**SRM INSTITUTE OF SCIENCE & TECHNOLOGY**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**



**18CSC302J - COMPUTER NETWORK**

**BATCH – 2 , SEMESTER – 5**

**Year :- 2020-2021**

|  |  |
| --- | --- |
| **NAME** | **Shushrut Kumar** |
| **Reg. No.** | **RA1811028010049** |
| **CSE-CC** | **J2** |

**B.tech-CSE-CC, Third Year (Section: J2)**

**Faculty Incharge: Vaishnavi Moorthy, Asst Prof/Dept of CSE**

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Exp. No. | Date | Exp. Name | Marks | signature |
| 1 | 05-Aug | Standard Library for Socket IO |  |  |
| 2 | 12-Aug | TCP/IP Client Server Communication |  |  |
| 3 | 20-Aug | UDP Echo Client Server Communication |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Faculty Signature

COMPUTER NETWORKING

Shushrut Kumar

Date: 5-08-2020

RA1811028010049

Lab Exercise: #1

**Aim:** Study of necessary header files with respect to socket programming and of Basic Functions of Socket Programming.

**1.man string** : The string header defines one variable type, one macro, and various functions for manipulating arrays of characters.

**2.man netdb** : This manual page is part of the POSIX Programmer's Manual. The Linux Implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

**3.man time** : The time.h header defines four variable types, two macro and various functions for manipulating date and time. **time**() returns the time as the number of seconds since the Epoch,

1970-01-01 00:00:00 +0000 (UTC).If *tloc* is non-NULL, the return value is also stored in the memory pointed to by *tloc*.

**4.man stdio** : The stdio header defines three variable types, several macros, and various functions for performing input and output.

**5.man ioctl** : The ioctl() system call manipulates the underlying device parameters of special files. In particular, many operating characteristics of character special files (e.g., terminals) may be controlled with ioctl() requests. The argument fd must be an open file descriptor.

**6.man stat** : The <sys/stat.h> header defines the structure of the data returned by the functionsfstat(), lstat(), and stat().

**7.man sys** : The **sysfs** filesystem is a pseudo-filesystem which provides an interface to kernel data structures. (More precisely, the files and directories in **sysfs** provide a view of the *kobject* structures defined internally within the kernel.) The files under **sysfs** provide information about devices, kernel modules, filesystems, and other kernel components.

**8.man ioctl** : The ioctl() system call manipulates the underlying device parameters of special files. In particular, many operating characteristics of character special files (e.g., terminals) may be controlled with ioctl() requests. The argument fd must be an open file descriptor.

**9.man errno** : The *<errno.h>* header file defines the integer variable *errno*, which is set by system calls and some library functions in the event of an error to indicate what went wrong.

**10.man pcap-filter** : The Packet Capture library provides a high level interface to packet capture systems. All packets on the network, even those destined for other hosts, are accessible through this mechanism. It also supports saving captured packets to a ``savefile'', and reading packets from a ``savefile''.

**11.man write : write()** writes up to *count* bytes from the buffer starting at *buf* to the file referred to by the file descriptor *fd*.The number of bytes written may be less than *count* if, for example, there is insufficient space on the underlying physical medium, or the

**RLIMIT\_FSIZE** resource limit is encountered.

**12.man inet : inet\_aton()** converts the Internet host address *cp* from the IPv4 numbers-and-dots notation into binary form (in network byte order) and stores it in the structure that *inp* points to. **inet\_aton**() returns nonzero if the address is valid, zero if not.

**13.man system** : The **sysfs** filesystem is a pseudo-filesystem which provides an interface to kernel data structures. (More precisely, the files and directories in **sysfs** provide a view of the *kobject* structures defined internally within the kernel.) The files under **sysfs** provide information about devices, kernel modules, filesystems, and other kernel components.

**14.man gethostname** : These system calls are used to access or to change the system hostname. More precisely, they operate on the hostname associated with the calling process's UTS namespace.

**15.man read** : read() attempts to read up to *count* bytes from file descriptor *fd* into the buffer starting at *buf*. On files that support seeking, the read operation commences at the file offset, and the file offset is incremented by the number of bytes read. If the file offset is at or past the end of file, no bytes are read, and read() returns zero.

**16.man htonl** : These functions shall convert 16-bit and 32-bit quantities between network byte order and host byte order.On some implementations, these functions are defined as macros.The uint32\_t and uint16\_t types are defined in *<inttypes.h>*.

**17.man htons** : The htonl() function converts the unsigned integer *hostlong* from host byte order to network byte order. The htons() function converts the unsigned short integer *hostshort* from host byte order to network byte order. The ntohl() function converts the unsigned integer *netlong* from network byte order to host byte order.

**18.man gethostbyname** : The gethostbyname() function returns a structure of type *hostent* for

the given host *name*. Here *name* is either a hostname or an IPv4 address in standard dot notation (as for inet\_addr(3)). If *name* is an IPv4 address, no lookup is performed and gethostbyname() simply copies *name* into the *h\_name* field and its *struct in\_addr* equivalent

into the *h\_addr\_list[0]* field of the returned *hostent* structure. If *name* doesn't end in a dot and the environment variable HOSTALIASES is set, the alias file pointed to by HOSTALIASES will first be searched for *name* (see hostname(7) for the file format). The current domain and its parents are searched unless *name* ends in a dot.

**19.man socket** : socket() creates an endpoint for communication and returns a file descriptor that refers to that endpoint. The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process.

**20.man bind** : When a socket is created with socket, it exists in a name space (address family) but has no address assigned to it. bind() assigns the address specified by *addr* to the socket referred to by the file descriptor *sockfd*. *addrlen* specifies the size, in bytes, of the address structure pointed to by *addr*. Traditionally, this operation is called “assigning a name to a socket”.

**21.man if\_config** : Ifconfig is used to configure the kernel-resident network interfaces. It is used at boot time to set up interfaces as necessary. After that, it is usually only needed when debugging or when system tuning is needed.\

**22.man accept** : The accept() system call is used with connection-based socket types

(SOCK\_STREAM, SOCK\_SEQPACKET). It extracts the first connection request on the queue of pending connections for the listening socket, *sockfd*, creates a new connected socket, and returns a new file descriptor referring to that socket. The newly created socket is not in the listening state. The original socket *sockfd* is unaffected by this call.

**23.man recv** : The recv(), recvfrom(), and recvmsg() calls are used to receive messages from a socket. They may be used to receive data on both connectionless and connection-oriented sockets. This page first describes common features of all three system calls, and then

describes the differences between the calls.

**RESULT:**

Various Functions related to socket programming were studied.

Lab Exercise: #2

**TCP/IP Client Server Communication**

**AIM:** Setting up of simple TCP/IP Client Server Communication.

**TCP SERVER :**

**Algorithm -**

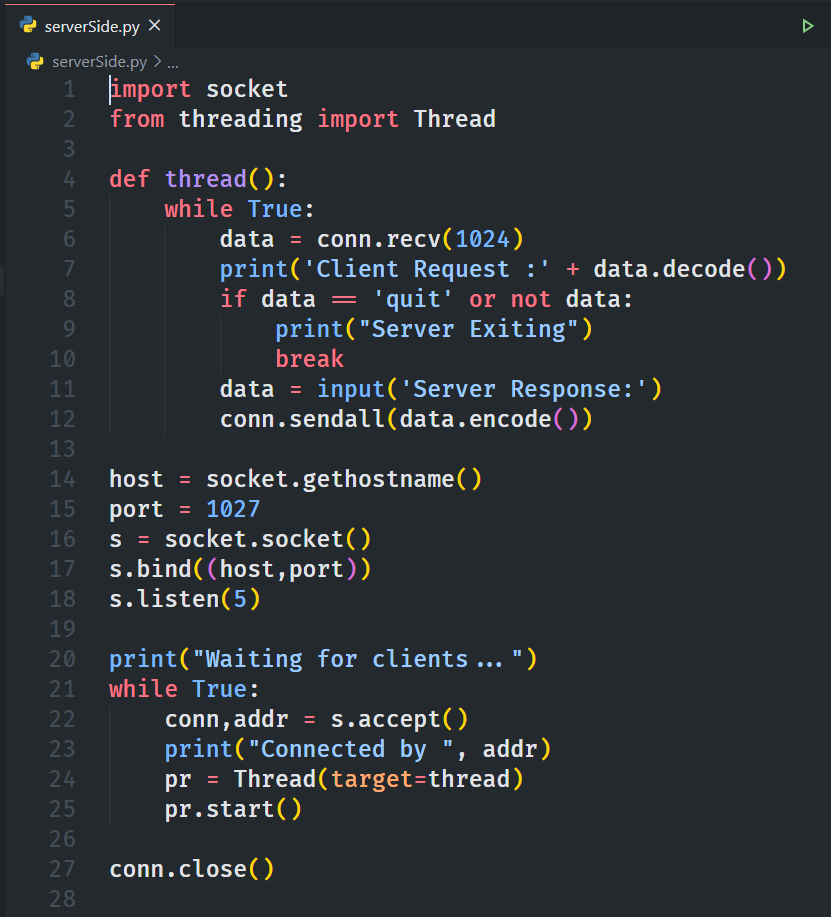
→ Create() - Create TCP socket.

→ Bind() - Bind the socket to the server address.

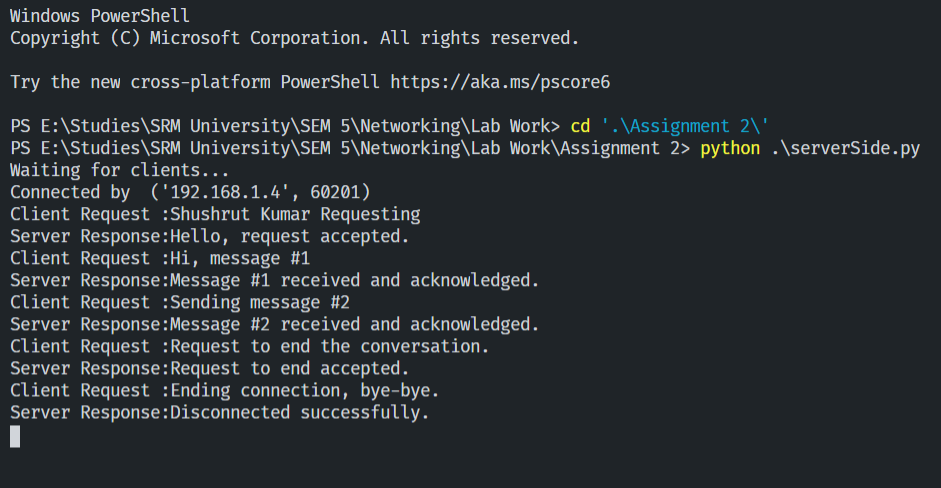
→ Listen() - put the server socket in a passive mode, where it waits for the client to approach the server to make a connection

→ Accept() - At this point, connection is established between client and server, and they are ready to transfer data.

→ Go back to Step 3.

**Code-**

**Output-**

****

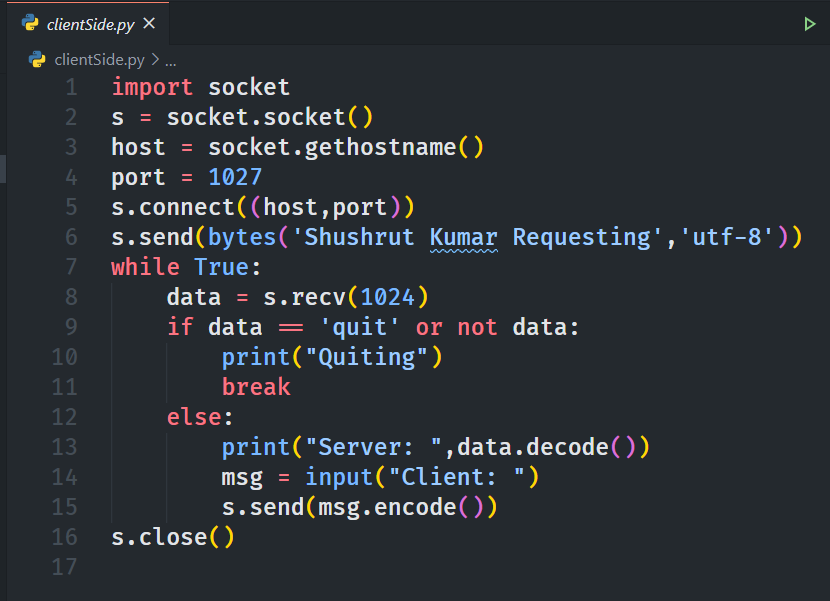
**TCP CLIENT :**

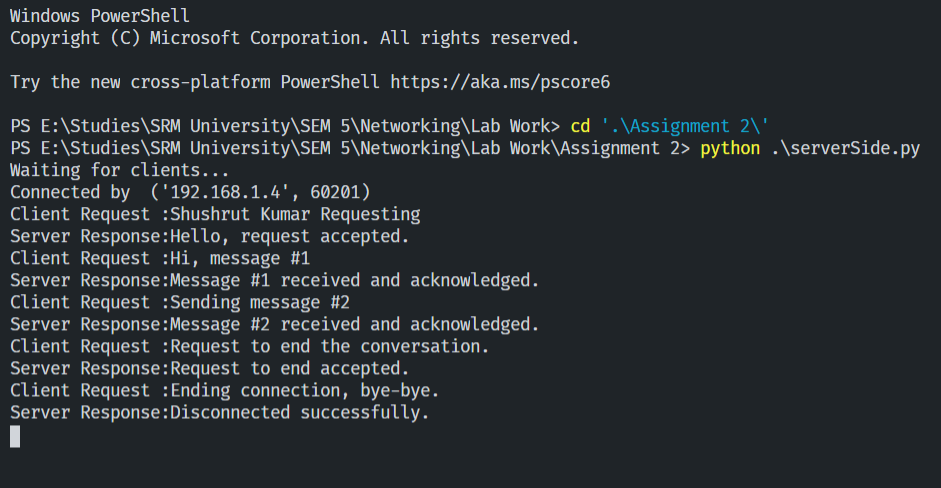
**Algorithm-**

→ Create TCP socket.

→ Connect newly created client socket to server.

**Code-**



**Output-**

**RESULT:**

A simple TCP/IP Server/Client Communication was set-up.

**Experiment - 3**

**UDP Server Client Implementation**

**AIM:**  To set up a UDP Echo Client Server Communication

**SERVER-**

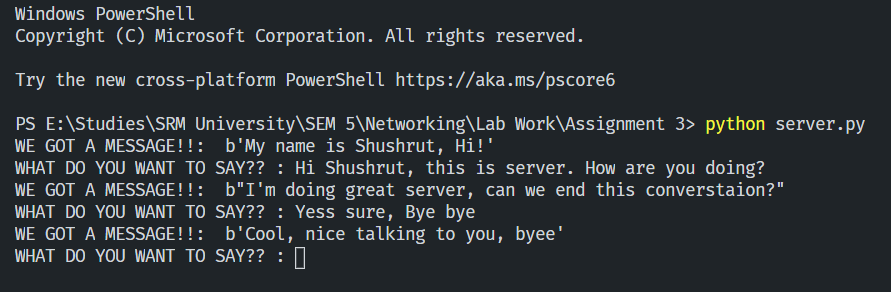
ALGORITHM:

1. Create UDP socket.
2. Bind the sockets to the server address.
3. Wait until the datagram packet arrives from the client.
4. Process the datagram packet and send a reply to the client.
5. Go back to step 3.

CODE



OUTPUT:

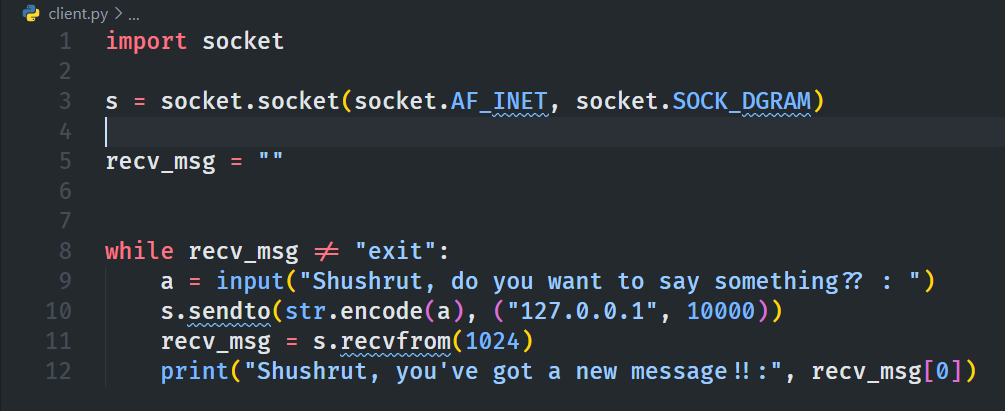


**CLIENT-**

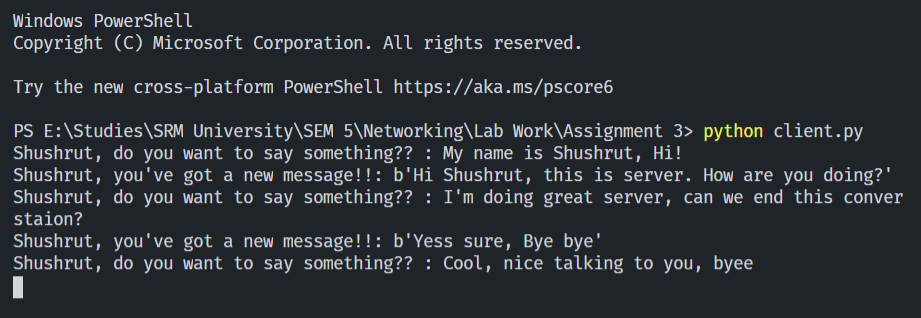
ALGORITHM:

1. Create UDP socket.
2. Send message to server.
3. Wait until response from server is received.
4. Process reply and go back to step 2, if necessary.
5. Close socket descriptor and exit.

CODE



OUTPUT:



**RESULT:**

A UDP Echo Client/Server Communication was established.

**---END--**