Ex. No. 1B **<u>Study of System Calls</u>** **Karthik D**

21 July 2021 **<u>UCS1511 - Networks Lab</u>** **195001047**

<u>Aim</u>
To study basic system calls associated with networking in C language and understand their usage variations

1. socket()

Description
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.

Header files
<sys/socket.h>

Syntax
***int** socket (**int** domain, **int** type, **int** protocol);*

- *<u>domain</u>*
  Specifies the protocol domain to be used for communications using this socket.Represented as an integer constant. Eg: AF_INET (for IPv4), AF_UNIX (for local communication)
- *<u>type</u>*
  Specifies the semantics that will be used for communications using this socket.
  Eg: SOCK_DGRAM (UDP), SOCK_STREAM (TCP)
- *<u>protocol</u>*
  Specifies the protocol to be used with the socket. If there is only a single protocol suitable for the specified domain and type, the default can be selected by specifying 0

Return value
- *<u>Success</u>*: A file descriptor for the new socket is returned.
- *<u>Error</u>*: -1 is returned, *errno* is set to indicate the error.

2. bind()

Description
Assigns the address specified in arguments to the socket, whose file descriptor is specified in the arguments. This process is commonly referred to as "assigning/binding a name to a socket"

Header files
<sys/socket.h>

Syntax
int *bind* ***(*** ***int*** *sockfd*, ***const struct sockaddr*** **addr*, ***socklen_t*** *addrlen);*

- *sockfd*
  Specifies the file descriptor of the socket which has to be bound to the given address
- *addr*
  Specifies the address to which the given socket must be bound. The address must be a pointer to the *sockaddr* structure
- *addrlen*
  Specifies the size, in bytes, of the address structure pointer - *addr*.

Return value
- *Success*: 0 is returned
- *Error*: -1 is returned, *errno* is set to indicate the error.

Structures used
- *sockaddr* structure is the data-type used to represent the address for the socket

```
struct sockaddr {
        sa_family_t sa_family;
        char sa_data[14];
}
```

3. listen()

Description
Used to start listening for connections on a socket. The socket is marked as a passive socket which will be used to accept incoming connection requests using accept() system call

Header files
<sys/socket.h>

Syntax
int *listen* ***(*** ***int*** *sockfd*, ***int*** *backlog* ***);***

- *sockfd*
  Specifies the file descriptor of the socket which has to be used to listen to connection requests
- *backlog*
  Specifies the maximum length to which the queue of pending connections for the specified socket can grow. Protocol specific measures are used when the requests crosses queue length

Return value
- *Success*: 0 is returned
- *Error*: -1 is returned, *errno* is set to indicate the error

4. **connect()**

   Description
   Used to initiate a connection on a socket. It connects a the socket specified in the arguments to the address given

   Header files
   <sys/socket.h>

   Syntax
   ***int** connect **(int** sockfd**, const struct sockaddr** *addr, **socklen_t** addrlen);*
   - <u>*sockfd*</u>
     Specifies the file descriptor of the socket which has to be used to initiate a connection.
   - <u>*addr*</u>
     Specifies the address to which the given socket must be connected to. The address must be a pointer to the *sockaddr* structure
   - <u>*addrlen*</u>
     Specifies the size, in bytes, of the address structure pointer - *addr*.

   Return value
   - <u>*Success*</u>: 0 is returned
   - <u>*Error*</u>: -1 is returned, *errno* is set to indicate the error

   Structures used
   - <u>*sockaddr*</u> structure is the data-type used to represent the address for the connection
     ```
     struct sockaddr {
             sa_family_t sa_family;
             char sa_data[14];
     };
     ```

5. **close()**

   Description
   Used to close a file descriptor such that it no longer refers to any file. Such a file descriptor can be re-used. This is used in common for file operations as well as sockets

   Header files
   <unistd.h>

Syntax
***int** close **(int** fd);*
- *<u>fd</u>*
  Specifies the file descriptor which needs to be freed. Can correspond to a socket or a file.

Return value
- *<u>Success</u>*: 0 is returned
- *<u>Error</u>*: -1 is returned, *errno* is set to indicate the error

6. **bzero()**

Description
Used to erase data in the *n* bytes of memory starting at the location specified by the pointer argument. Typically used to “zero” a byte string during communication between sockets

Header files
<strings.h>

Syntax
***void** bzero **(void** ****s*, **size_t** *n);*
- *<u>s</u>*
  A pointer to the memory location starting point, from where on the data has to be erased
- *<u>n</u>*
  The number of bytes, starting at the location pointed to by *s*, that needs to be erased and set to zero i.e *\0 (null)*.

Return value
No return value

7. **htons(), htonl(), ntohs(), ntohs()**

Description
Used to convert the unsigned integer specified as argument from host byte order to the network byte order. Different systems may use different byte-ordering representations (big-endian, little-endian). This call is used to convert the bytes suitable.
Function names like *<u>hton[s/l]</u>* convert from host byte order to the network byte order.
Function names like *<u>ntoh[s/l]</u>* convert from network byte order to the host byte order.
The last letter of the function name denotes the data size where *s* denotes *short* and *l* denotes *long*

Header files
<arpa/inet.h>

Syntax

uint16_t *htons (* ***uint16_t*** *hostshort);*
uint32_t *htonl (* ***uint32_t*** *hostlong);*
uint16_t *ntohs (* ***uint16_t*** *netshort);*
uint32_t *ntohl* ***(*** ***uint32_t*** *netlong);*

- *hostshort*
  Specifies the *unsigned short integer* in host byte-order to be converted
- *hostlong*
  Specifies the *unsigned long integer* in host byte-order to be converted
- *netshort*
  Specifies the *unsigned short integer* in network byte-order to be converted
- *netlong*
  Specifies the *unsigned long integer* in network byte-order to be converted

Return value
Returns the corresponding converted byte-order value

8. read()

Description
Used to read data from a file specified by its descriptor. Attempts to read upto specified number of bytes from the supplied file descriptor and stores it in the buffer pointer location specified.

Header files
<unistd.h>

Syntax
ssize_t *read (* ***int*** *fd,* ***void*** **buf,* ***size_t*** *count);*
- *fd*
  Specifies the file descriptor of the file/socket from where the bytes need to be read
- *buf*
  Specifies a pointer to the location from where on the read bytes have to be stored
- *count*
  Specifies the number of bytes to be read from the file/socket, starting from the last read location

Return value

- <u>*Success*</u>: The number of bytes read is returned. 0 indicates that the end of file is reached
- <u>*Error*</u>: -1 is returned, *errno* is set to indicate the error

9. write()

Description

Used to write data to a file specified by its file descriptor. Attempts to write upto the specified number of bytes, reading from the buffer pointer location specified.

Header files

<unistd.h>

Syntax

ssize_t *write (* ***int*** *fd****,*** ***const void*** **buf****,*** ***size_t*** *count);*

- <u>*fd*</u>
  Specifies the file descriptor of the file/socket to which the bytes need to be written
- <u>*buf*</u>
  Specifies a pointer to the location from where bytes to be written have to be read
- <u>*count*</u>
  Specifies the number of bytes to be read from the buffer and written into the file/socket

Return value

- <u>*Success*</u>: The number of bytes written is returned
- <u>*Error*</u>: -1 is returned, *errno* is set to indicate the error

10. send()

Description

Used to send/transmit a message on a socket. This call can be used only when the socket is in a *connected* state.

Header files

<sys/socket.h>

Syntax

ssize_t *send (* ***int*** *sockfd,* ***const void*** **buf,* ***size_t*** *len,* ***int*** *flags);*

- <u>*sockfd*</u>
  Specifies the file descriptor of the socket on which the message has to be transmitted

- *buf*
  Specifies a pointer to the location from where bytes to be sent have to be read
- *len*
  Specifies the number of bytes to be read from the buffer and transmitted over the socket
- *flags*
  Specified as a bitwise-OR of flags defined in the header <sys/socket.h>.
  Eg: MSG_EOR (Terminate record), MSG_OOB Send out-of-band data on socket)

Return value
- *Success*: The number of bytes transmitted is returned
- *Error*: -1 is returned, *errno* is set to indicate the error

11. recv()

Description
Used to receive a message on a socket. This call can be used on both connectionless and connection oriented sockets

Header files
<sys/socket.h>

Syntax
***ssize_t** recv (**int** sockfd, **const void** *buf, **size_t** len, **int** flags);*
- *sockfd*
  Specifies the file descriptor of the socket on which the message has to be received
- *buf*
  Specifies a pointer to the location to which bytes that are received have to be written
- *len*
  Specifies the number of bytes to be received over the socket and written to the *buf*
- *flags*
  Specified as a bitwise-OR of flags defined in the header <sys/socket.h>.
  Eg: MSG_PEEK (Receive data without removing read bytes from message the queue), MSG_OOB (request out-of-band data on socket)

Return value
- *Success*: The number of bytes received is returned
- *Error*: -1 is returned, *errno* is set to indicate the error

12. sendto()

Description
Used to send a message on a socket. This call can be used only when the socket is in a *connected* state. The socket to which the message has to be transmitted is also specified as an argument

Header files
<sys/socket.h>

Syntax
ssize_t** sendto **(int** sockfd**, void *restrict** buf**, size_t** len**, int** flags**, struct sockaddr *restrict** dest_addr**, socklen_t restrict** addrlen **);

- *sockfd*
  Specifies the file descriptor of the socket over which the message has to sent
- *buf*
  Specifies a pointer to the location to from where bytes to be sent are read
- *len*
  Specifies the number of bytes to be read from *buf* and transmitted over the socket
- *flags*
  Specified as a bitwise-OR of flags defined in the header <sys/socket.h>.
  Eg: MSG_EOR (Terminate record), MSG_OOB Send out-of-band data on socket)
- *dest_addr*
  Specifies the address of the socket for which the message being transmitted is intended. This is called the target socket.
- *addrlen*
  Specifies the size of the destination address structure that is being passed in *dest_addr*

Return value
- *Success*: The number of bytes received is returned
- *Error*: -1 is returned, *errno* is set to indicate the error

Structures used
- *sockaddr* structure is the data-type used to represent the address of the destination socket

```
struct sockaddr {
        sa_family_t sa_family;
        char sa_data[14];
}
```

13. recvfrom()

Description
Used to receive a message on a socket. This call can be used on both connectionless and connection oriented sockets. The socket from which the message is received is also updated in the supplied pointer location

Header files
<sys/socket.h>

Syntax
ssize_t *recvfrom* ***(int*** *sockfd****, void *restrict*** *buf****, size_t*** *len****, int*** *flags****, struct sockaddr *restrict*** *src_addr****, socklen_t *restrict*** *addrlen);*

- *sockfd*
  Specifies the file descriptor of the socket on which the message has to be received
- *buf*
  Specifies a pointer to the location to which bytes that are received have to be written
- *len*
  Specifies the number of bytes to be received over the socket and written to the *buf*
- *flags*
  Specified as a bitwise-OR of flags defined in the header <sys/socket.h>.
  Eg: MSG_PEEK (Receive data without removing read bytes from message the queue), MSG_OOB (request out-of-band data on socket)
- *src_addr*
  Specifies the memory location to which the message sender's address must be written. If this is NULL, the address is not written
- *addrlen*
  Specifies the memory location to which the size of the structure representing the sender's address must be written. It must be initialized to the buffer size used for *src_addr*. If a size smaller than the actual sender address size is specified, the written address in *src_addr* will be truncated

Return value
- *Success*: The number of bytes received is returned
- *Error*: -1 is returned, *errno* is set to indicate the error

Structures used
- *sockaddr* structure is the data-type used to represent the address of the source socket
  struct **sockaddr** {
  sa_family_t sa_family;

```
        char sa_data[14];
    }
```

14. select()

Description
Used to allow a program to monitor multiple file descriptors, waiting until one or more of them become *ready* for a particular I/O operation. It blocks execution until this scenario occurs, an interrupt is raised or the timeout interval elapses.

Header files
<sys/select.h>

Syntax
```
int select ( int nfds, fd_set *restrict readfds,
             fd_set *restrict writefds, fd_set *restrict exceptfds,
             struct timeval *restrict timeout );
```

- *nfds*
  Specified as an integer value that is one more than the largest number of file descriptors listed among *readfds, writefds* and *exceptfds*. Used to iterate through these structures
- *readfds*
  Specifies a set of file descriptors, as a pointer to the *fd_set* structure, that have to be monitored to check if they are ready for reading. When the call returns, only the read-ready file descriptors are contained in *readfds*
- *writefds*
  Specifies a set of file descriptors, as a pointer to the *fd_set* structure, that have to be monitored to check if they are ready for writing. When the call returns, only the write-ready file descriptors are contained in *writefds*
- *exceptfds*
  Specifies a set of file descriptors, as a pointer to the *fd_set* structure, that have to be monitored to check if they are in one of the exceptional conditions. When the call returns, only the file descriptors for which the exceptional condition is reached are contained in *exceptfds*
- *timeout*
  Specifies the time interval as a *timeval* structure instance pointer, denoting the time duration for which the *select()* call should block and wait for a file descriptor to reach its ready state

Return value
- *Success*: The number of file descriptors contained in the three descriptor sets.
- *Timeout*: 0 is returned
- *Error*: -1 is returned, *errno* is set to indicate the error

Structures used

- *timeval* structure is the data-type used to represent the timeout duration in the *select()* call

  struct **timeval** {
      time_t tv_sec;
      suseconds_t tv_usec;
  };

- *fd_set* structure is the data-type used to represent a set of file descriptors

  struct **fd_set** {
      u_int fd_count;
      SOCKET fd_array[FD_SETSIZE];
  };

15. setsockopt()

Description

Used to set a specific option at the specified protocol level to the supplied value for the given socket specified using its file descriptor

Header files

<sys/socket.h>

Syntax

int *setsockopt* ***(*** ***int*** *socket,* ***int*** *level,* ***int*** *option_name,*
    const void *******option_value,* ***socklen_t*** *option_len);*

- *socket*

  Specifies the file descriptor of the socket for which the option value has to be set
- *level*

  Specifies the protocol level at which the option resides.
  Eg: SOL_SOCKET (for socket-level)
- *option_name*

  Specifies a single option's name whose value has to be set for the specified socket
- *option_value*

  Specifies the value to which the given socket's specified option must be set. Specified as a pointer to a suitable data-type. For boolean options, 0 is disabled and 1 is enabled
- *option_len*

  Specifies the length of entries in the pointer location of *option_value*

Return value

- *Success*: 0 is returned
- *Error*: -1 is returned, *errno* is set to indicate the error

16. fcntl()

Description
Used to manipulate a given file descriptor based on the specified operation. Depending on the command specified, a third argument may or may not be required

Header files
<fcntl.h>

Syntax
int** fcntl **(** **int** fd**,** **int** cmd**,** ... **);
- *fd*
  Specifies the file descriptor of the socket/file which has to be manipulated
- *cmd*
  Specifies the command using its name as defined in the header file <fcntl.h>
  Eg: F_DUPFD (to duplicate a file descriptor), F_GETFD (to obtain the file's flags)
- <optional-third-argument>
  Specifies the value for the third argument if the specified command requires it

Return value
- Depends on the operation specified
  Eg: F_DUPFD (returns new file descriptor), F_GETFD (returns value of file's flags)
- *Error*: -1 is returned, *errno* is set to indicate the error

17. getpeername()

Description
Used to obtain the address of the peer connected to the given socket and stores into the buffer pointed to by the supplied pointer

Header files
<sys/socket.h>

Syntax
int** getpeername **(** **int** sockfd**,** **struct sockaddr *restrict** addr**,** **socklen_t *restrict** addrlen **);
- *sockfd*
  Specifies the file descriptor of the socket whose peer has to be found
- *addr*
  Specifies a pointer to a buffer where the address of the peer will be written
- addrlen

Specifies the amount of space covered by the memory location pointed to by *addr*. If this is less than the found peer address, the peer address is truncated. The size of the returned peer address structure is written into this location when the function returns

Return value

- *Success:* 0 is returned
- *Error*: -1 is returned, *errno* is set to indicate the error

Structures used

- *sockaddr* structure is the data-type used to represent the peer address found

  struct **sockaddr** {

  sa_family_t sa_family;

  char sa_data[14];

  }

18. gethostname()

Description

Used to obtain the hostname of the system associated with the calling process' UTS namespace

Header files

<unistd.h>

Syntax

***int** gethostname **(char** *name**, size_t** len);*

- *name*

  Specifies a character buffer location into which the hostname of the system will be written for returning
- *len*

  Specifies the length of the memory region covered by the character buffer - *name*. If the hostname is larger than this value, it is truncated to suit the size

Return value

- *Success:* 0 is returned
- *Error*: -1 is returned, *errno* is set to indicate the error

19. gethostbyname()

Description
Used to return a structure of *hostent* type for the given hostname, containing information including its IP address

Header files
<unistd.h>

Syntax
***struct hostent** *gethostbyname (**const char** *name);*

- *name*
  Specifies the name of the host as a character pointer for which the host details need to be obtained

Return value

- *Success:* A *hostent* structure containing the host information is returned
- *Error*: NULL is returned, *h_errno* is set to indicate the error

Structures used

- *hostent* structure is the data-type used to represent the host information returned

```
struct hostent {
        char  *h_name;          /* official name of host */
        char **h_aliases;       /* alias list */
        int    h_addrtype;      /* host address type */
        int    h_length;        /* length of address */
        char **h_addr_list;     /* list of addresses */
}
```

20. gethostbyaddr()

Description
Used to return a structure of *hostent* type for the given host address, containing information including its IP address

Header files
<unistd.h>

Syntax
***struct hostent** *gethostbyaddr (**const char** *addr, **int** len, **int** type);*

- *addr*
  Specifies the address of the host whose information needs to be found. The nature of this argument should concur with the *type* argument passed
- *len*
  Specifies the structure size of the specified address - *addr*.
- *type*
  Specifies the type of address passed in *addr*, so that it can be read accordingly. Valid types are AF_INET (for IPv4) and AF_INET6 (for IPv6)

Return value

- *Success:* A *hostent* structure containing the host information is returned
- *Error*: NULL is returned, *h_errno* is set to indicate the error

Structures used

- *hostent* structure is the data-type used to represent the host information returned
```
struct hostent {
        char  *h_name;          /* official name of host */
        char **h_aliases;       /* alias list */
        int    h_addrtype;      /* host address type */
        int    h_length;        /* length of address */
        char **h_addr_list;     /* list of addresses */
}
```
- *in_addr* structure is the data-type used to represent an IPv4 host address
```
struct in_addr {
   in_addr_t s_addr;
};
```
- *in6_addr* structure is the data-type used to represent an IPv6 host address
```
struct in6_addr {
   unsigned char s6_addr[16];
};
```

Result

Studied some basic system calls associated with networking in C language and understood their usage variations