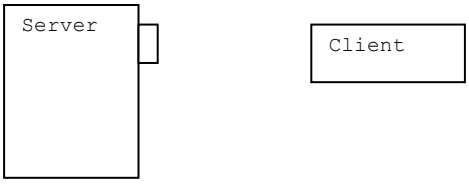
Forking executing routins:

This routine forks off one server and four client processes.

```
#include <stdio.h>

main ( )

int n, status;

if (fork() == 0)                        /* Create a server process */
        {
            execl (“server”, “server”, 0);              /* Relative path name */
        }
        else
        {
            for (n = 1; n <= 4; n++)
            {
                if (fork ( ) == 0)
                {
                      execl ("client", "client”, 0);/* Relative path name
                 }
            }

for (n = 1; n <= 5; n++)                        /* Wait for the children to exit */
                 wait (&status);
}
```

Forming A Socket Connection: Stage 1

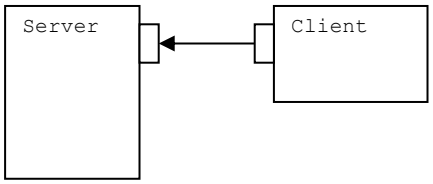


The server presents a named socket and waits to accept incomming requests

First Part Of Server Process Code

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

main ( )
{
  char  ServerMessage[50] = "Welcome to my socket\n";

   int      server_fd,          /* Original server file descriptor */
            client_fd, /* Client's file descriptor */
            server_len;

   struct sockaddr   server;              /* defined in socket.h

  server_fd = socket (AF_UNIX, SOCK_STREAM, 0);      /* Create unnamed socket */
  server. sa_family = AF_UNIX;                              /* Prepare socket Id: type and name
  strcpy (server. sa_data, "MySocket");              /* Call it MySocket */
  server_len; = sizeof (server);

  bind (server_fd,  &server, server_len);                    /* Bind the socket to the name */

  listen (server_fd, 4);                /* Listen to the socket. Allow a maximum of four pending connections */

  client_fd = accept  (server_fd, &server, &server_len);     /* Accept a client connection - wait until one is received */
  . . .
}
```

Socket Connection: Stag 2

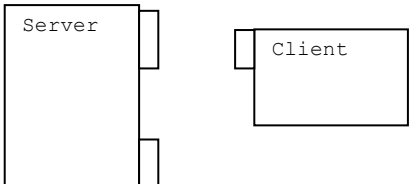


A Client process that knows the name of the Server socket requests a connection

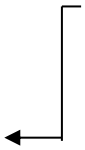
first Part Of Client Process Cod

```
Main ( )
{  char ClientMessage[50] = "I like your socket\n";
   int    socket_fd,  result;
   struct sockaddr  client;

   socket_fd = socket (AF_UNIX, SOCK_STREAM, 0);      /* Create an unnamed socket */

   strcpy (client.sa_data, "MySocket");               /* Assign structure describing server socket */
   client.sa_family = AF_UNIX;

   do                                        /* Busy wait: loop until successful connection made */
   {
     result = connect (socket_fd, &client, sizeof (client));      /* Attempt to connect the unnamed socket to the server */
   }  while (result == -1);
. . .
}
```

Socket Connection: Stage 3



The Server grants the request, creating a new unnamed socket to act as an end node, freeing the named socket to accept new clients.



Second Part Of Server process Code

The server can then communicate with the client via -read ( ) and *write ( )* calls, and/or continue to accept connections on the same server_fd socket, possibly after forking...
ReadSocket and WriteSocket are given to make it easier to send and receive messages through sockets.

```
WriteSocket (client_fd, ServerMessage);      /* Send ServerMessage to client */
 ReadSocket(client_fd, Reply);               /* read the reply back from the client */
close (client_fd);                           /* Close server end of client's socket */
 . . .
```

Second Part Of Client ProcessCode

The client can then communicate with the server via *read* ( ) and write ( ) calls...

```
write (socket_fd, ClientMessage, strlen (ClientMessage));      /* Send message to server (WriteSocket is better)*/
. . .
close (socket_fd);                                             /* Close client end of socket */
```

Utility routines ReadSocket and WriteSocket

These routines allow easy line-by-line I/O via a socket connection, and are linked into player.c and referee.c. Please note that you must have a "\n" at the end of each string for this routine to work.

```
  ReadSocket (int fd, char *str)

  /* Read characters and build a string until the NULL is read. This "chunks" input into digestible strings */
  do
  {
       read (fd, str, 1);
  }   while (*str++ != '\n');

  *str = '\0';                  /* NULL Terminator */
}

WriteSocket  (int fd, char *str)              /* Write a string */
{
  write (fd, str, strlen (str));
}
```