**Ex: 1 Distributed Components Laboratory**

**Date: 9/9/2020 POSTLAB**

**Describe the following core socket functions *(with syntax, data type, parameters and return value)* required to write a complete client and server program in C.**

**1. socket ()**

**Description:**

To perform network I/O, the first thing a process must do is, call the socket function, specifying the type of communication protocol desired and protocol family, etc.

**Syntax:**

int socket (int family, int type, int protocol);

**Parameters:**

**family** − It specifies the protocol family

**type** − It specifies the kind of socket you want

**protocol** − The argument should be set to the specific protocol or 0 to select the system's default for the given combination of family and type

**Return Type:**

This call returns a socket descriptor that you can use in later system calls or -1 on error.

**2. bind()**

**Description:**

The *bind* function assigns a local protocol address to a socket. With the Internet protocols, the protocol address is the combination of either a 32-bit IPv4 address or a 128-bit IPv6 address, along with a 16-bit TCP or UDP port number. This function is called by TCP server only.

**Syntax:**

int bind(int sockfd, struct sockaddr \*my\_addr,int addrlen);

**Parameters:**

* **sockfd** − It is a socket descriptor returned by the socket function.
* **my\_addr** − It is a pointer to struct sockaddr that contains the local IP address and port.
* **addrlen** − Set it to sizeof(struct sockaddr).

**3. listen()**

**Description:**

The *listen* function is called only by a TCP server and it performs two actions −

* The listen function converts an unconnected socket into a passive socket, indicating that the kernel should accept incoming connection requests directed to this socket.
* The second argument to this function specifies the maximum number of connections the kernel should queue for this socket.

**Syntax:**

int listen(int sockfd,int backlog);

**Parameters:**

* **sockfd** − It is a socket descriptor returned by the socket function.
* **backlog** − It is the number of allowed connections.

**Return Type:**

This call returns 0 on success, otherwise it returns -1 on error.

**4. accept()**

**Description:**

The *accept* function is called by a TCP server to return the next completed connection from the front of the completed connection queue.

**Syntax:**

int accept (int sockfd, struct sockaddr \*cliaddr, socklen\_t \*addrlen);

**Parameters:**

**sockfd** − It is a socket descriptor returned by the socket function.

**cliaddr** − It is a pointer to struct sockaddr that contains client IP address and port.

**addrlen** − Set it to sizeof(struct sockaddr).

**Return Type:**

This call returns a non-negative descriptor on success, otherwise it returns -1 on error.

**5. read()**

**Description:**

The *read* function attempts to read nbyte bytes from the file associated with the buffer, fildes, into the buffer pointed to by buf.

**Syntax:**

int read(int fildes, const void \*buf, int nbyte);

**Parameters:**

* **fildes** − It is a socket descriptor returned by the socket function.
* **buf** − It is the buffer to read the information into.
* **nbyte** − It is the number of bytes to read.

**Return Type:**

Upon successful reading, it returns the number of bytes read, and -1 when there is any error.

**6. write()**

**Description:**

The *write* function attempts to write nbyte bytes from the buffer pointed by *buf* to the file associated with the open file descriptor, *fildes*.

**Syntax:**

int write(int fildes, const void \*buf, int nbyte);

**Parameters:**

* **fildes** − It is a socket descriptor returned by the socket function.
* **buf** − It is a pointer to the data you want to send.
* **nbyte** − It is the number of bytes to be written. If nbyte is 0, write() will return 0 and have no other results if the file is a regular file; otherwise, the results are unspecified.

**Return Type:**

Upon successful completion, write() returns the number of bytes actually written to the file associated with fildes. This number is never greater than nbyte. Otherwise, -1 is returned.

**7. connect()**

**Description:**

The *connect* function is used by a TCP client to establish a connection with a TCP server.

**Syntax:**

int connect(int sockfd, struct sockaddr \*serv\_addr, int addrlen);

**Parameters:**

**sockfd** − It is a socket descriptor returned by the socket function.

**serv\_addr** − It is a pointer to struct sockaddr that contains destination IP address and port.

**addrlen** − Set it to sizeof(struct sockaddr).

**Return Type:**

This call returns 0 if it successfully connects to the server, otherwise it returns -1 on error.

**8. close()**

**Description:**

The *close* function is used to close the communication between the client and the server.

**Syntax:**

int close(int sockfd);

**Parameters:**

**sockfd** − It is a socket descriptor returned by the socket function.

**Return Type:**

This call returns 0 on success, otherwise it returns -1 on error.

**9. send()**

**Description:**

The *send* function is used to send data over stream sockets or CONNECTED datagram sockets. If you want to send data over UNCONNECTED datagram sockets, you must use sendto() function.

**Syntax:**

int send(int sockfd, const void \*msg, int len, int flags);

**Parameters:**

* **sockfd** − It is a socket descriptor returned by the socket function.
* **msg** − It is a pointer to the data you want to send.
* **len** − It is the length of the data you want to send (in bytes).
* **flags** − It is set to 0.

**Return Type:**

This call returns the number of bytes sent out, otherwise it will return -1 on error.

**10. recv ()**

**Description:**

The *recv* function is used to receive data over stream sockets or CONNECTED datagram sockets. If you want to receive data over UNCONNECTED datagram sockets you must use recvfrom().

**Syntax:**

int recv(int sockfd, void \*buf, int len, unsigned int flags);

**Parameters:**

* **sockfd** − It is a socket descriptor returned by the socket function.
* **buf** − It is the buffer to read the information into.
* **len** − It is the maximum length of the buffer.
* **flags** − It is set to 0.

**Return Type:**

This call returns the number of bytes read into the buffer, otherwise it will return -1 on error.

**11. recvfrom ()**

**Description:**

The *recvfrom* function is used to receive data from UNCONNECTED datagram sockets.

**Syntax:**

int recvfrom(int sockfd, void \*buf, int len, unsigned int flags struct sockaddr \*from, int \*fromlen;

**Parameters:**

* **sockfd** − It is a socket descriptor returned by the socket function.
* **buf** − It is the buffer to read the information into.
* **len** − It is the maximum length of the buffer.
* **flags** − It is set to 0.
* **from** − It is a pointer to struct sockaddr for the host where data has to be read.
* **fromlen** − It is set it to sizeof(struct sockaddr).

**Return Type:**

This call returns the number of bytes read into the buffer, otherwise it returns -1 on error.

**12. sendto()**

**Description:**

The *sendto* function is used to send data over UNCONNECTED datagram sockets.

**Syntax:**

int sendto(int sockfd, const void \*msg, int len, unsigned int flags, const struct sockaddr \*to, int tolen);

**Parameters:**

* **sockfd** − It is a socket descriptor returned by the socket function.
* **msg** − It is a pointer to the data you want to send.
* **len** − It is the length of the data you want to send (in bytes).
* **flags** − It is set to 0.
* **to** − It is a pointer to struct sockaddr for the host where data has to be sent.
* **tolen** − It is set it to sizeof(struct sockaddr).

**Return Type:**

This call returns the number of bytes sent, otherwise it returns -1 on error.

**13. shutdown:**

**Description:**

The *shutdown* function is used to gracefully close the communication between the client and the server. This function gives more control in comparison to the *close* function.

**Syntax:**

int shutdown(int sockfd, int how;

**Parameters:**

* **sockfd** − It is a socket descriptor returned by the socket function.
* **how** − Put one of the numbers −
  + **0** − indicates that receiving is not allowed,
  + **1** − indicates that sending is not allowed, and
  + **2** − indicates that both sending and receiving are not allowed. When *how* is set to 2, it's the same thing as close()

**Return Type:**

This call returns 0 on success, otherwise it returns -1 on error.

**14. select()**

**Description:**

The *select* function indicates which of the specified file descriptors is ready for reading, ready for writing, or has an error condition pending.

**Syntax:**

int select(int nfds, fd\_set \*readfds, fd\_set \*writefds, fd\_set \*errorfds, struct timeval \*timeout;

**Parameters:**

* **nfds** − It specifies the range of file descriptors to be tested. The select() function tests file descriptors in the range of 0 to nfds-1
* **readfds** − It points to an object of type *fd\_set* that on input, specifies the file descriptors to be checked for being ready to read, and on output, indicates which file descriptors are ready to read. It can be NULL to indicate an empty set.
* **writefds** − It points to an object of type *fd\_set* that on input, specifies the file descriptors to be checked for being ready to write, and on output, indicates which file descriptors are ready to write. It can be NULL to indicate an empty set.
* **exceptfds** − It points to an object of type *fd\_set* that on input, specifies the file descriptors to be checked for error conditions pending, and on output indicates, which file descriptors have error conditions pending. It can be NULL to indicate an empty set.
* **timeout** − It points to a timeval struct that specifies how long the select call should poll the descriptors for an available I/O operation. If the timeout value is 0, then select will return immediately. If the timeout argument is NULL, then select will block until at least one file/socket handle is ready for an available I/O operation. Otherwise *select* will return after the amount of time in the timeout has elapsed OR when at least one file/socket descriptor is ready for an I/O operation.

**Return Type:**

The return value from select is the number of handles specified in the file descriptor sets that are ready for I/O. If the time limit specified by the timeout field is reached, select return 0.

**Result:**

The above functions have been explored and studied successfully.