έχουμε δημιουργήσει μια συνάρτηση keyboard_read() που την χρησιμοποιούμε για να διαβάσουμε χαρακτήρες από το πληκτρολόγιο, μια συνάρτηση num_control για να ελέγχουμε αν οι χαρακτήρες που έδωσε ο χρήστης είναι αποδεκτοί, μια συνάρτηση receiveMessage που αποτυπώνει ότι στέλνει RPC Clinet/Socket Server. Και τέλος μια συνάρτηση μενού η οποία παρουσιάζει το μενού με της επιλογές του χρήστη, διαβάζει τα δεδομένα από το πληκτρολόγιο και τα αποθηκεύει στην δομή data_save και το κάνουμε return.

```
typedef struct {
    char select;
    char * a_value;
    char * Y_value;
    char * n_value;
    int n_length;
    int Y_length;
    int a_length;
} data_save;
```

Στην main τώρα κάνουμε την αντιστοιχεί διαδικασία με των Socket Server, απλά αντί για accept() κάνουμε connect() και ανάλογα τι επιλογή διαμορφώνουμε τα δεδομένα σε έναν πίνακα χαρακτήρων τις μορφής char buf[]={select;n;Y.value[0],Y.value[1],Y.value[2]....Y.value[n];a} των οποίων προωθούμε στην sendto() και καλούμε την recive περιμένοντας τα αποτελέσματα.