```
/*
                  NETWORK PROGRAMMING WITH SOCKETS

In this program we illustrate the use of Berkeley sockets for interprocess
communication across the network. We show the communication between a server
process and a client process.

Since many server processes may be running in a system, we identify the
desired server process by a "port number". Standard server processes have
a worldwide unique port number associated with it. For example, the port
number of SMTP (the sendmail process) is 25. To see a list of server
processes and their port numbers see the file /etc/services

In this program, we choose port number 6000 for our server process. Here we
shall demonstrate TCP connections only. For details and for other types of
connections see:

       Unix Network Programming
            -- W. Richard Stevens, Prentice Hall India.

To create a TCP server process, we first need to open a "socket" using the
socket() system call. This is similar to opening a file, and returns a socket
descriptor. The socket is then bound to the desired port number. After this
the process waits to "accept" client connections.

*/

#include <stdio.h>
#include <sys/types.h>

/* The following three files must be included for network programming */
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

                  /* THE SERVER PROCESS */

      /* Compile this program with cc server.c -o server
         and then execute it as ./server &
      */
main()
{
      int                 sockfd, newsockfd ; /* Socket descriptors */
      int                 clilen;
      struct sockaddr_in        cli_addr, serv_addr;

      int i;
      char buf[100];            /* We will use this buffer for communication */

      /* The following system call opens a socket. The first parameter
         indicates the family of the protocol to be followed. For internet
         protocols we use AF_INET. For TCP sockets the second parameter
         is SOCK_STREAM. The third parameter is set to 0 for user
         applications.
      */
      if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
            printf("Cannot create socket\n");
      exit(0);
}

/* The structure "sockaddr_in" is defined in <netinet/in.h> for the
   internet family of protocols. This has three main fields. The
   field "sin_family" specifies the family and is therefore AF_INET
   for the internet family. The field "sin_addr" specifies the
   internet address of the server. This field is set to INADDR_ANY
   for machines having a single IP address. The field "sin_port"
   specifies the port number of the server.
*/
serv_addr.sin_family          = AF_INET;
serv_addr.sin_addr.s_addr     = INADDR_ANY;
serv_addr.sin_port            = htons(6000);

/* With the information provided in serv_addr, we associate the server
   with its port using the bind() system call.
*/
if (bind(sockfd, (struct sockaddr *) &serv_addr,
                        sizeof(serv_addr)) < 0) {
      printf("Unable to bind local address\n");
      exit(0);
}

listen(sockfd, 5); /* This specifies that up to 5 concurrent client
                  requests will be queued up while the system is
                  executing the "accept" system call below.
               */

/* In this program we are illustrating a concurrent server -- one
   which forks to accept multiple client connections concurrently.
   As soon as the server accepts a connection from a client, it
   forks a child which communicates with the client, while the
   parent becomes free to accept a new connection. To facilitate
   this, the accept() system call returns a new socket descriptor
   which can be used by the child. The parent continues with the
   original socket descriptor.
*/
while (1) {

      /* The accept() system call accepts a client connection.
         It blocks the server until a client request comes.

         The accept() system call fills up the client's details
         in a struct sockaddr which is passed as a parameter.
         The length of the structure is noted in clilen. Note
         that the new socket descriptor returned by the accept()
         system call is stored in "newsockfd".
      */
      clilen = sizeof(cli_addr);
      newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr,
                        &clilen) ;

      if (newsockfd < 0) {
            printf("Accept error\n");
            exit(0);
      }
```

```
            /* Having successfully accepted a client connection, the
               server now forks. The parent closes the new socket
               descriptor and loops back to accept the next connection.
            */
            if (fork() == 0) {

                  /* This child process will now communicate with the
                     client through the send() and recv() system calls.
                  */
                  close(sockfd);    /* Close the old socket since all
                                 communications will be through
                                 the new socket.
                              */

                  /* We initialize the buffer, copy the message to it,
                     and send the message to the client.
                  */

                  strcpy(buf,"Message from server");
                  send(newsockfd, buf, strlen(buf) + 1, 0);

                  /* We again initialize the buffer, and receive a
                     message from the client.
                  */
                  for(i=0; i < 100; i++) buf[i] = '\0';
                  recv(newsockfd, buf, 100, 0);
                  printf("%s\n", buf);

                  close(newsockfd);
                  exit(0);
            }

            close(newsockfd);
      }
}
```