SSN COLLEGE OF ENGINEERING, KALAVAKKAM

(An Autonomous Institution, Affiliated to Anna University, Chennai)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS1511 – COMPUTER NETWORKS LAB

Lab Exercise 1 – System Calls

Aim:

Learn and understand the use of system calls.

Study:

1. **socket():** create an endpoint for communication
   a. 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.
   b. Header Files:
      i. #include <sys/types.h>
      ii. #include <sys/socket.h>
   c. Syntax:
      int socket(int domain, int type, int protocol);
   d. Params:
      i. **domain**: argument specifies a communication domain; this selects the protocol family which will be used for communication.
      ii. **type:** argument serves a second purpose: in addition to specifying a socket type, it may include the bitwise OR of any of the following values, to modify the behavior of socket().
      iii. **protocol**: specifies a particular protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type within a given protocol family, in which case protocol can be specified as 0. However, it is possible that many protocols may exist, in which case a particular protocol must be specified in this manner.
   e. Return Value: On success, a file descriptor for the new socket is returned. On error, -1 is returned, and errno is set to indicate the error.
2. **bind():** bind a name to a socket
   a. Description: assign a local socket address address to a socket identified by descriptor socket that has no local socket address assigned. Sockets

created with the socket() function are initially unnamed; they are identified only by their address family.

b. Header Files:
    i. #include <sys/types.h>
    ii. #include <sys/socket.h>
c. Syntax:
int bind(int socket, const struct sockaddr *address, socklen_t address_len);
d. Params:
    i. **socket:** Specifies the file descriptor of the socket to be bound
    ii. **address:** Points to a sockaddr structure containing the address be bound to the socket. The length and format of the address depend on the address family of the socket.
    iii. **address_len:** Specifies the length of the sockaddr structure pointed to by the address argument
e. Return Value: Upon successful completion, bind() shall return 0; otherwise, -1 shall be returned and errno set to indicate the error.
f. Structure used:

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

3. listen() - listen for connections on a socket

a. Description: marks the socket referred to by sockfd as a passive socket, that is, as a socket that will be used to accept incoming connection requests using accept(2).
b. Header Files:
    i. #include <sys/types.h>
    ii. #include <sys/socket.h>
c. Syntax:
int listen(int sockfd, int backlog);
d. Params:
    i. **sockfd**: argument is a file descriptor that refers to a socket of type SOCK_STREAM or SOCK_SEQPACKET.
    ii. **backlog:** argument defines the maximum length to which the queue of pending connections for sockfd may grow.
e. Return Value: On success, zero is returned. On error, -1 is returned, and errno is set to indicate the error.

4. **connect():** initiate a connection on a socket
   a. Description: The connect() function shall attempt to make a connection on a connection-mode socket or to set or reset the peer address of a connectionless-mode socket
   b. Header Files:
      i. #include <sys/types.h>
      ii. #include <sys/socket.h>
   c. Syntax:
      int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
   d. Params:
      i. **socket**: Specifies the file descriptor associated with the socket.
      ii. **address**: Points to a sockaddr structure containing the peer address. The length and format of the address depend on the address family of the socket.
      iii. **address_len**: Specifies the length of the sockaddr structure pointed to by the address argument.
   e. Return Value: Upon successful completion, connect() shall return 0; otherwise, -1 shall be returned and errno set to indicate the error.
   f. Structure used:

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

5. **accept():** accept a connection on a socket
   a. Description: The accept() function shall extract the first connection on the queue of pending connections, create a new socket with the same socket type protocol and address family as the specified socket, and allocate a new file descriptor for that socket.
   b. Header Files:
      i. #include <sys/types.h>
      ii. #include <sys/socket.h>
   c. Syntax:
      int accept(int socket, struct sockaddr *restrict address, socklen_t *restrict address_len);
   d. Params:
      i. **socket:** Specifies a socket that was created with socket(), has been bound to an address with bind(), and has issued a successful call to listen().

ii. **address:** either a null pointer, or a pointer to a sockaddr structure where the address of the connecting socket shall be returned.
iii. **address_len:** Points to a socklen_t structure which on input specifies the length of the supplied sockaddr structure, and on output specifies the length of the stored address.

e. Return Value**:** Upon successful completion, accept() shall return the non-negative file descriptor of the accepted socket. Otherwise, -1 shall be returned and errno set to indicate the error.
f. Structure used:

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

6. close(): close a file descriptor

a. Description: closes a file descriptor, so that it no longer refers to any file and may be reused. Any record locks (see fcntl(2)) held on the file it was associated with, and owned by the process, are removed (regardless of the file descriptor that was used to obtain the lock).
b. Header Files:
    i. #include <unistd.h>
c. Syntax:
int close(int fd);
d. Params:
    i. **fd:** is the last file descriptor referring to the underlying open file description (see open(2)), the resources associated with the open file description are freed; if the descriptor was the last reference to a file which has been removed using unlink(2) the file is deleted.
e. Return Value: returns zero on success. On error, -1 is returned, and errno is set appropriately.

7. bzero(): zero a byte string

a. Description: erases the data in the n bytes of the memory starting at the location pointed to by s, by writing zeros (bytes containing '\0') to that area.
b. Header Files:
    i. #include <strings.h>
c. Syntax:
void bzero(void *s, size_t n);
d. Params:

i. s: starting location of the memory to be erased
ii. n: number of bytes of the memory to be erased

e. Return Value: None

8. htons, htonl, ntohs, ntohl(): convert values between host and network byte order

a. Description:
   i. **htonl():** function converts the unsigned integer hostlong from host byte order to network byte order.
   ii. **htons():** function converts the unsigned short integer hostshort from host byte order to network byte order.
   iii. **ntohl():** function converts the unsigned integer netlong from network byte order to host byte order.
   iv. **ntohs():** function converts the unsigned short integer netshort from network byte order to host byte order.

b. Header Files:
   i. #include <arpa/inet.h>

c. Syntax:
   i. uint32_t htonl(uint32_t hostlong);
   ii. uint16_t htons(uint16_t hostshort);
   iii. uint32_t ntohl(uint32_t netlong);
   iv. uint16_t ntohs(uint16_t netshort);

d. Params:
   i. hostlong: unsigned integer
   ii. hostshort: unsigned short integer
   iii. netlong: unsigned integer
   iv. netshort: unsigned short integer

e. Return Value: The htonl() and htons() functions shall return the argument value converted from host to network byte order. The ntohl() and ntohs() functions shall return the argument value converted from network to host byte order.

9. read(): read from a file descriptor

a. Description: attempts to read up to count bytes from file descriptor fd into the buffer starting at buf.

b. Header Files:
   i. #include <unistd.h>

c. Syntax:
   ssize_t read(int fd, void *buf, size_t count);

d. Params:
   i. **fd:** file descriptor
   ii. **buf:** pointer to buffer

iii. **count:** count of bytes to read

e. Return Value: On success, the number of bytes read is returned (zero indicates end of file), and the file position is advanced by this number. On error, -1 is returned, and errno is set appropriately.

10.write(): write to a file descriptor

a. Description: writes up to count bytes from the buffer pointed buf to the file referred to by the file descriptor fd.
b. Header Files:
    i. #include <unistd.h>
c. Syntax:
ssize_t write(int fd, const void *buf, size_t count);
d. Params:
    i. **fd:** file descriptor
    ii. **buf:** pointer to buffer
    iii. **count:** count of bytes to write
e. Return Value: On success, the number of bytes written is returned (zero indicates nothing was written). On error, -1 is returned, and errno is set appropriately.

11.send(): send a message on a socket

a. Description: function shall initiate transmission of a message from the specified socket to its peer. The send() function shall send a message only when the socket is connected (including when the peer of a connectionless socket has been set via connect()).
b. Header Files:
    i. #include <sys/types.h>
    ii. #include <sys/socket.h>
c. Syntax:
ssize_t send(int socket, const void *buffer, size_t length, int flags);
d. Params:
    i. **socket:** Specifies the socket file descriptor.
    ii. **buffer:** Points to the buffer containing the message to send.
    iii. **length:** Specifies the length of the message in bytes.
    iv. **flags:** Specifies the type of message transmission. Values of argument are formed by logically OR'ing MSG_EOR or MSG_OOB or MSG_NOSIGNAL.
e. Return Value: Upon successful completion, send() shall return the number of bytes sent. Otherwise, -1 shall be returned and errno set to indicate the error.

12.recv(): receive a message from a connected socket

a. Description: function shall receive a message from a connection-mode or connectionless-mode socket. It is normally used with connected sockets because it does not permit the application to retrieve the source address of received data.
b. Header Files:
    i. #include <sys/types.h>
    ii. #include <sys/socket.h>
c. Syntax:
   ssize_t recv(int socket, const void *buffer, size_t length, int flags);
d. Params:
    i. **socket:** Specifies the socket file descriptor.
    ii. **buffer:** Points to a buffer where the message should be stored.
    iii. **length:** Specifies the length in bytes of the buffer pointed to by the buffer argument.
    iv. **flags:** Specifies the type of message transmission. Values of argument are formed by logically OR'ing MSG_PEEK or MSG_OOB or MSG_WAITALL.
e. Return Value: Upon successful completion, recv() shall return the length of the message in bytes. If no messages are available to be received and the peer has performed an orderly shutdown, recv() shall return 0. Otherwise, -1 shall be returned and errno set to indicate the error.

13. sendto(): send a message on a socket

a. Description: function shall send a message through a connection-mode or connectionless-mode socket. If the socket is connectionless-mode, the message shall be sent to the address specified by dest_addr. If the socket is connection-mode, dest_addr shall be ignored.
b. Header Files:
    i. #include <sys/types.h>
    ii. #include <sys/socket.h>
c. Syntax:
   ssize_t sendto(int socket, const void *message, size_t length, int flags, const struct sockaddr *dest_addr, socklen_t dest_len);
d. Params:
    i. **socket:** Specifies the socket file descriptor.
    ii. **message:** Points to a buffer containing the message to be sent.
    iii. **length:** Specifies the size of the message in bytes.
    iv. **flags:** Specifies the type of message transmission. Values of this argument are formed by logically OR'ing MSG_EOR or MSG_OOB

v. **dest_addr:** Points to a sockaddr structure containing the destination address. The length and format of the address depend on the address family of the socket.
vi. **dest_len:** Specifies the length of the sockaddr structure pointed to by the dest_addr argument.

e. Return Value: Upon successful completion, sendto() shall return the number of bytes sent. Otherwise, -1 shall be returned and errno set to indicate the error.
f. Structure used:

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

14.recvfrom(): receive a message from a socket

a. Description: function shall receive a message from a connection-mode or connectionless-mode socket. It is normally used with connectionless-mode sockets because it permits the application to retrieve the source address of received data.
b. Header Files:
    i. #include <sys/types.h>
    ii. #include <sys/socket.h>
c. Syntax:
ssize_t recvfrom(int socket, void *restrict buffer, size_t length, int flags, struct sockaddr *restrict address, socklen_t *restrict address_len);
d. Params:
    i. **socket:** Specifies the socket file descriptor.
    ii. **buffer:** Points to the buffer where the message should be stored.
    iii. **length:** Specifies the length in bytes of the buffer pointed to by the buffer argument.
    iv. **flags:** Specifies the type of message reception. Values of this argument are formed by logically OR'ing MSG_PEEK or MSG_OOB or MSG_WAITALL.
    v. **address:** A null pointer, or points to a sockaddr structure in which the sending address is to be stored. The length and format of the address depend on the address family of the socket.
    vi. **address_len:** Specifies the length of the sockaddr structure pointed to by the address argument.
e. Return Value: Upon successful completion, recvfrom() shall return the length of the message in bytes. If no messages are available to be received and the peer has performed an orderly shutdown, recvfrom() shall return

0. Otherwise, the function shall return -1 and set errno to indicate the error.

f. Structure used:

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

15.select(): synchronous I/O multiplexing

a. Description: allow a program to monitor multiple file descriptors, waiting until one or more of the file descriptors become "ready" for some class of I/O operation (e.g., input possible). A file descriptor is considered ready if it is possible to perform the corresponding I/O operation (e.g., read(2)) without blocking.

b. Header Files:
   i. #include <sys/select.h>

c. Syntax:
int select(int nfds, fd_set *restrict readfds, fd_set *restrict writefds, fd_set *restrict errorfds, struct timeval *restrict timeout);

d. Params:
   i. **nfds:** argument specifies the range of descriptors to be tested. The first nfds descriptors shall be checked in each set; that is, the descriptors from zero through nfds-1 in the descriptor sets shall be examined.
   ii. **readfds:** if not a null pointer, 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.
   iii. **writefds:** if not a null pointer, 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.
   iv. **errorfds:** if not a null pointer, 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.
   v. **timeout:** parameter controls how long the select() function shall take before timing out. If the timeout parameter is not a null pointer, it specifies a maximum interval to wait for the selection to complete. If the specified time interval expires without any requested operation becoming ready, the function shall return.

e. Return Value**:** Upon successful completion, the pselect() and select() functions shall return the total number of bits set in the bit masks. Otherwise, -1 shall be returned, and errno shall be set to indicate the error.
f. Structure used:

```
struct timeval {
        time_t      tv_sec;     /* seconds */
        suseconds_t tv_usec;    /* microseconds */
    };
```

16.setsockopt(): set the socket options

a. Description: function shall set the option specified by the option_name argument, at the protocol level specified by the level argument, to the value pointed to by the option_value argument for the socket associated with the file descriptor specified by the socket argument.
b. Header Files:
    i. #include <sys/types.h>
    ii. #include <sys/socket.h>
c. Syntax:
   int setsockopt(int socket, int level, int option_name, const void *option_value, socklen_t option_len);
d. Params:
    i. **socket**: socket associated with the file descriptor
    ii. **level**: specifies the protocol level at which the option resides
    iii. **option_name**: specifies a single option to set
    iv. **option_value**: a pointer to the buffer in which the value for the requested option is specified
    v. **option_len**: the size, in bytes, of the buffer pointed to by the optval parameter
e. Return Value: Upon successful completion, setsockopt() shall return 0. Otherwise, -1 shall be returned and errno set to indicate the error.

17.fcntl(): manipulate file descriptor

a. Description: fcntl() performs one of the operations described below on the open file descriptor fd. The operation is determined by cmd.
b. Header Files:
    i. #include <unistd.h>
    ii. #include <fcntl.h>
c. Syntax:
   int fcntl(int fd, int cmd, ...);
d. Params:
    i. **fd:** open file descriptor
    ii. **cmd:** determines operation

iii. **…**: can take an optional third argument. Whether or not this argument is required is determined by cmd. The required argument type is indicated in parentheses after each cmd name (in most cases, the required type is int, and we identify the argument using the name arg), or void is specified if the argument is not required.

e. Return Value: Upon successful completion, the value returned shall depend on cmd. Otherwise, -1 shall be returned and errno set to indicate the error.

18.getpeername(): get the name of the peer socket

a. Description: shall retrieve the peer address of the specified socket, store this address in the sockaddr structure pointed to by the address argument, and store the length of this address in the object pointed to by the address_len argument.

b. Header Files:
   i. #include <sys/types.h>
   ii. #include <sys/socket.h>

c. Syntax:
   int getpeername(int socket, struct sockaddr *restrict address, socklen_t *restrict address_len);

d. Params:
   i. **socket:** socket file descriptor
   ii. **address:** points to the sockaddr structure to store the address
   iii. **address_len:** should be initialized to indicate the amount of space pointed to by address

e. Return Value: Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and errno set to indicate the error.

f. Structure used:

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

19.gethostname(): get hostname

a. Description: returns the null-terminated hostname in the character array name, which has a length of len bytes. If the null-terminated hostname is too large to fit, then the name is truncated, and no error is returned.

b. Header Files:
   i. #include <unistd.h>

c. Syntax:
   int gethostname(char *name, size_t len);

d. Params:
    i. **name:** character array to store hostname
    ii. **len:** length of the character array pointed to by name
e. Return Value: On success, zero is returned. On error, -1 is returned, and errno is set appropriately.

20.gethostbyname(): get network host entry

a. Description: 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)), or an IPv6 address in colon (and possibly dot) notation.
b. Header Files:
    i. #include <netdb.h>
c. Syntax:
   struct hostent *gethostbyname(const char *name);
d. Params:
    i. **name:** hostname, or an IPv4 address in standard dot notation, or an IPv6 address in colon
e. Return Value: returns the hostent structure or a NULL pointer if an error occurs. On error, the h_errno variable holds an error number. When non-NULL, the return value may point at static data, see the notes below.
f. Structure used:

```
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 */
}
#define h_addr h_addr_list[0] /* for backward compatibility */
```

21.gethostbyaddr(): get network host entry

a. Description: returns a structure of type hostent for the given host address addr of length len and address type type
b. Header Files:
    i. #include <netdb.h>
    ii. #include <sys/socket.h>
c. Syntax:
   struct hostent *gethostbyaddr(const void *addr, socklen_t len, int type);
d. Params:
    i. **addr:** given host address
    ii. **len**: length of host address pointed by addr

iii. **type:** type of address

e. Return Value: returns the hostent structure or a NULL pointer if an error occurs. On error, the h_errno variable holds an error number. When non-NULL, the return value may point at static data, see the notes below.

f. Structure used:

```
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 */
}
#define h_addr h_addr_list[0] /* for backward compatibility */
```

Learning Outcomes:

1. Understood network related system calls.
2. Discovered their syntax, description and working.
3. Explored about their return values.