Network programmmming 2nd unit

1.Explain apis for
concurrent server with neat diagrams

Ans -

Fork()

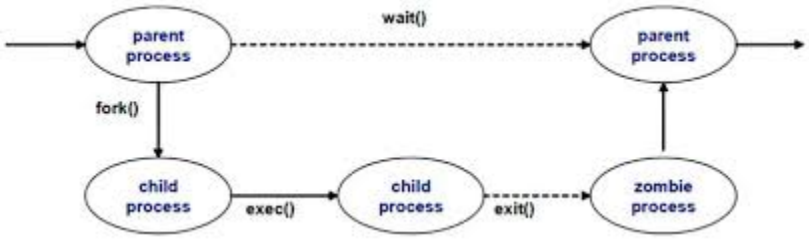
A process makes a copy of itself so that one copy can handle one operation while the other copy does another task. This is typical for network servers.

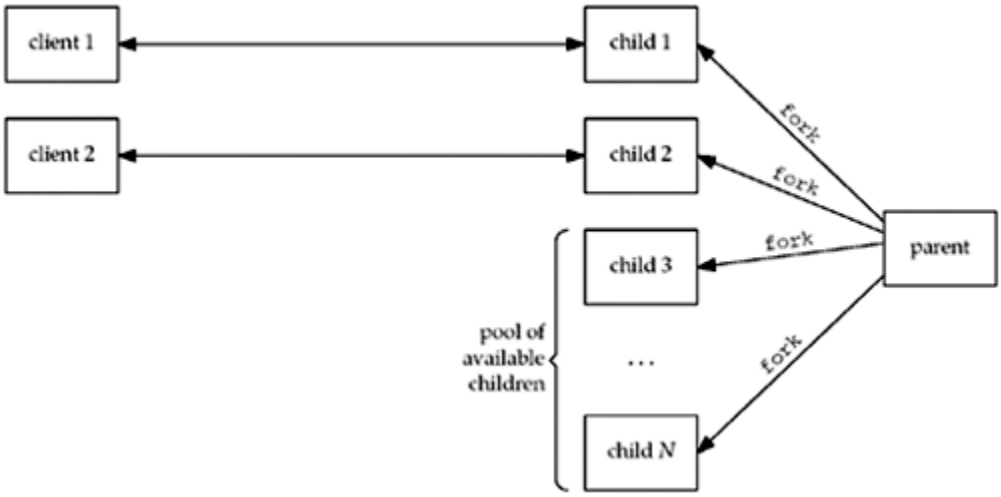
A process wants to execute another program parallelly. Since the only way to create a new process is by calling fork, the process first calls fork to make a copy of itself, and then one of the copies calls exec to replace itself with the new program. This is typical for programs such as shells.

exec()
replaces the current process image with the new program file, and this new program normally starts at the main function. The process ID does not change.

```
//
fork()

int main()

{ fork();

printf("Hello world!\n");

return 0; }


/*
For exec() */

int main()

{ int i;

printf("I am EXEC.c called by execvp() ");

return 0;

}
```

```
//
execDEMO.c
int main()
{
char *args[]={"./EXEC",NULL};
execv(args[0],args);
printf("Ending-----");
return 0;
}
```





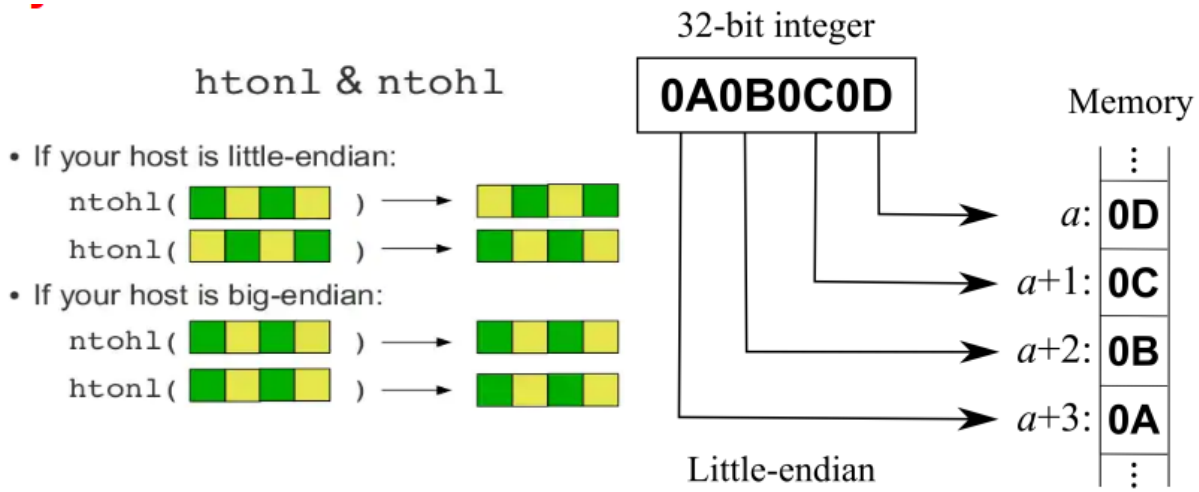
2.Explain the primary byte conversion functions

Functions are
htons(): host to network short — This function converts 16-bit quantities from host byte order to network byte order.

ntonl(): host to network long — This function converts 32-bit quantities from host byte order to network byte order.

ntohs(): network to host short — This function converts 16-bit quantities from network byte order to host byte order.

ntohl(): network to host long — This function converts 32-bit quantities from network byte order to host byte order.



3.**Explain elecemtary TCP socket functions with code sniptes or write the full code**

(10 or 8 marks)

socket function-
#include <sys/socket.h>
int socket (int family, int type, int protocol) ;
Returns: non-negative descriptor if OK, -1 on error

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

family specifies the protocol family

The protocol argument to the socket function should be set to the specific protocolor 0 to select the system's default for the given combination of family and type.

2.bind Function

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.

#include <sys/socket.h>
int bind (int sockfd, const struct sockaddr *myaddr, socklen_t addrlen);*
/ Returns: 0 if OK,-1 on error */

3.listen Function

The listen function is called only by a TCP server and it performs two actions:
Convert 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.

#include <sys/socket.h>
int listen (int sockfd, int backlog);
/* Returns: 0 if OK,-1 on error */

4.accept Function

The accept function is called by a TCP server to return the next completed connection from the front of the completed connection queue. If the completed connection queue is empty, the process is put to sleep.

#include <sys/socket.h>
int accept (int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);*
/ Returns: 0 if OK,-1 on error */

5.connect Function

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

#include <sys/socket.h>
int connect(int sockfd, const struct sockaddr *servaddr, socklen_t addrlen);*
/ Returns: 0 if OK,-1 on error */

4.List and explain client/server functions of a TCP echo client

same as 3rd answer, In the snipet they can ask like the IA question where we need to ractify the code and correct the mistakes.

IA snippet corrected answer

5.**Explain the getsocketname and getpeername functions with code snips (vey low)**

These two functions return either the local protocol address associated with a socket (getsockname) or the foreign protocol address associated with a socket (getpeername).

#include <sys/socket.h>
int getsockname(int sockfd, struct sockaddr * localaddr, socklen_t * addrlen);
int getpeername(int sockfd, struct sockaddr * peeraddr, socklen_t * addrlen);
Both return: 0 if OK, -1 on error

These two functions are required for the following reasons:
 After connect successfully returns in a TCP client that does not call bind , getsockname returns the local IP address and local port number assigned to the connection by the kernel.
 After calling bind with a port number of 0 (telling the kernel to choose the local port

number), getsockname returns the local port number that was assigned.
 getsockname can be called to obtain the address family of a socket
 In a TCP server that binds the wildcard IP address , once a connection is established with a client (accept returns successfully), the server can call getsockname to obtain the local IP address assigned to the connection. The socket descriptor argument in this call must be that of the connected socket, and not the listening socket

6.**Explain socket adress structire**

Pv4 Socket Address Structure

An IPv4 socket address structure, commonly called an "Internet socket address structure," is named sockaddr_in and is defined by including the <netinet/in.h> header.

The Internet (IPv4) socket address structure: sockaddr_in.

```
struct in_addr {
  in_addr_t   s_addr;           /* 32-bit IPv4 address */
                                /* network byte ordered */
};

struct sockaddr_in {
  uint8_t         sin_len;      /* length of structure (16) */
  sa_family_t     sin_family;   /* AF_INET */
  in_port_t       sin_port;     /* 16-bit TCP or UDP port number */
                                /* network byte ordered */
  struct in_addr  sin_addr;     /* 32-bit IPv4 address */
                                /* network byte ordered */
  char            sin_zero[8];  /* unused */
};
```

The four socket functions that pass a socket address structure from the process to the kernel, bind, connect, sendto, and sendmsg.

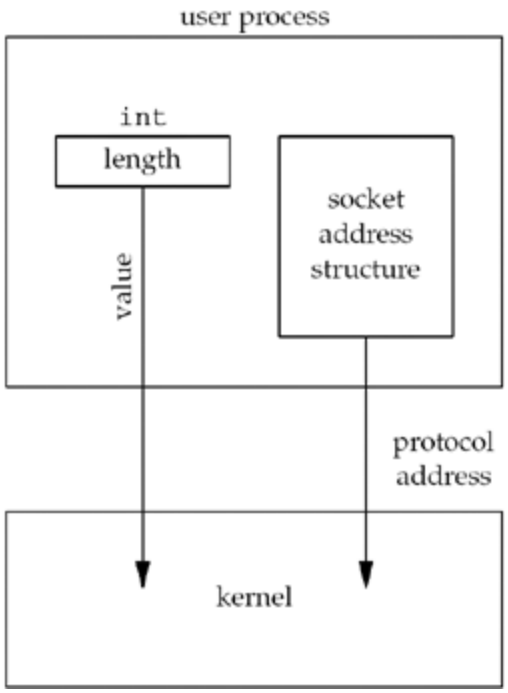
The five socket functions that pass a socket address structure from the kernel to the process, accept, recvfrom, recvmsg, getpeername, and getsockname, all set the sin_len member before returning to the process.

7.**Value-Result Arguments**

Three functions, bind, connect, and sendto, pass a socket address structure from the process to the kernel. One argument to these three functions is the pointer to the socket address structure and another argument is the integer size of the structure, as in

```
struct sockaddr_in serv;

/* fill in serv{} */
connect (sockfd, (SA *) &serv, sizeof(serv));
```

Since the kernel is passed both the pointer and the size of what the pointer points to, it knows exactly how much data to copy from the process into the kernel.



Four functions, accept, recvfrom, getsockname, and getpeername, pass a socket address structure from the kernel to the process, the reverse direction from the previous scenario. Two of the arguments to these four functions are the pointer to the socket address structure along with a pointer to an integer containing the size of the structure, as in

```
struct sockaddr_un  cli;   /* Unix domain */
socklen_t  len;
```

```
len = sizeof(cli);         /* len is a value */
getpeername(unixfd, (SA *) &cli, &len);
/* len may have changed */
```