Socket Programming in C/C++: Handling multiple clients on server without multi threading

Difficulty Level : Hard • Last Updated : 30 May, 2018

This tutorial assumes you have a basic knowledge of socket programming, i.e you are familiar with basic server and client model. In the basic model, server handles only one client at a time, which is a big assumption if you want to develop any scalable server model.
The simple way to handle multiple clients would be to spawn new thread for every new client connected to the server. This method is strongly not recommended because of various disadvantages, namely:

- Threads are difficult to code, debug and sometimes they have unpredictable results.
- Overhead switching of context
- Not scalable for large number of clients
- Deadlocks can occur

Select()

A better way to handle multiple clients is by using **select()** linux command.

- Select command allows to monitor multiple file descriptors, waiting until one of the file descriptors become active.
- For example, if there is some data to be read on one of the sockets select will provide that information.

Select works like an interrupt handler, which gets activated as soon as any file descriptor sends any data.

```
fd_set readfds;

// Clear an fd_set
FD_ZERO(&readfds);

// Add a descriptor to an fd_set
FD_SET(master_sock, &readfds);

// Remove a descriptor from an fd_set
FD_CLR(master_sock, &readfds);

//If something happened on the master socket , then its an incoming
FD_ISSET(master_sock, &readfds);
```

Activating select: Please read the man page for select to check all the arguments for select command.

```
activity = select( max_fd + 1 , &readfds , NULL , NULL , NULL);
```

Implementation:

```
//Example code: A simple server side code, which echos back the received m
//Handle multiple socket connections with select and fd_set on Linux
#include <stdio.h>
#include <string.h>   //strlen
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>   //close
#include <arpa/inet.h>    //close
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/time.h> //FD_SET, FD_ISSET, FD_ZERO macros

#define TRUE   1
#define FALSE  0
#define PORT 8888

int main(int argc , char *argv[])
{
```

```
char buffer[1025];  //data buffer of 1K

//set of socket descriptors
fd_set readfds;

//a message
char *message = "ECHO Daemon v1.0 \r\n";

//initialise all client_socket[] to 0 so not checked
for (i = 0; i < max_clients; i++)
{
    client_socket[i] = 0;
}

//create a master socket
if( (master_socket = socket(AF_INET , SOCK_STREAM , 0)) == 0)
{
    perror("socket failed");
    exit(EXIT_FAILURE);
}

//set master socket to allow multiple connections ,
//this is just a good habit, it will work without this
if( setsockopt(master_socket, SOL_SOCKET, SO_REUSEADDR, (char *)&opt,
      sizeof(opt)) < 0 )
{
    perror("setsockopt");
    exit(EXIT_FAILURE);
}

//type of socket created
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons( PORT );

//bind the socket to localhost port 8888
if (bind(master_socket, (struct sockaddr *)&address, sizeof(address))<
{
    perror("bind failed");
    exit(EXIT_FAILURE);
}
printf("Listener on port %d \n", PORT);

//try to specify maximum of 3 pending connections for the master socke
if (listen(master_socket, 3) < 0)
{
    perror("listen");
```

```
while(TRUE)
{
    //clear the socket set
    FD_ZERO(&readfds);

    //add master socket to set
    FD_SET(master_socket, &readfds);
    max_sd = master_socket;

    //add child sockets to set
    for ( i = 0 ; i < max_clients ; i++)
    {
        //socket descriptor
        sd = client_socket[i];

        //if valid socket descriptor then add to read list
        if(sd > 0)
            FD_SET( sd , &readfds);

        //highest file descriptor number, need it for the select funct
        if(sd > max_sd)
            max_sd = sd;
    }

    //wait for an activity on one of the sockets , timeout is NULL ,
    //so wait indefinitely
    activity = select( max_sd + 1 , &readfds , NULL , NULL , NULL);

    if ((activity < 0) && (errno!=EINTR))
    {
        printf("select error");
    }

    //If something happened on the master socket ,
    //then its an incoming connection
    if (FD_ISSET(master_socket, &readfds))
    {
        if ((new_socket = accept(master_socket,
                (struct sockaddr *)&address, (socklen_t*)&addrlen))<0)
        {
            perror("accept");
            exit(EXIT_FAILURE);
        }

        //inform user of socket number - used in send and receive comm
        printf("New connection , socket fd is %d , ip is : %s , port :
```

```
            perror("send");
        }

        puts("Welcome message sent successfully");

        //add new socket to array of sockets
        for (i = 0; i < max_clients; i++)
        {
            //if position is empty
            if( client_socket[i] == 0 )
            {
                client_socket[i] = new_socket;
                printf("Adding to list of sockets as %d\n" , i);

                break;
            }
        }
    }

    //else its some IO operation on some other socket
    for (i = 0; i < max_clients; i++)
    {
        sd = client_socket[i];

        if (FD_ISSET( sd , &readfds))
        {
            //Check if it was for closing , and also read the
            //incoming message
            if ((valread = read( sd , buffer, 1024)) == 0)
            {
                //Somebody disconnected , get his details and print
                getpeername(sd , (struct sockaddr*)&address , \
                    (socklen_t*)&addrlen);
                printf("Host disconnected , ip %s , port %d \n" ,
                      inet_ntoa(address.sin_addr) , ntohs(address.sin_

                //Close the socket and mark as 0 in list for reuse
                close( sd );
                client_socket[i] = 0;
            }

            //Echo back the message that came in
            else
            {
                //set the string terminating NULL byte on the end
                //of the data read
                buffer[valread] = '\0';
```

```
    return 0;
}
```

Compile the file and run the server.
Use telnet to connect the server as a client.

Try running on different machines using following command:

```
telnet localhost 8888
```

Code Explanation:

- We have created a fd_set variable readfds, which will monitor all the active file descriptors of the clients plus that of the main server listening socket.
- Whenever a new client will connect, master_socket will be activated and a new fd will be open for that client. We will store its fd in our client_list and in the next iteration we will add it to the readfds to monitor for activity from this client.
- Similarly, if an old client sends some data, readfds will be activated and we will check from the list of existing client to see which client has send the data.
  Alternatives:
  There are other functions that can perform tasks similar to select. pselect , poll , ppoll

  This article is contributed by **Akshat Sinha**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

  Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.