More useful functions

`shutdown` shut down part of a full-duplex connection

```
int shutdown(int socket, int how);
```

Can be used to tell server that we have sent the whole request.

`getsockname` returns the current address to which the `socket` is bound with using the buffer pointed to by `address`.

```
int getsockname(int socket,
                struct sockaddr *address,
                socklen_t *address_len);
```

`getpeername` get the name (`address`) of the peer connected to a `socket`; useful if a server has called a `fork/exec` combination and only the `socket` is known.

```
int getpeername(int socket,
                struct sockaddr *address,
                socklen_t *address_len);
```

Parsing and Printing Addresses

`inet_ntoa` Convert struct `in_addr` to printable form 'a.b.c.d'

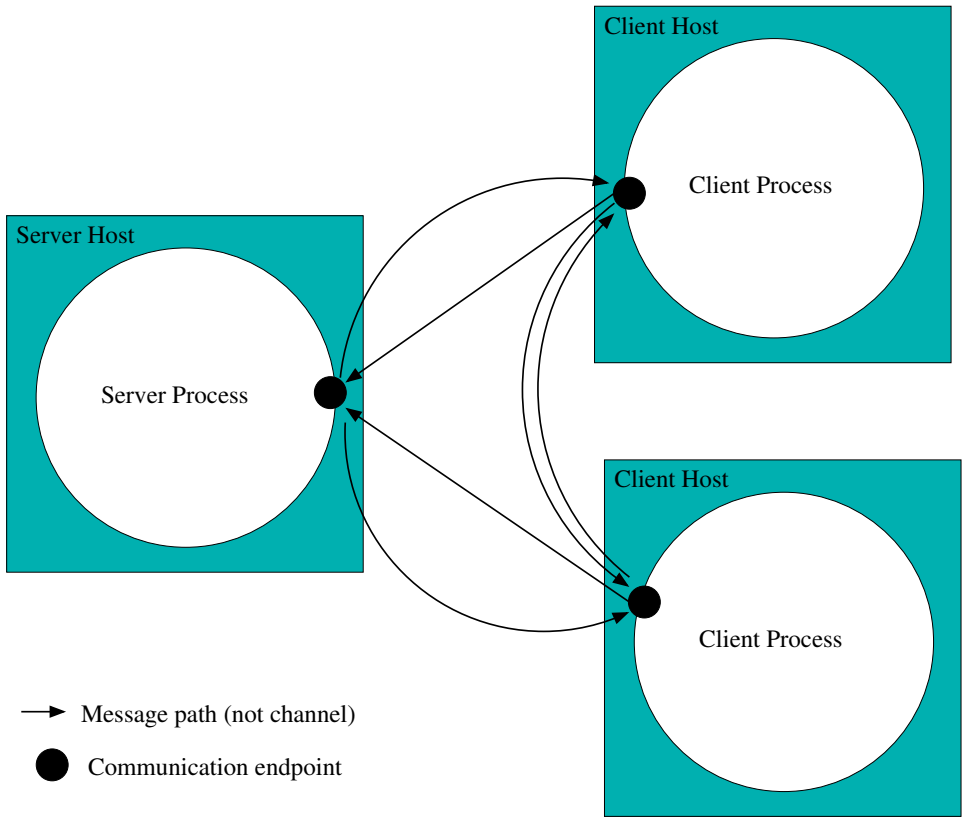
`inet_addr` Convert IP address string in '.' notation to 32bit network address

`inet_ntop` Convert address from network format to printable presentation format

`inet_pton` Convert presentation format address to network format

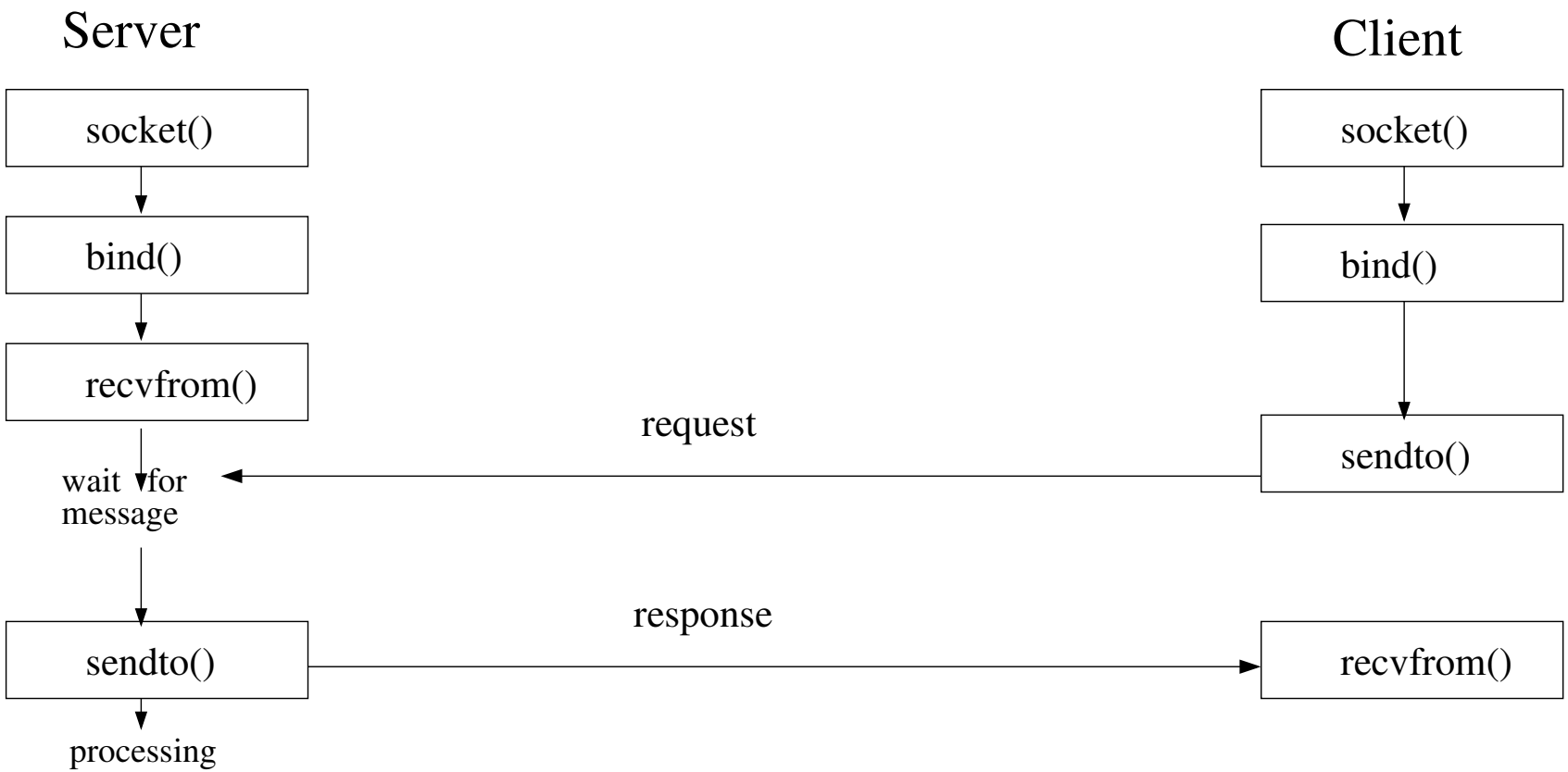
Bonus: `inet_ntop` and `inet_pton` also work with IPv6!

Internet User Datagram Protocol (UDP)



- No Connections: Think postcards, not telephone.
- `Datagrams` (messages) exchanged.
- Datagrams *either* arrive (possibly out of order) *or* get lost!

UDP communication



sendto, recvfrom

```
ssize_t sendto(int sock, void *buff, size_t length,
               int flags, struct sockaddr *dest_addr,
               socklen_t dest_len);
```

- Send a message to a socket
- Similar to `write()` & `send()` but designates destination

```
ssize_t recvfrom(int socket, void *buff,size_t length,
               int flags, struct sockaddr *address,
               socklen_t *address_len);
```

- Receive a message from a socket
- Similar to `read()` & `recv()` but designates source address
- `address_len` is value-result and must be initialized to the size of the buffer pointed to by the address pointer
- last two arguments can be `NULL`

Usually `flags = 0` ; rarely used (ex. out of band data)

A simple echoing UDP server

Client on linux03 (along with the input to send out):

```
antoulas@linux03:~> fortune | ./inet_dgm_client linux02 59579
Hlade's Law:
    If you have a difficult task, give it to a lazy person --
    they will find an easier way to do it.
antoulas@linux03:~>
```

Server on linux02 (along with the bytes to receive from client):

```
antoulas@linux02:~> ./inet_dgm_server
Socket port: 59579
Received from linux03: Hlade's Law:
Received from linux03:  If you have a difficult task, give it to a lazy person
     --
Received from linux03:  they will find an easier way to do it.
```

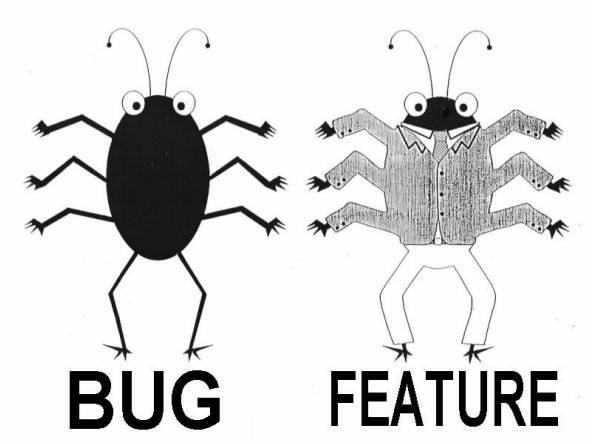
```c
/* inet_dgr_server.c: Internet datagram sockets server */
#include <sys/types.h>                          /* sockets */
#include <sys/socket.h>                         /* sockets */
#include <netinet/in.h>                /* Internet sockets */
#include <netdb.h>                        /* gethostbyaddr */
#include <arpa/inet.h>                        /* inet_ntoa */
#include <stdio.h>
#include <stdlib.h>
void perror_exit(char *message);

char *name_from_address(struct in_addr addr) {
    struct hostent *rem; int asize = sizeof(addr.s_addr);
    if((rem = gethostbyaddr(&addr.s_addr, asize, AF_INET)))
        return rem->h_name;  /* reverse lookup success */
    return inet_ntoa(addr); /* fallback to a.b.c.d form */
}

void main() {
    int n, sock; unsigned int serverlen, clientlen;
    char buf[256], *clientname;
    struct sockaddr_in server, client;
    struct sockaddr *serverptr = (struct sockaddr*) &server;
    struct sockaddr *clientptr = (struct sockaddr*) &client;
    /* Create datagram socket */
```

```
    if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
        perror_exit("socket");
    /* Bind socket to address */
    server.sin_family = AF_INET;        /* Internet domain */
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    server.sin_port = htons(0);         /* Autoselect a port */
    serverlen = sizeof(server);
    if (bind(sock, serverptr, serverlen) < 0)
        perror_exit("bind");
    /* Discover selected port */
    if (getsockname(sock, serverptr, &serverlen) < 0)
        perror_exit("getsockname");
    printf("Socket port: %d\n", ntohs(server.sin_port));
    while(1) { clientlen = sizeof(client);
        /* Receive message */
        if ((n = recvfrom(sock, buf, sizeof(buf), 0, clientptr, &clientlen)) <
             0)
            perror("recvfrom");
        buf[sizeof(buf)-1]='\0'; /* force str termination */
        /* Try to discover client's name */
        clientname = name_from_address(client.sin_addr);
        printf("Received from %s: %s\n", clientname, buf);
        /* Send message */
        if (sendto(sock, buf, n, 0, clientptr, clientlen)<0)
            perror_exit("sendto");
    }}

void perror_exit(char *message)
{
    perror(message);
    exit(EXIT_FAILURE);
}
```

```c
/* inet_dgr_client.c: Internet datagram sockets client    */
#include <sys/types.h>                          /* sockets */
#include <sys/socket.h>                         /* sockets */
#include <netinet/in.h>                /* Internet sockets */
#include <netdb.h>                        /* gethostbyname */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void main(int argc, char *argv[]) {
   int sock; char buf[256]; struct hostent *rem;
   struct sockaddr_in server, client;
   unsigned int serverlen = sizeof(server);
   struct sockaddr *serverptr = (struct sockaddr *) &server;
   struct sockaddr *clientptr = (struct sockaddr *) &client;
   if (argc < 3) {
      printf("Please give host name and port\n"); exit(1);}
   /* Create socket */
   if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
      perror("socket"); exit(1); }
   /* Find server's IP address */
   if ((rem = gethostbyname(argv[1])) == NULL) {
      herror("gethostbyname"); exit(1); }
```

```
   /* Setup server's IP address and port */
   server.sin_family = AF_INET;          /* Internet domain */
   memcpy(&server.sin_addr, rem->h_addr, rem->h_length);
   server.sin_port = htons(atoi(argv[2]));
   /* Setup my address */
   client.sin_family = AF_INET;          /* Internet domain */
   client.sin_addr.s_addr=htonl(INADDR_ANY); /*Any address*/
   client.sin_port = htons(0);           /* Autoselect port */
   /* Bind my socket to my address*/
   if (bind(sock, clientptr, sizeof(client)) < 0) {
      perror("bind"); exit(1); }
   /* Read continuously messages from stdin */
   while (fgets(buf, sizeof buf, stdin)) {
      buf[strlen(buf)-1] = '\0';              /* Remove '\n' */
      if (sendto(sock, buf, strlen(buf)+1, 0, serverptr, serverlen) < 0) {
         perror("sendto"); exit(1); }         /* Send message */
      bzero(buf, sizeof buf);                 /* Erase buffer */
      if (recvfrom(sock, buf, sizeof(buf), 0, NULL, NULL) < 0) {
         perror("recvfrom"); exit(1); }       /* Receive message */
      printf("%s\n", buf);
      }
}
```

- Everything looks good and runs ok BUT there is a BUG!
- Remember that UDP is *unreliable*



rlsd: a remote file/dir-listing server

Server on linux02:

```
antoulas@linux02:~> ./rlsd
```

Client on linux03:

```
antoulas@linux03:~> ./rls linux02.di.uoa.gr /usr/share/dict
README
connectives
propernames
web2
web2a
words
antoulas@linux03:~>
```

`rlsd.c` remote `ls` server (TCP)

`fdopen` allows buffered I/O by opening socket as file stream

```c
/* rlsd.c - a remote ls server - with paranoia */
#include  <stdio.h>
#include  <stdlib.h>
#include  <unistd.h>
#include  <sys/types.h>
#include  <sys/socket.h>
#include  <netinet/in.h>
#include  <netdb.h>
#include  <time.h>
#include  <string.h>
#include  <ctype.h>
#define PORTNUM  15000        /* rlsd listens on this port */

void perror_exit(char *msg);
void sanitize(char *str);
```

```c
int main(int argc, char *argv[]) {
    struct sockaddr_in myaddr;  /* build our address here */
    int c, lsock, csock;  /* listening and client sockets */
    FILE    *sock_fp;             /* stream for socket IO */
    FILE    *pipe_fp;              /* use popen to run ls */
    char    dirname[BUFSIZ];              /* from client */
    char    command[BUFSIZ];              /* for popen() */

    /** create a TCP a socket **/
    if ((lsock = socket( AF_INET, SOCK_STREAM, 0)) < 0)
        perror_exit( "socket" );
    /** bind address to socket. **/
    myaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    myaddr.sin_port = htons(PORTNUM);/*port to bind socket*/
    myaddr.sin_family = AF_INET;  /* internet addr family */
    if(bind(lsock,(struct sockaddr *)&myaddr, sizeof(myaddr)))
          perror_exit( "bind" );
    /** listen for connections with Qsize=5 **/
    if ( listen(lsock, 5) != 0 )
        perror_exit( "listen" );
```

```
    while ( 1 ){ /* main loop: accept - read - write */
            /* accept connection, ignore client address */
            if ( (csock = accept(lsock, NULL, NULL)) < 0 )
                perror_exit("accept");
            /* open socket as buffered stream */
            if ((sock_fp = fdopen(csock,"r+")) == NULL)
                perror_exit("fdopen");
            /* read dirname and build ls command line */
            if (fgets(dirname, BUFSIZ, sock_fp) == NULL)
                perror_exit("reading dirname");
            sanitize(dirname);      /* clear wild characters */
            snprintf(command, BUFSIZ, "ls %s", dirname);
            /* Invoke ls through popen */
            if ((pipe_fp = popen(command, "r")) == NULL )
                perror_exit("popen");
            /* transfer data from ls to socket */
            while( (c = getc(pipe_fp)) != EOF )
                putc(c, sock_fp);
            pclose(pipe_fp);
            fclose(sock_fp);
    }
    return 0;
}
```

```c
/* it would be very bad if someone passed us an dirname like
 * "; rm *"  and we naively created a command  "ls ; rm *".
 * So..we remove everything but slashes and alphanumerics.
 */
void sanitize(char *str)
{
    char *src, *dest;
    for ( src = dest = str ; *src ; src++ )
        if ( *src == '/' || isalnum(*src) )
            *dest++ = *src;
    *dest = '\0';
}

/* Print error message and exit */
void perror_exit(char *message)
{
    perror(message);
    exit(EXIT_FAILURE);
}
```

rls.c sends a directory name to rlsd and reads back a directory listing (TCP)

write_all guarantees to send all the bytes requested, provided no error occurs, by repeatedly calling `write()`

```
#include <sys/types.h>                          /* sockets */
#include <sys/socket.h>                         /* sockets */
#include <netinet/in.h>                 /* internet sockets */
#include <netdb.h>                         /* gethostbyname */
#define  PORTNUM 15000
#define  BUFFSIZE 256
void perror_exit(char *msg);

/* Write() repeatedly until 'size' bytes are written */
int write_all(int fd, void *buff, size_t size) {
    int sent, n;
    for(sent = 0; sent < size; sent+=n) {
        if ((n = write(fd, buff+sent, size-sent)) == -1)
            return -1; /* error */
    }
    return sent;
}
```

```c
int main(int argc, char *argv[]) {
    struct sockaddr_in  servadd; /* The address of server */
    struct hostent *hp;          /* to resolve server ip */
    int     sock, n_read;     /* socket and message length */
    char    buffer[BUFFSIZE];        /* to receive message */

    if ( argc != 3 ) {
        puts("Usage: rls <hostname> <directory>");exit(1);}
    /* Step 1: Get a socket */
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1 )
        perror_exit( "socket" );
    /* Step 2: lookup server's address and connect there */
    if ((hp = gethostbyname(argv[1])) == NULL) {
        herror("gethostbyname"); exit(1);}
    memcpy(&servadd.sin_addr, hp->h_addr, hp->h_length);
    servadd.sin_port = htons(PORTNUM); /* set port number */
    servadd.sin_family = AF_INET ;     /* set socket type */
    if (connect(sock, (struct sockaddr*) &servadd, sizeof(servadd)) !=0)
        perror_exit( "connect" );
    /* Step 3: send directory name + newline */
```

```
    if ( write_all(sock, argv[2], strlen(argv[2])) == -1)
        perror_exit("write");
    if ( write_all(sock, "\n", 1) == -1 )
        perror_exit("write");
    /* Step 4: read back results and send them to stdout */
    while( (n_read = read(sock, buffer, BUFFSIZE)) > 0 )
        if (write_all(STDOUT_FILENO, buffer, n_read)<n_read)
            perror_exit("fwrite");
    close(sock);
    return 0;
}
```

The Rock–Paper–Scissors zero-sum game:

- One referee process.
- Two players: a local process, a remote process
- Referee talks to the local process through pipes
- Referee talks to the remote process through sockets

Server

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>           /* For wait */
#include <sys/types.h>          /* For sockets */
#include <sys/socket.h>         /* For sockets */
#include <netinet/in.h>         /* For Internet sockets */
#include <netdb.h>          /* For gethostbyname */

#define READ    0
#define WRITE   1

int read_data (int fd, char *buffer);
int write_data (int fd, char* message);
void prs (int *score1, int *score2, int len1, int len2);
```

```c
int main(int argc, char *argv[])
{
    int n, port, sock, newsock;
    int i, pid, fd1[2], fd2[2], option, status;
    int  score1=0, score2=0;                    /* Score variables */
    char buf[60], buf2[60], buf3[60];             /* Buffers */
    char *message[] = { "ROCK", "PAPER", "SCISSORS" }; /* prs options */

    unsigned int serverlen, clientlen;  /* Server - client variables */
    struct sockaddr_in server, client;
    struct sockaddr *serverptr, *clientptr;
    struct hostent *rem;

    if ( argc < 3 ){              /* At least 2 arguments */
        fprintf(stderr, "usage: %s <n> <port>\n", argv[0]);
        exit(0);
    }

    n = atoi(argv[1]);            /* Number of games */
    port = atoi(argv[2]);             /* Port */
```

```
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1){     /* Create socket */
    perror("socket");
    exit(-1);
}
server.sin_family = AF_INET;     /* Internet domain */
server.sin_addr.s_addr = htonl(INADDR_ANY);
server.sin_port = htons(port);  /* The given port */
serverptr = (struct sockaddr *) &server;
serverlen = sizeof server;
if (bind(sock, serverptr, serverlen) < 0){
    perror("bind"); exit(-1);
}
if (listen(sock, 5) < 0){
    perror("listen");exit(-1);
}

printf("I am the referee with PID %d waiting for game request at port %d\n",
      (int) getpid(), port);
```

```
    if (pipe (fd1) == -1){  /* First pipe: parent -> child */
        perror("pipe");exit(-1);
    }
    if (pipe (fd2) == -1){  /* Second pipe: child -> parent */
        perror("pipe");exit(-1);
    }

    if ((pid = fork()) == -1)     /* Create child for player 1 */
    {
        perror("fork");exit(-1);
    }
```

```
if ( !pid ){          /* Child process */
    close(fd1[WRITE]);close(fd2[READ]); /* Close unused*/
    srand (getppid());
    printf("I am player 1 with PID %d\n", (int) getpid());
    for(;;){     /* While read "READY" */
        /* Read "READY" or "STOP" */
        read_data (fd1[READ], buf);
        option = rand()%3;
        if ( strcmp("STOP", buf)){  /* If != "STOP" */
            write_data (fd2[WRITE], message[option]);
            /* Send random option */
            read_data (fd1[READ], buf);
            /* Read result of this game */
            printf ("%s", buf);
            /* Print result */
            }
        else break;
        }
    /* Read final result */
```

```
        read_data (fd1[READ], buf);
        /* Print final result */
        printf("%s", buf);
        close(fd1[READ]); close(fd2[WRITE]);
        }
    else {      /* Parent process */
        clientptr = (struct sockaddr *) &client;
        clientlen = sizeof client;
        close(fd1[READ]); close(fd2[WRITE]);
        printf("Player 1 is child of the referee\n");
        if ((newsock = accept(sock, clientptr, &clientlen)) < 0){
```

```
        perror("accept"); exit(-1);
    }
    if ((rem = gethostbyaddr((char *) &client.sin_addr.s_addr, sizeof client
        .sin_addr.s_addr, client.sin_family)) == NULL) {
        perror("gethostbyaddr");exit(-1);
    }

    printf("Player 2 connected %s\n",rem->h_name);
    write_data (newsock, "2"); /* Send player's ID (2) */
    for(i = 1; i <= n; i++){
        write_data (fd1[WRITE], "READY");
        write_data (newsock, "READY");
        read_data (fd2[READ], buf);
        read_data (newsock, buf2);
        /* Create result string */
```

```
            write_data (fd1[WRITE], buf3);
            write_data (newsock, buf3);
            prs(&score1,&score2,strlen(buf),strlen(buf2));
        }

        /* Calculate final results for each player */
        if ( score1 == score2 ){
            sprintf(buf, "Score = %d - %d (draw)\n", score1, score2);
            sprintf(buf2, "Score = %d - %d (draw)\n", score1, score2);
        }else if (score1 > score2 ){
            sprintf(buf, "Score = %d - %d (you won)\n", score1, score2);
            sprintf(buf2, "Score = %d - %d (player 1 won)\n", score1, score2);
        }else{
            sprintf(buf, "Score = %d - %d (player 2 won)\n", score1, score2);
            sprintf(buf2, "Score = %d - %d (you won)\n", score1, score2);
        }
        write_data (fd1[WRITE], "STOP");
        write_data (fd1[WRITE], buf);
```

```
        close(newsock); /* Close socket */
    }
    return 0;
}

int read_data (int fd, char *buffer){/* Read formated data */
    char temp;int i = 0, length = 0;
    if ( read ( fd, &temp, 1 ) < 0 )    /* Get length of string */
        exit (-3);
    length = temp;
    while ( i < length )    /* Read $length chars */
        if ( i < ( i+= read (fd, &buffer[i], length - i)))
            exit (-3);
    return i;   /* Return size of string */
}

int write_data ( int fd, char* message ){/* Write formated data */
    char temp; int length = 0;
    length = strlen(message) + 1;   /* Find length of string */
    temp = length;
```

```
*    PAPER wins ROCK, ROCK wins SCISSORS and SCISSORS win PAPER.
*    This means, for the 1st player to be the winner the difference in the
*    number of letters must be equal to 3 (SCISSORS-PAPER) or 1 (PAPER-ROCK)
*    or -4 (ROCK-SCISSORS). If not, then the 2nd player wins!
*    (If we have a zero, then we call it a draw and nobody get points)
*/
void prs(int *score1, int *score2, int len1, int len2)  /* Calculate score */
{
    int result = len1 - len2; /* len1 = buf1 length, len2 = buf2 length */
    if (result == 3 || result == 1 || result == -4) /* 1st player wins */
```

Client

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>        /* For bcopy */
#include <unistd.h>
#include <sys/wait.h>       /* For wait */
#include <sys/types.h>      /* For sockets */
#include <sys/socket.h>     /* For sockets */
#include <netinet/in.h>     /* For Internet sockets */
#include <netdb.h>      /* For gethostbyname */

int read_data (int fd, char *buffer);
int write_data (int fd, char* message);
```

```c
int main (int argc, char *argv[])
{
    int i, port, sock, option;
    char opt[3], buf[60], *message[] = { "PAPER", "ROCK", "SCISSORS" };
    unsigned int serverlen;
    struct sockaddr_in server;
    struct sockaddr *serverptr;
    struct hostent *rem;
    if (argc < 3){  /* At least 2 arguments */
        fprintf(stderr, "usage: %s <domain> <port>\n", argv[0]);
        exit(-1);
    }
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0){
        perror("socket");
        exit(-1);
    }
    /* Find server address */
    if ((rem = gethostbyname(argv[1])) == NULL){
        perror("gethostbyname");
        exit(-1);
```

```
    }
    port = atoi(argv[2]);
    server.sin_family = AF_INET;
    bcopy((char *) rem -> h_addr, (char *) &server.sin_addr, rem -> h_length);
    server.sin_port = htons(port);
    serverptr = (struct sockaddr *) &server;
    serverlen = sizeof server;
    if (connect(sock, serverptr, serverlen) < 0){
        perror("connect");exit(-1);
    }
```

```
read_data (sock, buf); /* Read player's ID (1 or 2) */
printf("I am player %d with PID %d\n", buf[0]-'0', (int) getpid());
for ( i = 1; ; i++ ){/* While read "READY" */
    read_data (sock, buf);  /* Read "READY" or "STOP" */
    if ( strcmp("STOP", buf) ){ /* If != "STOP" */
        printf("Give round %d play: ", i);
        scanf("%s", opt);
        switch (*opt){  /* First letter of opt */
        /* Note: The other 2 are \n and \0 */
            case 'p':option = 0; break;
            case 'r':option = 1; break;
            case 's':option = 2; break;
            default: fprintf(stderr, "Wrong option %c\n", *opt);
                option = ((int)*opt)%3; break;
        }
        write_data (sock, message[option]);
        read_data (sock, buf);
        printf ("%s", buf);
    }else break;
}
read_data (sock, buf);  /* Read final score */
printf("%s", buf);
close(sock);
```

Server

```
jackal@jackal-laptop:~/Set006/src
jackal@jackal-laptop:~/Set006/src$ ./prsref 3 2323
I am the referee with PID 4587 waiting for game request at port 2323
I am player 1 with PID 4588
Player 1 is child of the referee
Player 2 connected localhost
Player 1:     PAPER     Player 2:     PAPER
Player 1:  SCISSORS     Player 2:  SCISSORS
Player 1:      ROCK     Player 2:  SCISSORS
Score = 1 - 0 (you won)
jackal@jackal-laptop:~/Set006/src
```

Client

```
jackal@jackal-laptop:~/Set006/src
jackal@jackal-laptop:~/Set006/src$ ./prs localhost 2323
I am player 2 with PID 4615
Give round 1 play: p
Player 1:     PAPER     Player 2:     PAPER
Give round 2 play: s
Player 1:  SCISSORS     Player 2:  SCISSORS
Give round 3 play: s
Player 1:      ROCK     Player 2:  SCISSORS
Score = 1 - 0 (player 1 won)
jackal@jackal-laptop:~/Set006/src
```