Page No.:
Date:

1. Why socket address structure is required in Unix network programming? Explain various socket structure available in Unix socket programming.

→ - A socket address structure is used in network programming to specify the address of a socket end point, which is a combination of an IP address and a port number.
- The socket address structure provides the information needed by the socket API to identify the network location of a socket endpoint.

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

There are two types of socket address structure:

(i) For IPv4:
- The socket address structure for IPv4 typically consists of two components:
  - Internet address (IP address)
  - 2 port number

```
struct sockaddr_in {
    short int sin_family;
    unsigned short int sin_port;
    struct in_addr sin_addr;
    unsigned char sin_zero[8];
};
```

```
struct in_addr {
    unsigned long s_addr;
};
```

(ii) For IPv6:

- The socket address structure for IPv6 is similar to that for IPv4, with a few differences.

```
struct sockaddr_in6 {
    sa_family_t    sin6_family;
    in_port_t      sin6_port;
    ~~in~~ ~~uint~~ u_int32_t   sin6_flowinfo;
    struct in6_addr  sin6_addr;
    u_int32_t      sin6_scope_id;
};

struct in6_addr {
    u_int8_t  s6_addr[16];
};
```

2. How socket address structure is passed from process to kernel and kernel to process. Explain with the help of diagram and functions.

→ • Value-Result Arguments:

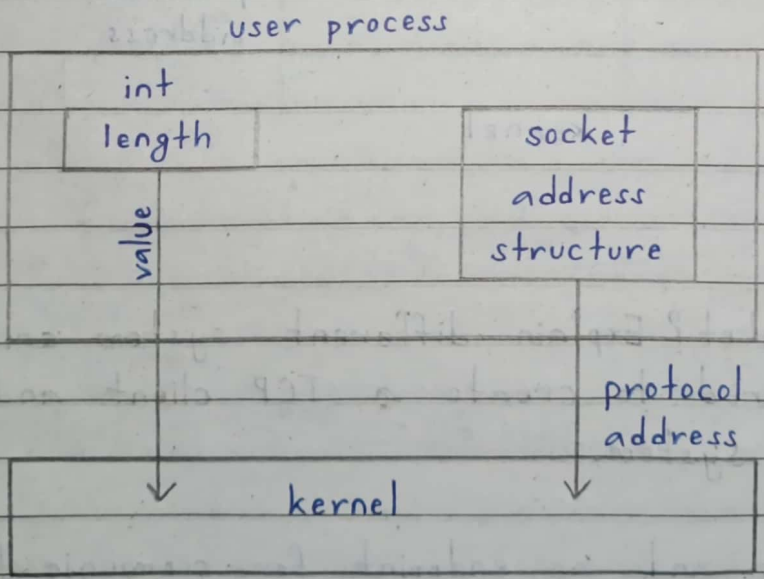
- When a socket address structure is passed to any socket function, it is always passed by reference (a pointer to the structure is passed).

- The length of the structure is also passed as an argument.
- In Unix network programming, the "value result argument" refers to a technique used to return multiple values from a function.
- The way in which the length is passed depends on which direction the structure is being passed:

(i) From the process to the kernel
(ii) From the kernel to the process

(i) From process to kernel:
- bind(), connect(), and sendto() functions pass a sockect address structure from the process to the kernel.

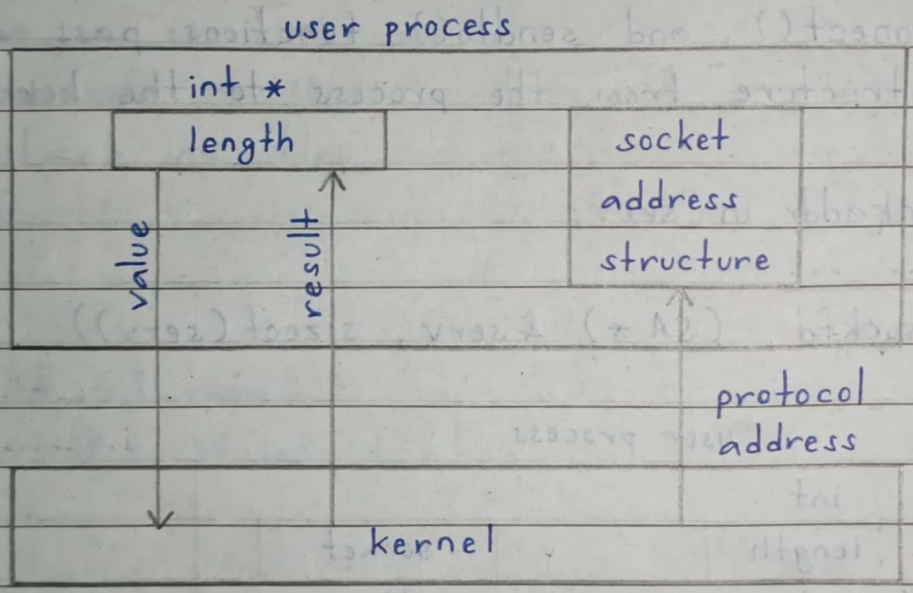
```
struct sockaddr_in serv;

connect (sockfd, (SA *) &serv, sizeof(serv));
```



(ii) From kernel to process:

- accept(), recvfrom(), getsockname(), and getpeername() functions pass a socket address structure from the kernel to the process.

```
struct sockaddr_un cli;
socklen_t len;
len = sizeof (cli);
getpeername (unixfd, (SA *) &cli, &len);
```

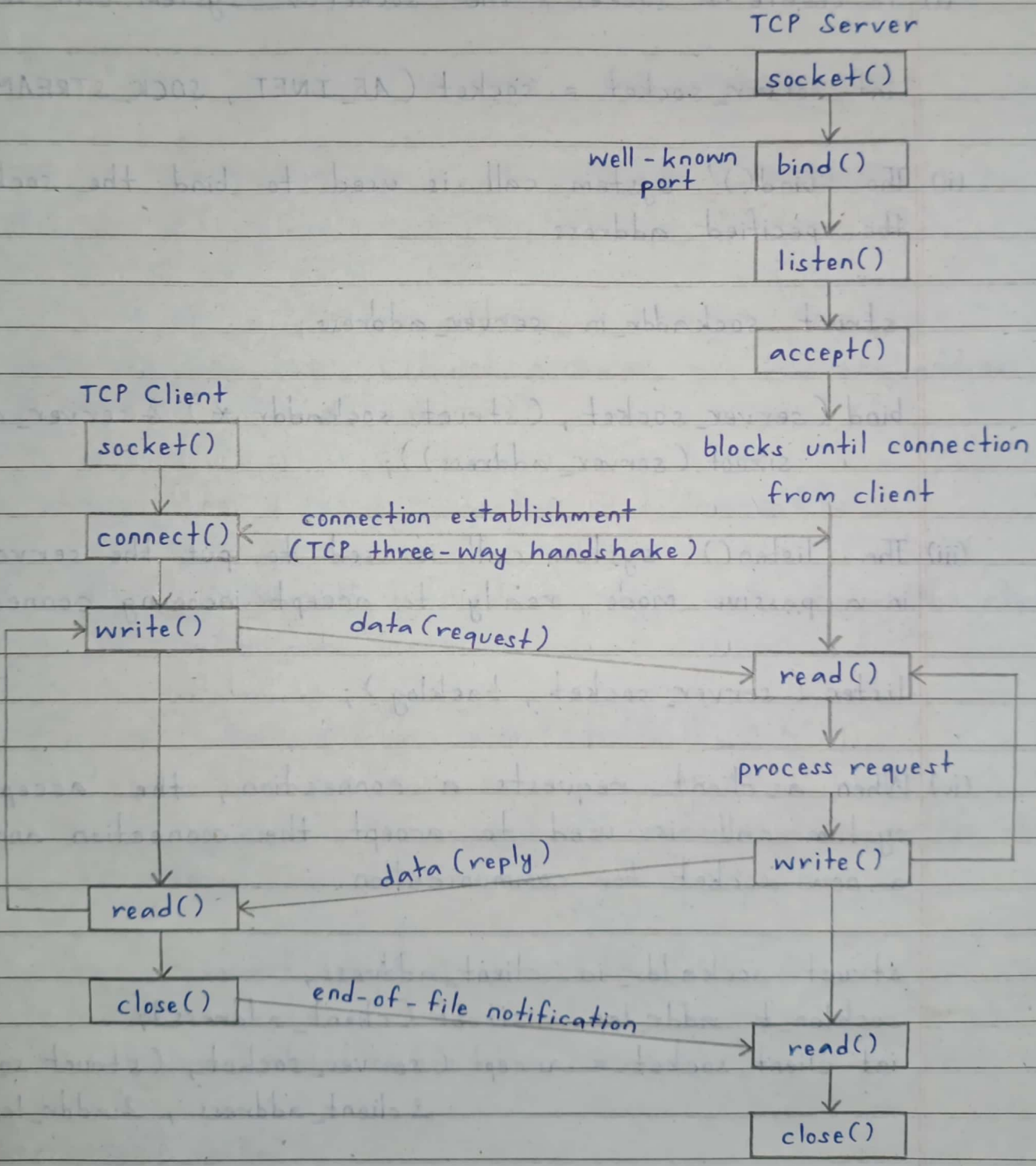


3. What is socket? Explain different system call in specific order, required to create a TCP client and TCP server in the Unix System.

→ A socket is ~~an~~ an endpoint for communication between two computers on a network.

Page No.:
Date:

It provides a common interface for network communication, allowing different applications and services to exchange data.



Page No.:
Date:

TCP Server Creation:

(i) To create a socket, the 'socket()' system call is used.

```
int server_socket = socket (AF_INET, SOCK_STREAM, 0);
```

(ii) The 'bind()' system call is used to bind the socket to the specified address.

```
struct sockaddr_in server_address;

bind (server_socket, (struct sockaddr *) &server_address,
      sizeof (server_address));
```

(iii) The 'listen()' system call is used to put the server socket in a passive mode, ready to accept incoming connections.

```
listen (server_socket, backlog);
```

(iv) When a client requests a connection, the 'accept()' system call is used to accept the connection and create a new socket for communication.

```
struct sockaddr_in client_address;
socklen_t addr_len = sizeof (client_address);
int client_socket = accept (server_socket, (struct sockaddr *)
                            &client_address, &addr_len);
```

Page No.:
Date:

TCP Client Creation:

(i) Similar to the server, a socket for the client is created.

```
int client_socket = socket (AF_INET, SOCK_STREAM, 0);
```

(ii) ~~The~~ The 'connect()' system call is used to establish a connection to the server.

```
struct sockaddr_in server_address;
connect (client_socket, (struct sockaddr *) &server_address,
        sizeof (server_address));
```

TCP Communication (After Connection Establishment):

(i) The 'send()' system call is used to send data from the client to the server or vice versa.

```
send (client_socket, buffer, sizeof(buffer), 0);
```

(ii) The 'recv()' system call is used to receive data on the client or server side.

```
recv (server_socket, buffer, sizeof (buffer), 0);
```

(iii) The sockets are closed when the communication is complete.

```
close (client_socket);
close (server_socket);
```

Page No.:
Date:

4. How synchronous communication is different from asynchronous communication? Explain various byte ordering and byte manipulation functions with prototype and example system call.

Feature	Synchronous communication	Asynchronous communication
Definition	Synchronous communication is a blocking communication where each operation waits for completion.	Asynchronous communication is a non-blocking communication where operations do not wait for completion.
Flow of execution	It is sequential and ordered. Sender and receiver operate in a synchronized manner.	It is not necessarily sequential. Sender and receiver operate independently.
Blocking behavior	Blocking calls; operations wait until completion before proceeding.	Non-blocking calls; operations do not wait and return control immediately.
Efficiency	Simplicity at the cost of potential inefficiencies due to waiting.	Improved efficiency as processes can continue with other tasks while waiting for response.
Complexity	It is simple to understand and implement.	It is more complex due to the need for callback functions, event handling, or message queues.

Page No.:
Date:

Some of the byte ordering functions are as follows:

(i) 'htonl'

- It converts 32-bit unsigned integer from host byte order to network byte order.

```
uint32_t htonl (uint32_t value);
```

(ii) 'ntohl'

- It converts 32-bit unsigned integer from network byte order to host byte order.

```
uint32_t ntohl (uint32_t value);
```

Some of the byte manipulation functions are as follows:

(i) 'memcpy'

- It copies a specified number of bytes from one memory location to another.

```
void *memcpy (void *dest, const void *src, size_t n);
```

(ii) 'memset'

- It sets a specified number of bytes in a block of memory to a given value.

```
void *memset (void *s, int c, size_t n);
```

Page No.:
Date:

5. Why do we need byte ordering in network programming? Differentiate little endian and big endian. Explain different address conversion function with prototype and return type of respective function.

→ Byte ordering is needed in network programming because different computer architectures may store multi-byte data types in memory differently.

The two main byte orderings are as follows:

(i) Big endian:
- The most significant byte is stored at the lowest memory address.
- E.g. The value '0x1234' is stored as '12 34'.

(ii) Little endian:
- The least significant byte is stored at the lowest memory address.
- E.g. The value '0x1234' is stored as '34 12'.

The different address conversion function includes:

(i) Host to Network Long (htonl)
(ii) Host to Network Short (htons)
(iii) Network to Host Long (ntohl)
(iv) Network to Host Short (ntohs)

Page No.:
Date:

6. What are the ways to pass the length of socket structure for different socket API's argument? Explain them in detail with function prototype and argument detail.

→ - In socket programming, passing the length of the socket structure is crucial for the proper functioning of various socket APIs.
- The length is often required to distinguish between different versions of the socket structure.
- Some of the ways to pass the length of socket structure for different socket API are as follows:

(i) bind function:

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

sockfd: socket file descriptor
addr: pointer to the sockaddr structure
addrlen: length of the sockaddr structure pointed by addr

(ii) connect function:

```
int connect (int sockfd, const struct sockaddr *addr,
             socklen_t, addrlen);
```

sockfd: socket file descriptor
addr: pointer to the sockaddr structure containing the address of the remote socket

Page No.:
Date:

(iii) accept function:

```
int accept (int sockfd, struct sockaddr *addr, socklen_t,
            addrlen);
```