International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 3 Issue: 4 2225 – 2229

TCP Concurrent Echo Program using Fork and Thread

Ms.Rupilaa V.M., Ms.Sangeetha M., Mr.Sathya Seelan K., Mr.Vadivel R.
Assistant Professor
Adithya Institute of Technology

Abstract – In networking, client-server model plays a vital role in exchanging information between processes. Client-server model predominantly relies on socket programming. Sockets allow communication between processes on same or different machines. Servers in the client-server model are of two types- Iterative and Concurrent. This paper describes about the elementary socket function for TCP client/server. An implementation of TCP Echo program for concurrent server using fork and thread is also given.

Keywords – *Socket, TCP, Concurrent server, fork, thread*

*****

I. INTRODUCTION

Rapid growth of Internet leads to the ultimate development of many net applications which use the client-server model. Fig. 1 shows the client-server model. Client-server model allows communication between processes or applications to exchange some information. The client process always initiates a connection to the server, while the server process always waits for requests from any client.

Fig. 1 Client-Server Model

Socket is used in client-server application framework. Sockets are of four types – Stream sockets, datagram sockets, raw sockets and sequenced-packet sockets [1]. A stream socket uses TCP as end-to-end protocol and of type SOCK_STREAM. Datagram socket uses UDP as end-to-end protocol and of type SOCK_DGRAM. A raw socket is of type SOCK_RAW and provides raw network protocol access. A sequenced-packet socket is similar to stream socket, with the exception that record boundaries are preserved and of type SOCK_SEQPACKET.

Servers are of two types in client-server model – Iterative server and concurrent server. Iterative server handles single request at a time and are easy to implement. Concurrent server handles multiple requests at a time but difficult to design and build.

Concurrent Server

Concurrent servers are designed using three basic mechanisms.

a. Process-based using fork
b. Event-based using I/O Multiplexing
c. Threads

a. Process-based using fork

- Spawn one server process to handle each client connection
- Kernel automatically interleaves multiple server processes
- Each server process has its own private address space

Fig. 2 shows the concurrent server implemented using fork(). In Fig. 2, client A has already established a connection with the server, which has created a child server process to handle the transaction. This allows the server to process client B request, without waiting for client A to complete

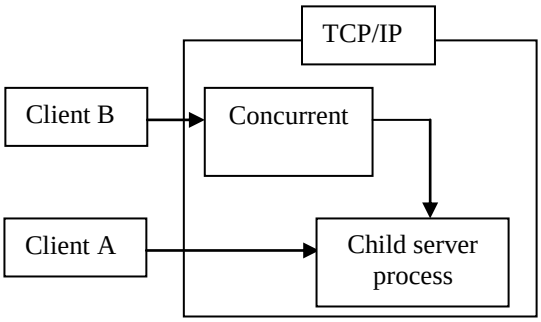

Fig. 2 Fork Execution Model

b. Event-based using I/O Multiplexing

- One process, one thread, but programmer manually interleaves multiple connections
- Relies on lower-level system abstractions

c. Threads

- Create one server thread to handle each client connection
- Kernel automatically interleaves multiple server threads
- All threads share the same address space

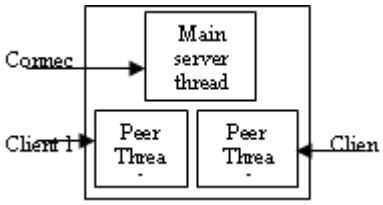

Fig. 3 Thread Execution Model

Fig. 3 shows the thread execution model. The main thread creates peer thread and handles multiple client requests.

This paper describes about the socket functions and thread functions used in TCP client-server communication in section 2. In section 3, the client-server model using socket function is given. Implementation of concurrent server using fork and thread is given in subsequent sections.

II. SOCKET AND THREAD FUNCTIONS

A. SOCKET FUNCTIONS

All the functions used in TCP client/server communication is defined in the header file #include<sys/socket.h>.

socket function – this function specifies the type of communication protocol. This function is used by both client and server.
Syntax: *int socket(int family, int type, int protocol);*

connect function – used by a TCP client to establish a connection with a TCP server.
Syntax: *int connect(int sockfd, const struct sockaddr *servaddr, socklen_t addrlen);*

bind function – assigns a local protocol address to a socket.
Syntax: *int bind(int sockfd, const struct sockaddr *myaddr, socklen_t addrlen);*

listen function – converts an unconnected socket into a passive socket and specifies the maximum number of connections the kernel should queue for this socket. This function is called only by TCP server. **Syntax:** *int listen(int sockfd, int backlog);*

accept function – return the next completed connection from the front of the completed connection queue.
Syntax: *int accept(int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);*

send function – send a message on a socket.
Syntax: *ssize_t send(int socket, const void *buffer, size_t length, int flags);*

recv function – receive data from a connected socket.
Syntax: *ssize_t recv(int socket, void *buffer, size_t length, int flags);*

close function – used to close a socket and terminate a TCP connection and included in the header file #include<unistd.h>
Syntax: *int close(int sockfd);*

B. THREAD FUNCTIONS

All the functions are defined in the header file #include<pthread.h>.

pthread_create – create a new thread.
Syntax: *int pthread_create(pthread_t *restrict thread, const pthread_attr_t *restrict attr, void *(*start_routine)(void*), void *restrict arg);*

pthread_self – obtain ID of the calling thread.
Syntax: *pthread_t pthread_self(void);*

pthread_detach – detach a thread.
Syntax: *int pthread_detach(pthread_t thread);*

pthread_exit – thread termination
Syntax: *void pthread_exit(void *value_ptr);*

III. ELEMENTARY TCP SOCKETS

Fig. 4 shows a timeline of the typical scenario that takes place between a TCP client and server. First, the server is started, and sometime later, the client is started and connects to the server. The client sends a request to the server, the server processes the request, and the server sends a reply back to the client. This continues until the client closes its end of the connection. The server then closes its end of the connection and either terminates or waits for a new client connection [3].

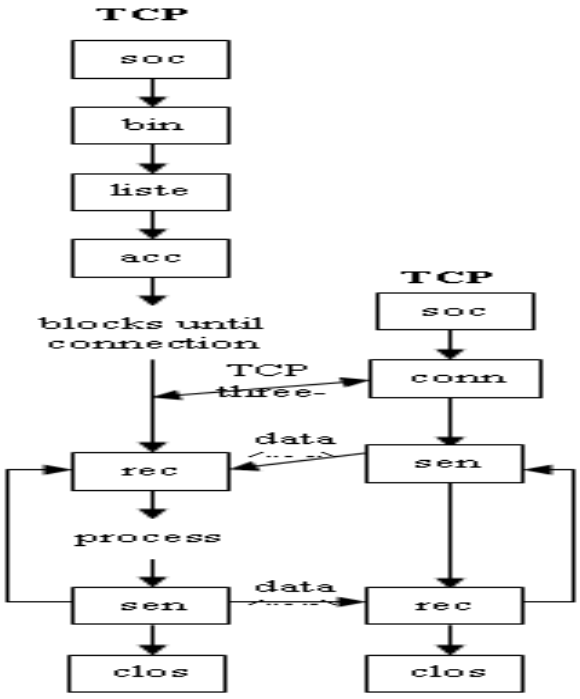

Fig. 4 Socket functions for elementary TCP client/server

 ISSN: 2321-8169
2225 – 2229

IV. TCP ECHO CLIENT/SERVER

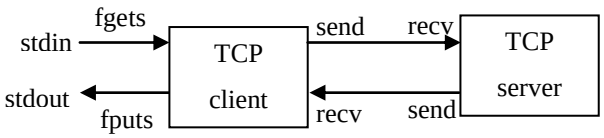

Fig. 5 Echo client/server

Fig. 5 depicts the echo client/server along with the functions used for input and output. An echo client/server program performs the following steps:

- The client reads a line of text from the standard input and writes the line to the server.
- The server reads the line from the network input and echoes the line back to the client.
- The client reads the echoed line and prints it on its standard output.

V. IMPLEMENTATION OF ECHO PROGRAM FOR CLIENT

```
#include<stdio.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
void str_echo(int s)
{
        char buf[50],buf1[50];
        puts("Enter the Message...");
        fgets(buf,50,stdin);

        send(s,buf,50,0);  //sending data to server

        //receiving data from server
        recv(s,buf1,50,0);
        puts("Message from Server...");
        fputs(buf1,stdout);
}

int main()
{
      int ls;
      struct sockaddr_in cli;
      puts("I am Client...");

         /*creating socket*/
      ls=socket(AF_INET,SOCK_STREAM,0);
      puts("Socket Created Successfully...");

         /*socket address structure*/
      cli.sin_family=AF_INET;
      cli.sin_addr.s_addr=inet_addr("127.0.0.1");
      cli.sin_port=htons(5000);

         /*connecting to server*/
      connect(ls,(struct   sockaddr*)&cli,sizeof(cli));
      puts("Connected with Server...");

      str_echo(ls);
      close(ls);
      return 0;
}
```

VI. IMPLEMENTATION OF ECHO PROGRAM FOR CONCURRENT SERVER USING FORK

```
#include<stdio.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<stdlib.h>

void str_echo(int s)
{
        char buf[50];

        //receiving data from client
        recv(s,buf,50,0);

        puts("Message from Client...");
        fputs(buf,stdout);
        send(s,buf,50,0);
}

int main()
{
      int ls,cs,len;
      struct sockaddr_in serv,cli;
      pid_t pid;

      puts("I am Server...");

         //creating socket
      ls=socket(AF_INET,SOCK_STREAM,0);
      puts("Socket Created Successfully...");

         //socket address structure
      serv.sin_family=AF_INET;
      serv.sin_addr.s_addr=INADDR_ANY;
      serv.sin_port=htons(5000);

      bind(ls,(struct sockaddr*)&serv,sizeof(serv));
      puts("Binding Done...");

      listen(ls,3);
      puts("Listening for Client...");

      for(; ;)
      {
         len=sizeof(cli);

         //accepting client connection
         cs=accept(ls,(struct sockaddr*)&cli,&len);
         puts("\nConnected to Client...");

         //creating child process
         if((pid=fork()) == 0)
         {
```

 ISSN: 2321-8169
2225 – 2229

```
                puts("Child process created...");
                close(ls);
                str_echo(cs);
                close(cs);
                exit(0);
        }
                close(cs);
    }
    return 0;

}
```

```
[root@localhost Desktop]# gcc client.c -o c
[root@localhost Desktop]# ./c
I am Client...
Socket Created Successfully...
Connected with Server...
Enter the Message...
socket programming
Message from Server...
socket programming
```

Fig. 6 Execution of Echo Client 1

```
[root@localhost Desktop]# gcc client.c -o c1
[root@localhost Desktop]# ./c1
I am Client...
Socket Created Successfully...
Connected with Server...
Enter the Message...
welcome
Message from Server...
welcome
```

Fig. 7 Execution of Echo Client 2

```
[root@localhost Desktop]# gcc conserv.c  -o s
[root@localhost Desktop]# ./s
I am Server...
Socket Created Successfully...
Binding Done...
Listening for Client...

Connected to Client...
Child process created...
Message from Client...
socket programming

Connected to Client...
Child process created...
Message from Client...
welcome
```

Fig. 8 Execution of Concurrent Echo Server

Fig. 6 shows the data read from client 1. Fig. 7 shows the data read from client 2. Fig.8 shows the data processed by concurrent server from client 1 and client 2.

VII. IMPLEMENTATION OF ECHO PROGRAM FOR CONCURRENT SERVER USING THREAD

```
#include<stdio.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<pthread.h>

void str_echo(int s)
{
        char buf[20];
        recv(s,buf,20,0);
        puts("Message from Client...");
        fputs(buf,stdout);
        send(s,buf,20,0);
}

static void *doit(void *arg)
{
        pthread_detach(pthread_self());
        str_echo((int)arg);
        close((int)arg);
        pthread_exit(0);
        return NULL;
}

int main()
{
    int ls,cs,len;

    struct sockaddr_in serv,cli;
    pid_t pid;
    pthread_t th;

    puts("I am Server...");

       //creating socket
    ls=socket(AF_INET,SOCK_STREAM,0);
    puts("Socket Created Successfully...");

       //socket address structure
    serv.sin_family=AF_INET;
    serv.sin_addr.s_addr=INADDR_ANY;
    serv.sin_port=htons(5000);

    bind(ls,(struct sockaddr*)&serv,sizeof(serv));
    puts("Binding Done...");

    listen(ls,3);
    puts("Listening for Client...");
```

 ISSN: 2321-8169
2225 – 2229

```
        for(; ;)
        {
    len=sizeof(cli);
    cs=accept(ls,(struct sockaddr*)&cli,&len);
    puts("Connected to Client...");

        //creating thread
     pthread_create(&th,NULL,&doit,(void *)cs);

        }
        return 0;

}
```

```
[root@localhost Desktop]# gcc client.c -o c
[root@localhost Desktop]# ./c
I am Client...
Socket Created Successfully...
Connected with Server...
Enter the Message...
socket programming
Message from Server...
socket programming
```

Fig. 9 Execution of Echo Client 1

```
[root@localhost Desktop]# gcc client.c -o c1
[root@localhost Desktop]# ./c1
I am Client...
Socket Created Successfully...
Connected with Server...
Enter the Message...
hai welcome
Message from Server...
hai welcome
```

Fig. 10 Execution of Echo Client 2

```
[root@localhost Desktop]# gcc thserver.c -o s -lpthread
[root@localhost Desktop]# ./s
I am Server...
Socket Created Successfully...
Binding Done...
Listening for Client...
Connected to Client...
Message from Client...
socket programming
Connected to Client...
Message from Client...
hai welcome
```

Fig. 11 Execution of Concurrent Echo Server

VIII. CONCLUSION

This paper describes about the elementary socket and thread functions needed for the implementation of concurrent server. Echo client/server program for concurrent server using fork and thread is implemented and shown along with the execution. Synchronization problem exist with threads which can be overcome by using mutex and condition variables. To expand the echo client/server into any application, change what the server does with the input it receives from its clients.

REFERENCES

[1] W. Richard Stevens, “Unix Network Programming Vol-I”, Second Edition, Pearson Education, 1998.
[2] D.E. Comer, “Internetworking with TCP/IP Vol - III”, (BSD Sockets Version), Second Edition, Pearson Education, 200 UNIT III
[3] Michael J. Donahoo and Kenneth L. Calvert, "TCP/IP Sockets in C: Practical Guide for Programmers", Second Edition, Morgan Kaufmann, 2001.
[4] Stefan Bocking,‖Socket++: A Uniform Application Programming Interface for Basic-Level Communication Services, IEEE Communication Magazine December 1996.
[5] Mattew Cook and Syed(shawon)M. Rahman, Java and C/C++ language feature in terms of Network Programming,‖ 2005.
[6] https://computing.llnl.gov/tutorials/pthreads/
[7] http://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread.h.html