Πανεπιστήμιο Δυτικής Αττικής
Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Υπολογιστών

Εργαστήριο Κατανεμημένων Συστημάτων - Εργασία 1

Χρήστος Μαργιώλης – 19390133

Μάιος 2022

Περιεχόμενα

1 Χρήση rpcgen

Πριν γίνει η ανάπτυξη του client/server κώδικα, πρέπει να παραχθούν τα απαραίτητα RPC αρχεία. Στο αρχείο `rpc.x` δηλώνονται τα ονόματα και οι δομές των Remote Procedure Calls.

Παράγουμε τα client και server stubs (αρχεία στα οποία θα συμπληρωθεί ο κώδικας), καθώς και το `Makefile` για να είναι πιο εύκολη η μεταγλώττιση:

```
$ rpcgen -Ss -C > rpc_server.c
$ rpcgen -Sc -C > rpc_client.c
$ rpcgen -Sm > Makefile.rpc
```

Στο `Makefile.rpc` κάνουμε μερικές αλλαγές ωστέ ο server να μην τρέχει στο background, και στην μεταγλώττιση να περιέχονται τα `rpc_client.c` και `rpc_server.c`:

```
...
TARGETS_SVC.c = rpc_server.c rpc_svc.c rpc_xdr.c
TARGETS_CLNT.c = rpc_client.c rpc_clnt.c rpc_xdr.c
...
CFLAGS += -g -DRPC_SVC_FG
RPCGENFLAGS = -C
```

2 Εκτέλεση κώδικα

Το αρχείο `sock_client.c` υλοποιεί τον client τον οποίο εκτελεί ο χρήστης. Το `rpc_client.c`, παρόλο που λέει "client", είναι ο server με τον οποίο επικοινωνεί ο `sock_client.c` και κάνει τις remote κλήσεις στον `rpc_server.c`, ο οποίος υλοποιεί τις συναρτήσεις των RPC.

Κάνουμε compile τους κώδικες:

```
$ make
make -f Makefile.rpc
rpcgen -C rpc.x
cc  -O2 -pipe -g -DRPC_SVC_FG -c rpc_client.c -o rpc_client.o
cc  -O2 -pipe -g -DRPC_SVC_FG -c rpc_clnt.c -o rpc_clnt.o
cc  -O2 -pipe -g -DRPC_SVC_FG -c rpc_xdr.c -o rpc_xdr.o
cc -o rpc_client  rpc_client.o rpc_clnt.o rpc_xdr.o  -O2 -pipe -g -DRPC_SVC_FG
cc  -O2 -pipe -g -DRPC_SVC_FG -c rpc_server.c -o rpc_server.o
cc  -O2 -pipe -g -DRPC_SVC_FG -c rpc_svc.c -o rpc_svc.o
cc -o rpc_server  rpc_server.o rpc_svc.o rpc_xdr.o  -O2 -pipe -g -DRPC_SVC_FG
cc sock_client.c -o sock_client
```

Τα τρία προγράμματα εκτελούνται ως εξής:

- `./rpc_server`
- `./rpc_client [-b backlog] [-p port] hostname`
- `./sock_client [-p port] hostname`

Πριν τρέξουμε οποιοδήποτε RPC πρόγραμμα, πρέπει να αρχίσουμε το `rpcbind(8)` service (για FreeBSD):

```
# service rpcbind start
```

Αφού τρέξουμε τον server, μπορούμε να δούμε ότι τα Remote Procedure Calls έχουνε εισαχθεί στον πίνακα RPC. Η εντολή `rpcinfo(8)` τυπώνει τον πίνακα:

```
$ ./rpc_server
$ rpcinfo
   program version netid     address                service    owner
    100000    4    tcp       0.0.0.0.0.111          rpcbind    superuser
    100000    3    tcp       0.0.0.0.0.111          rpcbind    superuser
    100000    2    tcp       0.0.0.0.0.111          rpcbind    superuser
    100000    4    udp       0.0.0.0.0.111          rpcbind    superuser
    100000    3    udp       0.0.0.0.0.111          rpcbind    superuser
    100000    2    udp       0.0.0.0.0.111          rpcbind    superuser
    100000    4    tcp6      ::.0.111               rpcbind    superuser
    100000    3    tcp6      ::.0.111               rpcbind    superuser
    100000    4    udp6      ::.0.111               rpcbind    superuser
    100000    3    udp6      ::.0.111               rpcbind    superuser
    100000    4    local     /var/run/rpcbind.sock  rpcbind    superuser
    100000    3    local     /var/run/rpcbind.sock  rpcbind    superuser
    100000    2    local     /var/run/rpcbind.sock  rpcbind    superuser
 536870912    1    udp6      ::.161.100             -          1001
 536870912    1    tcp6      ::.93.202              -          1001
 536870912    1    udp       0.0.0.0.44.39          -          1001
 536870912    1    tcp       0.0.0.0.219.61         -          1001
 536870913    1    udp6      ::.161.100             -          1001
 536870913    1    tcp6      ::.93.202              -          1001
 536870913    1    udp       0.0.0.0.44.39          -          1001
```

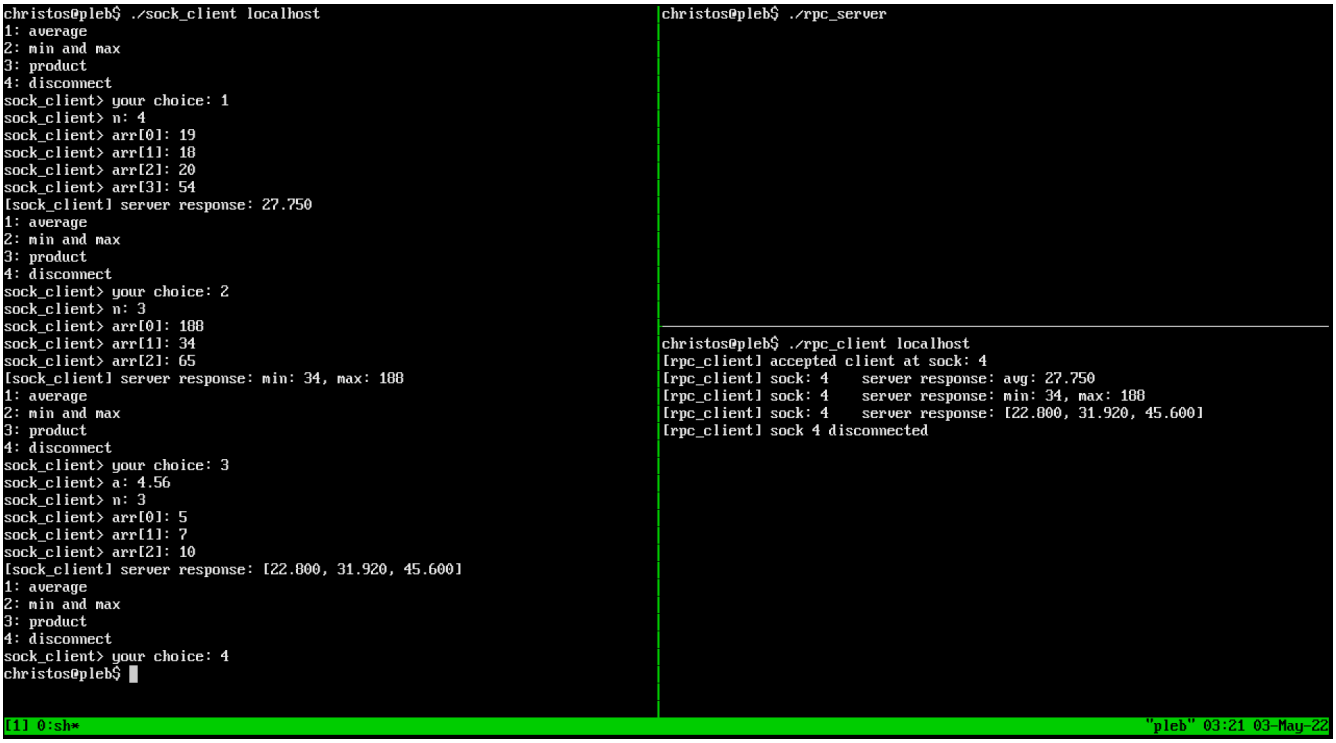
```
536870913    1    tcp        0.0.0.0.219.61        -          1001
536870914    1    udp6       ::.161.100            -          1001
536870914    1    tcp6       ::.93.202             -          1001
536870914    1    udp        0.0.0.0.44.39         -          1001
536870914    1    tcp        0.0.0.0.219.61        -          1001
```

Οι γραμμές που ξεκινούν με 53687* είναι τα RPC που δημιούργησε ο `rpc_server`.

3 Ενδεικτικά τρεξίματα

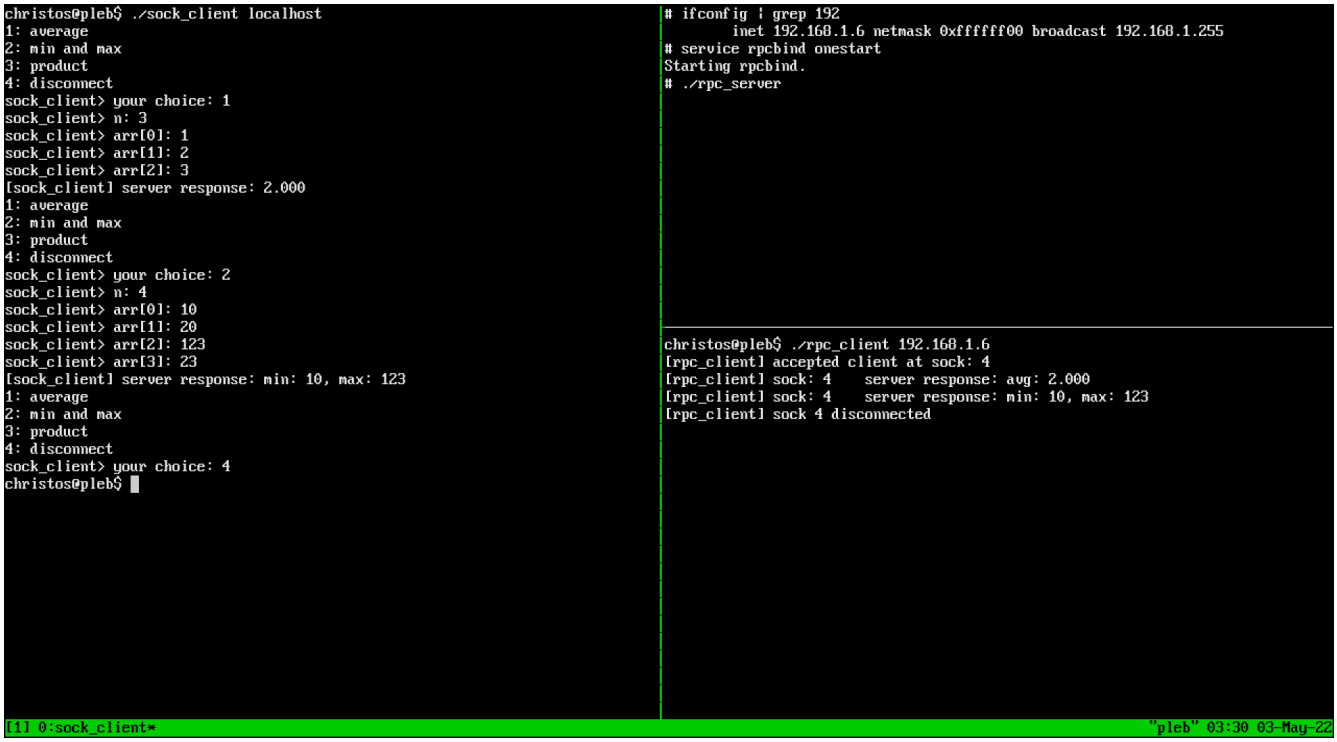
Σε κάθε ενότητα έχω παραθέσει τα output των 3 προγραμμάτων στην τελική τους κατάσταση. 'Ισως χρειαστεί λίγη μεγέθυνση...

3.1 Server και client στον ίδιο υπολογιστή



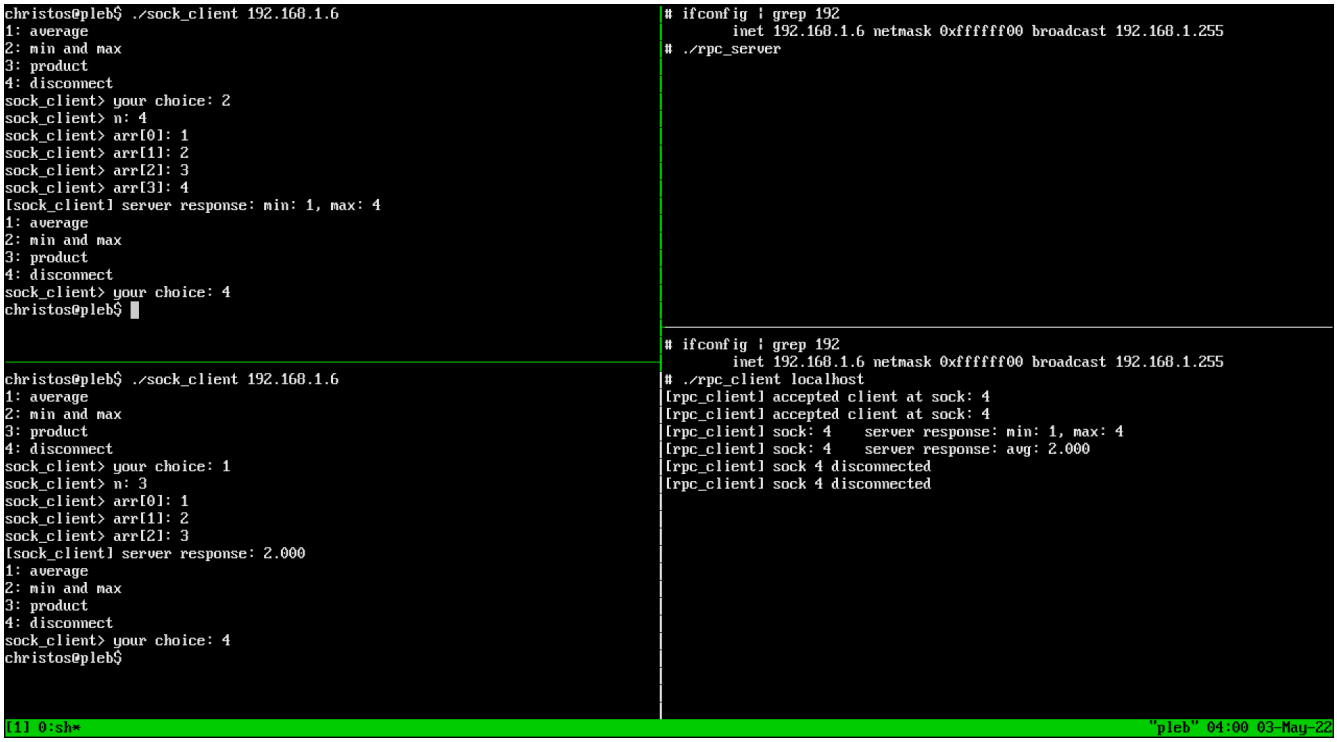
3.2 Server και client σε διαφορετικές IP

Ο RPC server τώρα τρέχει σε FreeBSD Jail, το οποίο, αν και βρίσκεται στο ίδιο τοπικό δίκτυο, έχει διαφορετική IP διεύθυνση. Ο RPC client και ο socket client τρέχουν ακόμα στο τρέχουν ακόμα στον localhost.



3.3 Δοκιμή concurrency

Ο RPC server και client τρέχουν και οι δύο σε FreeBSD Jail, και αυτή τη φορά θα δοκιμάσουμε να ελέγξουμε αν ο server είναι όντως concurrent.



4 Κώδικας

Ο κώδικας είναι σχολιασμένος στα σημεία που θεωρώ ότι μπορεί να υπάρξει σύχγηση, και όχι ακόμα και σε σημεία που είναι λίγο-πολύ ξεκάθαρο το τι συμβαίνει.

4.1 rpc.x

```
struct arg_arr {
        int     n;
        int     arr<>;
};

struct arg_prod {
        float   a;
        struct arg_arr arr;
};

struct minmax {
        int     arr<2>;
};
```

```
struct float_arr {
        float   arr<>;
};

program calc_avg_PROG {
        version calc_avg_VERS {
                float calc_avg(arg_arr) = 1;
        } = 1;
} = 0x20000000;

program calc_minmax_PROG {
        version calc_minmax_VERS {
                struct minmax calc_minmax(arg_arr) = 1;
        } = 1;
} = 0x20000001;

program calc_prod_PROG {
        version calc_prod_VERS {
                struct float_arr calc_prod(arg_prod) = 1;
        } = 1;
} = 0x20000002;
```

4.2 rpc_server.c

```
#include <err.h>
#include <stdlib.h>
#include <stdio.h>

#include "rpc.h"

static void *emalloc(size_t);

static void *
emalloc(size_t nb)
{
        void *p;

        if ((p = malloc(nb)) == NULL)
                err(1, "malloc");
        return (p);
}

float *
calc_avg_1_svc(arg_arr *argp, struct svc_req *rqstp)
{
        static float  result;
        int i, sum;

        for (i = 0, sum = 0; i < argp->n; i++)
                sum += argp->arr.arr_val[i];
        result = sum / (float)argp->n;

        return (&result);
}

struct minmax *
calc_minmax_1_svc(arg_arr *argp, struct svc_req *rqstp)
{
        static struct minmax  result;
        int i, *min, *max;

        result.arr.arr_len = 2;
        result.arr.arr_val = emalloc(2 * sizeof(int));
        min = &result.arr.arr_val[0];
        max = &result.arr.arr_val[1];
        *min = *argp->arr.arr_val;
        *max = *argp->arr.arr_val;
        for (i = 0; i < argp->n; i++) {
```

```
                if (argp->arr.arr_val[i] < *min)
                        *min = argp->arr.arr_val[i];
                if (argp->arr.arr_val[i] > *max)
                        *max = argp->arr.arr_val[i];
        }

        return (&result);
}

struct float_arr *
calc_prod_1_svc(arg_prod *argp, struct svc_req *rqstp)
{
        static struct float_arr  result;
        int i;

        result.arr.arr_len = argp->arr.n;
        result.arr.arr_val = emalloc(argp->arr.n * sizeof(float));
        for (i = 0; i < argp->arr.n; i++)
                result.arr.arr_val[i] = argp->a * argp->arr.arr.arr_val[i];

        return (&result);
}
```

4.3 rpc_client.c

```
#include <sys/socket.h>
#include <sys/types.h>

#include <arpa/inet.h>
#include <netdb.h>
#include <netinet/in.h>

#include <err.h>
#include <libgen.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include "rpc.h"

static void     serve_client(char *, int);
static void     sighandler(int);
static void     *emalloc(size_t);
static void     usage(void);

/* program name */
static char     *argv0;
/* becomes true when a termination signal is caught */
static volatile sig_atomic_t f_quit = 0;

void
calc_avg_prog_1(char *host, int sock)
{
        CLIENT *clnt;
        float  *result_1;
        arg_arr  calc_avg_1_arg;
        int *np, *arrp;

#ifndef DEBUG
        clnt = clnt_create(host, calc_avg_PROG, calc_avg_VERS, "netpath");
        if (clnt == (CLIENT *) NULL) {
                clnt_pcreateerror(host);
                exit(1);
        }
#endif  /* DEBUG */

        /* avoid typing all this */
```

```
        np = &calc_avg_1_arg.n;

        /* receive number of elements */
        if (recv(sock, np, sizeof(int), 0) < 0)
                goto fail;
        calc_avg_1_arg.arr.arr_len = *np;
        calc_avg_1_arg.arr.arr_val = emalloc(*np * sizeof(int));
        arrp = calc_avg_1_arg.arr.arr_val;

        /* receive array */
        if (recv(sock, arrp, *np * sizeof(int), 0) < 0)
                goto fail;

        /* make RPC */
        result_1 = calc_avg_1(&calc_avg_1_arg, clnt);
        if (result_1 == (float *) NULL) {
                clnt_perror(clnt, "call failed");
        }

        /* send results back to sock_client */
        printf("[%s] sock: %d\tserver response: avg: %.3f\n", argv0, sock, *result_1);
        if (send(sock, result_1, sizeof(float), 0) < 0)
                goto fail;
        return;
fail:
        /* we failed... */
        fprintf(stderr, "[%s] connection with client %d dropped\n", argv0, sock);
        close(sock);
        if (arrp != NULL)
                free(arrp);
#ifndef DEBUG
        clnt_destroy(clnt);
#endif          /* DEBUG */
}


void
calc_minmax_prog_1(char *host, int sock)
{
        CLIENT *clnt;
        struct minmax  *result_1;
        arg_arr  calc_minmax_1_arg;
        int i, *np, *arrp;

#ifndef DEBUG
        clnt = clnt_create(host, calc_minmax_PROG, calc_minmax_VERS, "netpath");
```

```
        if (clnt == (CLIENT *) NULL) {
                clnt_pcreateerror(host);
                exit(1);
        }
#endif  /* DEBUG */

        np = &calc_minmax_1_arg.n;

        if (recv(sock, np, sizeof(int), 0) < 0)
                goto fail;
        calc_minmax_1_arg.arr.arr_len = *np;
        calc_minmax_1_arg.arr.arr_val = emalloc(*np * sizeof(int));
        arrp = calc_minmax_1_arg.arr.arr_val;

        if (recv(sock, arrp, *np * sizeof(int), 0) < 0)
                goto fail;

        result_1 = calc_minmax_1(&calc_minmax_1_arg, clnt);
        if (result_1 == (struct minmax *) NULL) {
                clnt_perror(clnt, "call failed");
        }

        printf("[%s] sock: %d\tserver response: min: %d, max: %d\n",
            argv0, sock, result_1->arr.arr_val[0], result_1->arr.arr_val[1]);
        if (send(sock, result_1->arr.arr_val, 2 * sizeof(int), 0) < 0)
                goto fail;
        return;
fail:
        fprintf(stderr, "[%s] connection with client %d dropped\n", argv0, sock);
        close(sock);
        if (arrp != NULL)
                free(arrp);
#ifndef DEBUG
        clnt_destroy(clnt);
#endif          /* DEBUG */
}


void
calc_prod_prog_1(char *host, int sock)
{
        CLIENT *clnt;
        struct float_arr  *result_1;
        arg_prod  calc_prod_1_arg;
        int i, *np, *arrp;
        float *ap;
```

```
#ifndef DEBUG
        clnt = clnt_create(host, calc_prod_PROG, calc_prod_VERS, "netpath");
        if (clnt == (CLIENT *) NULL) {
                clnt_pcreateerror(host);
                exit(1);
        }
#endif  /* DEBUG */

        ap = &calc_prod_1_arg.a;
        np = &calc_prod_1_arg.arr.n;

        if (recv(sock, ap, sizeof(float), 0) < 0)
                goto fail;
        if (recv(sock, np, sizeof(int), 0) < 0)
                goto fail;
        calc_prod_1_arg.arr.arr.arr_len = *np;
        calc_prod_1_arg.arr.arr.arr_val = emalloc(*np * sizeof(int));
        arrp = calc_prod_1_arg.arr.arr.arr_val;

        if (recv(sock, arrp, *np * sizeof(int), 0) < 0)
                goto fail;

        result_1 = calc_prod_1(&calc_prod_1_arg, clnt);
        if (result_1 == (struct float_arr *) NULL) {
                clnt_perror(clnt, "call failed");
        }

        printf("[%s] sock: %d\tserver response: [", argv0, sock);
        for (i = 0; i < calc_prod_1_arg.arr.n; i++) {
                printf("%.3f%s", result_1->arr.arr_val[i],
                    i == calc_prod_1_arg.arr.n - 1 ? "" : ", ");
        }
        printf("]\n");

        if (send(sock, result_1->arr.arr_val, *np * sizeof(float), 0) < 0)
                goto fail;
        return;
fail:
        fprintf(stderr, "[%s] connection with client %d dropped\n", argv0, sock);
        close(sock);
        if (arrp != NULL)
                free(arrp);
#ifndef DEBUG
        clnt_destroy(clnt);
#endif          /* DEBUG */
```

```
}

static void
serve_client(char *host, int cfd)
{
        int n;

        for (;;) {
                /* receive option */
                if (recv(cfd, &n, sizeof(int), 0) < 0) {
                        /* something went wrong, we cant continue */
                        fprintf(stderr, "[%s] connection with %d dropped\n",
                            argv0, cfd);
                        close(cfd);
                        _exit(0);
                }
                switch (n) {
                case 1:
                        calc_avg_prog_1(host, cfd);
                        break;
                case 2:
                        calc_minmax_prog_1(host, cfd);
                        break;
                case 3:
                        calc_prod_prog_1(host, cfd);
                        break;
                case 4:
                        printf("[%s] sock %d disconnected\n", argv0, cfd);
                        close(cfd);
                        return;
                }
        }
}

/*
 * Gets called in case of a SIGINT or SIGTERM.
 */
static void
sighandler(int sig)
{
        f_quit = 1;
}

/*
 * Error checking malloc(3).
 */
```

```
static void *
emalloc(size_t nb)
{
        void *p;

        if ((p = malloc(nb)) == NULL)
                err(1, "malloc");
        return (p);
}

static void
usage(void)
{
        fprintf(stderr, "usage: %s [-b backlog] [-p port] hostname\n", argv0);
        exit(0);
}

int
main(int argc, char *argv[])
{
        struct sockaddr_in sin;
        struct hostent *hp;
        struct sigaction sa;
        int backlog = 5;
        int port = 9999;
        int sfd, cfd;
        char *host, ch;

        argv0 = basename(*argv);
        while ((ch = getopt(argc, argv, "b:p:")) != -1) {
                switch (ch) {
                case 'b':
                        if ((backlog = atoi(optarg)) < 1)
                                errx(1, "backlog value must be > 1");
                        break;
                case 'p':
                        if ((port = atoi(optarg)) < 1024)
                                errx(1, "can't use port number < 1024");
                        break;
                case '?':
                default:
                        usage();
                }
        }
        argc -= optind;
        argv += optind;
```

```
if (argc < 1)
        usage();
host = *argv;

memset(&sa, 0, sizeof(sa));
sigfillset(&sa.sa_mask);
sa.sa_handler = sighandler;

/* be sensitive to termination signals */
if (sigaction(SIGINT, &sa, NULL) < 0)
        err(1, "sigaction(SIGINT)");
if (sigaction(SIGTERM, &sa, NULL) < 0)
        err(1, "sigaction(SIGTERM)");

if ((sfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        err(1, "socket(AF_INET)");
memset(&sin, 0, sizeof(sin));
sin.sin_family = AF_INET;
sin.sin_port = htons(port);
sin.sin_addr.s_addr = INADDR_ANY;
if (bind(sfd, (struct sockaddr *)&sin, sizeof(sin)) < 0)
        err(1, "connect");
if (listen(sfd, backlog) < 0)
        err(1, "listen");

for (;;) {
        if (f_quit)
                break;

        if ((cfd = accept(sfd, NULL, NULL)) < 0)
                continue;
        printf("[%s] accepted client at sock: %d\n", argv0, cfd);

        switch (fork()) {
        case -1:
                err(1, "fork");
        case 0:
                serve_client(host, cfd);
                _exit(0);
        default:
                close(cfd);
        }

}
close(sfd);
```

```
        return (0);
}
```

4.4 sock_client.c

```
#include <sys/socket.h>
#include <sys/types.h>

#include <arpa/inet.h>
#include <netdb.h>
#include <netinet/in.h>

#include <err.h>
#include <libgen.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

static void     esend(int, void *, size_t, int);
static void     erecv(int, void *, size_t, int);
static void     *emalloc(size_t);
static void     safe_scanf(void *, char *, char *, ...);
static void     calc_avg(int);
static void     calc_minmax(int);
static void     calc_prod(int);
static void     usage(void);

static char     *argv0;

/*
 * Error checking send(2).
 */
static void
esend(int fd, void *msg, size_t len, int flags)
{
        if (send(fd, msg, len, flags) < 0)
                err(1, "send");
}

/*
 * Error checking recv(2).
 */
static void
erecv(int fd, void *msg, size_t len, int flags)
{
        if (recv(fd, msg, len, flags) < 0)
                err(1, "recv");
```

```
}

/*
 * Error checking malloc(3).
 */
static void *
emalloc(size_t nb)
{
        void *p;

        if ((p = malloc(nb)) == NULL)
                err(1, "malloc");
        return (p);
}

/*
 * The server might break if we give it incorrent input, so we have to make
 * sure we'll always read proper input from scanf() before we send it to the
 * server.
 */
static void
safe_scanf(void *n, char *type, char *fmt, ...)
{
        va_list ap;
        char buf[BUFSIZ];
        int rc;

        do {
                va_start(ap, fmt);
                vsprintf(buf, fmt, ap);
                va_end(ap);
                printf("\r%s", buf);
                rc = scanf(type, n);
                (void)getchar();
        } while (rc != 1);
}

static void
calc_avg(int fd)
{
        float res;
        int *arr, n, i;

        safe_scanf(&n, "%d", "%s> n: ", argv0);
        arr = emalloc(n * sizeof(int));
        for (i = 0; i < n; i++)
```

```
                safe_scanf(&arr[i], "%d", "%s> arr[%d]: ", argv0, i);

        esend(fd, &n, sizeof(int), 0);
        esend(fd, arr, n * sizeof(int), 0);

        erecv(fd, &res, sizeof(float), 0);
        printf("[%s] server response: %.3f\n", argv0, res);

        free(arr);
}

static void
calc_minmax(int fd)
{
        int res[2], *arr, n, i;

        safe_scanf(&n, "%d", "%s> n: ", argv0);
        arr = emalloc(n * sizeof(int));
        for (i = 0; i < n; i++)
                safe_scanf(&arr[i], "%d", "%s> arr[%d]: ", argv0, i);

        esend(fd, &n, sizeof(int), 0);
        esend(fd, arr, n * sizeof(int), 0);

        erecv(fd, &res, 2 * sizeof(int), 0);
        printf("[%s] server response: min: %d, max: %d\n",
            argv0, res[0], res[1]);

        free(arr);
}

static void
calc_prod(int fd)
{
        float *res, a;
        int *arr, n, i;

        safe_scanf(&a, "%f", "%s> a: ", argv0);
        safe_scanf(&n, "%d", "%s> n: ", argv0);
        arr = emalloc(n * sizeof(int));
        for (i = 0; i < n; i++)
                safe_scanf(&arr[i], "%d", "%s> arr[%d]: ", argv0, i);

        esend(fd, &a, sizeof(float), 0);
        esend(fd, &n, sizeof(int), 0);
        esend(fd, arr, n * sizeof(int), 0);
```

```
        res = emalloc(n * sizeof(float));
        erecv(fd, res, n * sizeof(float), 0);
        printf("[%s] server response: [", argv0);
        for (i = 0; i < n; i++)
                printf("%.3f%s", res[i], i == n - 1 ? "" : ", ");
        printf("]\n");

        free(arr);
        free(res);
}

static void
usage(void)
{
        fprintf(stderr, "usage: %s [-p port] hostname\n", argv0);
        exit(0);
}

int
main(int argc, char *argv[])
{
        struct sockaddr_in sin;
        struct hostent *hp;
        int port = 9999;
        int fd, n;
        char *host, ch;

        argv0 = basename(*argv);
        while ((ch = getopt(argc, argv, "p:")) != -1) {
                switch (ch) {
                case 'p':
                        if ((port = atoi(optarg)) < 1024)
                                errx(1, "can't use port number < 1024");
                        break;
                case '?':
                default:
                        usage();
                }
        }
        argc -= optind;
        argv += optind;

        if (argc < 1)
                usage();
        host = *argv;
```

```
        if ((fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
                err(1, "socket(AF_INET)");
        memset(&sin, 0, sizeof(sin));
        sin.sin_family = AF_INET;
        sin.sin_port = htons(port);
        if (!inet_aton(host, &sin.sin_addr)) {
                if ((hp = gethostbyname(host)) == NULL)
                        errx(1, "gethostbyname(%s) failed", host);
                (void)memcpy(&sin.sin_addr, hp->h_addr, hp->h_length);
        }
        if (connect(fd, (struct sockaddr *)&sin, sizeof(sin)) < 0)
                err(1, "connect");

        for (;;) {
                printf("1: average\n2: min and max\n3: product\n4: disconnect\n");
                safe_scanf(&n, "%d", "%s> your choice: ", argv0);

                switch (n) {
                case 1:
                        /*
                         * Send our choice to the server so that it can read at
                         * the appropriate RPC.
                         */
                        esend(fd, &n, sizeof(int), 0);
                        calc_avg(fd);
                        break;
                case 2:
                        esend(fd, &n, sizeof(int), 0);
                        calc_minmax(fd);
                        break;
                case 3:
                        esend(fd, &n, sizeof(int), 0);
                        calc_prod(fd);
                        break;
                case 4:
                        esend(fd, &n, sizeof(int), 0);
                        goto end;
                default:
                        printf("[%s] invalid choice\n", argv0);
                }
        }
end:
        close(fd);

        return (0);
```

```
}
```

4.5 Makefile.rpc

```
# This is a template makefile generated         by rpcgen

# Parameters

CLIENT = rpc_client
SERVER = rpc_server

SOURCES_CLNT.c =
SOURCES_CLNT.h =
SOURCES_SVC.c =
SOURCES_SVC.h =
SOURCES.x = rpc.x

TARGETS_SVC.c = rpc_server.c rpc_svc.c rpc_xdr.c
TARGETS_CLNT.c = rpc_client.c rpc_clnt.c rpc_xdr.c
TARGETS = rpc.h rpc_xdr.c rpc_clnt.c rpc_svc.c

OBJECTS_CLNT = $(SOURCES_CLNT.c:%.c=%.o) $(TARGETS_CLNT.c:%.c=%.o)
OBJECTS_SVC = $(SOURCES_SVC.c:%.c=%.o) $(TARGETS_SVC.c:%.c=%.o)
# Compiler flags
CFLAGS += -g -DRPC_SVC_FG
RPCGENFLAGS = -C

# Targets

all : $(CLIENT) $(SERVER)

$(TARGETS) : $(SOURCES.x)
        rpcgen $(RPCGENFLAGS) $(SOURCES.x)

$(OBJECTS_CLNT) : $(SOURCES_CLNT.c) $(SOURCES_CLNT.h) $(TARGETS_CLNT.c)

$(OBJECTS_SVC) : $(SOURCES_SVC.c) $(SOURCES_SVC.h) $(TARGETS_SVC.c)

$(CLIENT) : $(OBJECTS_CLNT)
        $(CC) -o $(CLIENT) $(OBJECTS_CLNT) $(LDLIBS) $(CFLAGS)

$(SERVER) : $(OBJECTS_SVC)
        $(CC) -o $(SERVER) $(OBJECTS_SVC) $(LDLIBS) $(CFLAGS)

clean:
         rm -f core $(TARGETS) $(OBJECTS_CLNT) $(OBJECTS_SVC) $(CLIENT) $(SERVER)
```

4.6 Makefile

```
CC = cc

all:
        make -f Makefile.rpc
        ${CC} sock_client.c -o sock_client

clean:
        make -f Makefile.rpc clean
        rm -f sock_client *.core
```