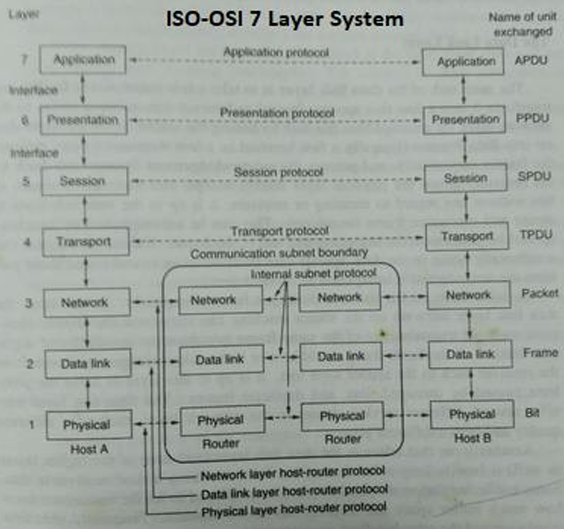
Network Programming

1. **Network Programming fundamentals**
   1. Introduction to networking and network programming
   2. Client/Server mode(paradigm)
   3. Communication Protocol:
      1. IP
      2. UDP
      3. TCP
      4. SCTP
   4. TCP-State transition diagram
   5. Protocol comparison
2. **UNIX Programming**
   1. Introduction to socket
   2. Socket address structure
   3. Value result arguments
   4. Byte ordering and manipulation functions
   5. Fork and exec functions
   6. Concurrent server
   7. UNIX Domain socket
   8. Internet Domain socket
   9. Socket calls
   10. Passing file descriptor
   11. Input/Out put models
       1. Blocking
       2. Nonblocking
       3. Multiplexing
       4. Signal driven
       5. Asynchronous model
   12. Socket options
       1. Getsockopt
       2. Setsockopt
       3. Fcntl
   13. Daemon process
       1. Syslogd daemon
       2. Syslog function
   14. Ioctl function
   15. Ioctl operations
   16. Socket operations
   17. Client/Server Unix & Internet domain implementation using C
3. **Winsock programming**

**Chapter-1**

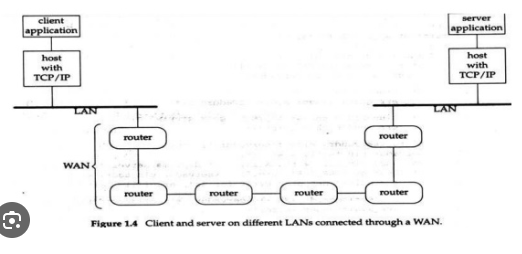
1. Introduction to Network and network programming
   1. Computer network: Is a system that connects multiple computers enabling them to communicate and share resources like; data, files, applications and hardware.

Reference: Network model



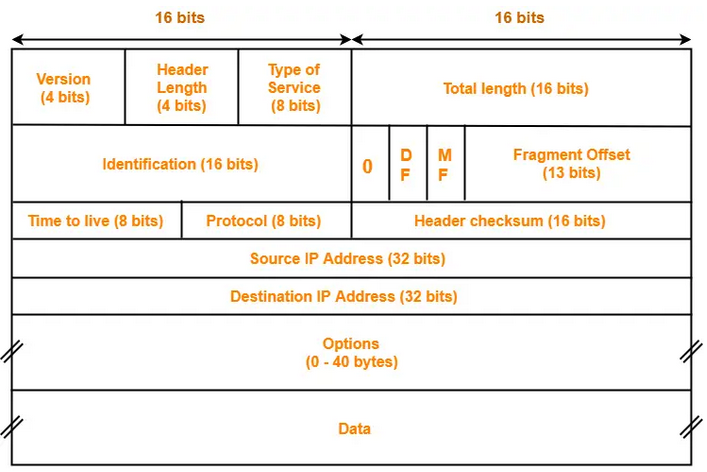
* 1. Network programming: is the process of writing software that allows application to communicate across the network where it enables to send and receive data over the network or internet
     1. Key components in network programming are:
        1. Protocols
        2. Sockets
        3. Client/server
        4. Network programming libraries
           1. C, C++, .Net, Java,Python etc.

1. Client/Server paradigm

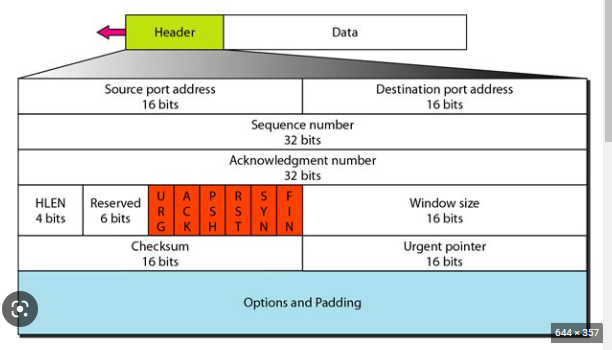


* 1. Server process:
     1. Application running of host having below features
        1. Handle communication TCP/UDP/IP
        2. Server can be thought of long-running program which response to request from peer called client eg Web server.
        3. Transfer data over communication based on I/O model
        4. Terminate communication
        5. Provide requested service
        6. Handle multiple client
        7. Communicate through end point this can be connected or not connected endpoint based on communication protocol
  2. Client process:

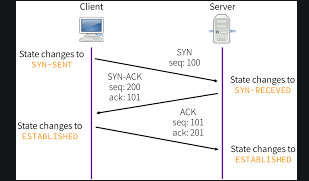
1. Client communicate with on server at a time
2. To communicate first server must ready to accept the client request
3. Client request for service and Communication through connected or not connected endpoint based on communication protocol.
4. Communication protocol
   1. Protocol: In computer networks, a **protocol** is a set of rules and standards that define how data is transmitted, received, and interpreted across devices in a network and respond back to the requester.
   2. Network communication protocol:
      1. IP



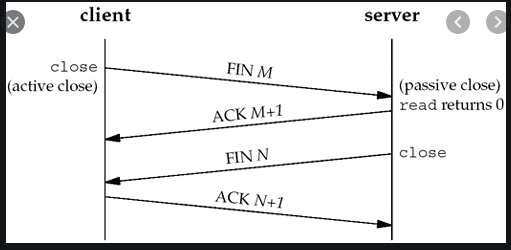
* + 1. TCP



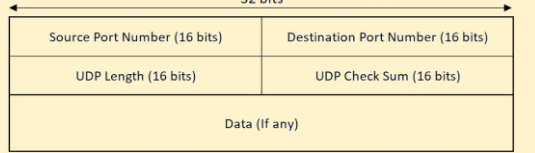
1. TCP 3-way handshake



TCP 4-way termination

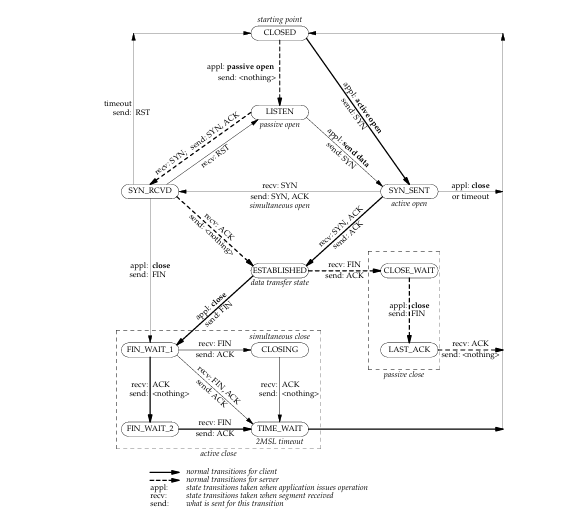


* + 1. UDP



* + 1. SCTP

1. TCP state transition diagram



1. TCP state description
   1. CLOSED: It represent no connection state at all.
   2. LISTEN: State for TCP server waiting for connection request from remote TCP on well known port.
   3. SYN\_SENT: If application perform active open in closed state then it will send SYN packet

And new state will be SYN\_SENT.

* 1. SYN\_RECEIVED: If application receive SYN and send its SYN and ACK to peer process then

State will set as SYN\_RECEIVED.

* 1. ESTABLISHED: Once SYN and ACK receive from peer process then state set as established

Which is the connected state where data transfer can be done.

* 1. FIN\_WAIT\_1: if application calls close before receiving FIN(an active close) the state will set

To FIN\_WAIT\_1

* 1. FIN\_WAIT\_2: represent waiting for TCP connection termination request from remote TCP.

Client (SEND FIN and RECIVE ACK but no FIN from Server)

* 1. CLOSE\_WAIT: If application receives FIN in ESTABLISHED state(passive close) the transition is

CLOSE\_WAIT.

* 1. CLOSING: Represent waiting for connection termination request ACK from remote TCP.
  2. LAST\_ACK: Represent waiting for ACK of the connection termination request previously

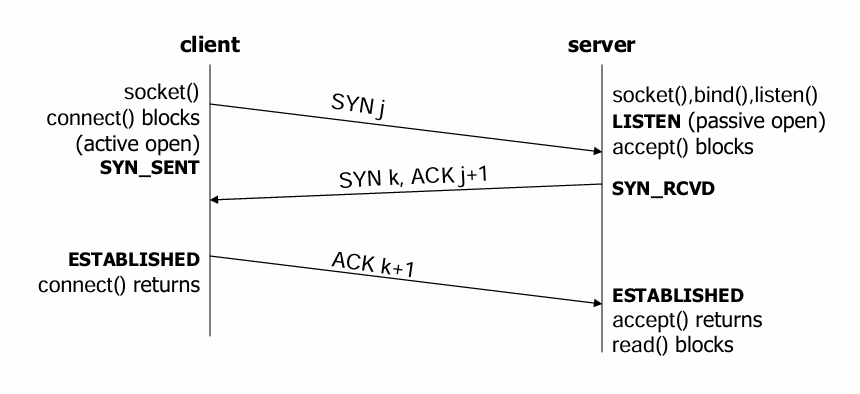
Sent to the remote TCP.

* 1. TIME\_WAIT: Represent waiting for enough time to make sure that remote TCP will

Received the ACK of it connection termination request.

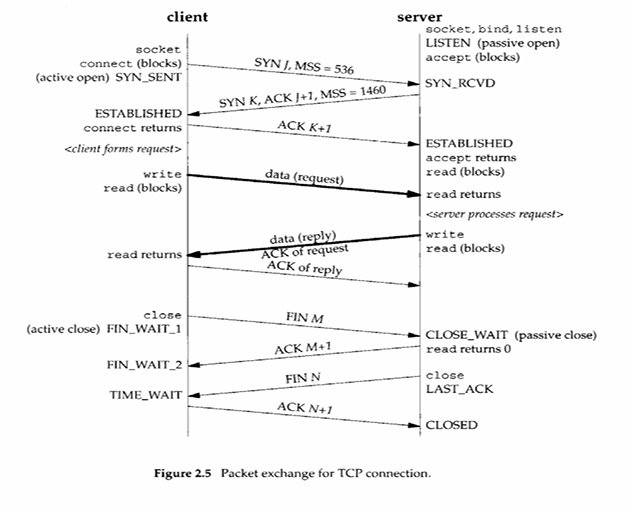
* 1. CLOSED: Represent no connection at All.

1. TCP-3 Way handshake

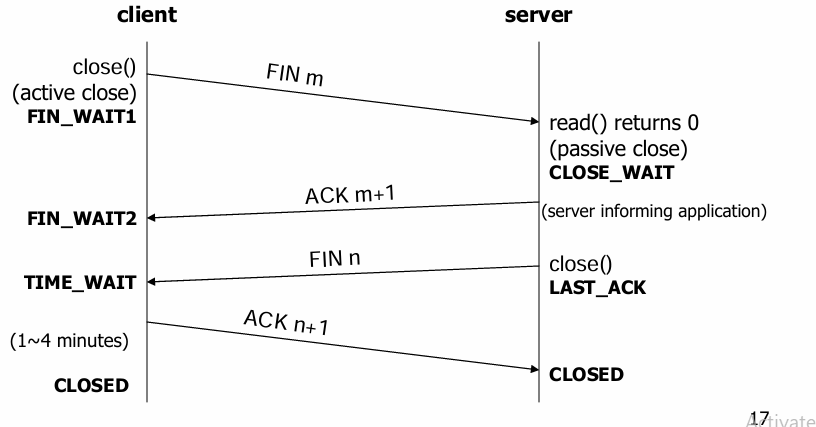


* 1. Step-1 socket is used to create the end point followed by connect call which indeed request the connection with the peer here TCP will send the SYN packet and maintain TCP state as SYN\_SENT.
  2. Step-2 At server it will execute socket() system call to create endpoint , bind() is used to map between ip , port and end point and use listen() to accept connection from client to server and put the active socket to passive. Here accept blocks means server keep waiting for the connection from client.
  3. Step-3 Once SYN packet received from client server will set its TCP as SYN\_RCVD state along with its own TCP SYN Packet to client.
  4. Step-4 once client receive the ACK for its SYN packet along with server SYN packet it will mark it TCP state as ESTABLISHED and return the ACK to server. In this state half connection is established between client and server from (Client side connection complete).
  5. Step-4 Server is waiting for the ACK from client which know as accept return and wait for reading the packet from client (Read Block), once ACK received for its SYN packet server as well will maintain its TCP state as ESTABLISHED now half connection from server done and connection is established between client/server and known as connected state, which is actually is in data transfer state.

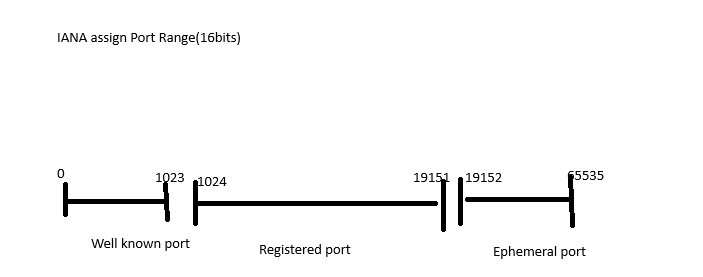
1. Packet Exchange for TCP Connection



1. Four-way termination



1. Port address for Transport layer (IANA assign)



**CHAPTER-2 UNIX NETWORK PROGRAMMING**

1. Socket Introduction
   1. Socket is communication end point used for exchanging information between

Process running with in the same system or between two process across the network.

* 1. Socket act as interface between the application and transport layers and helps in transmitting the data by accessing under lying protocol like TCP/IP.
  2. Types of socket.
     + Stream socket: User file transfer, web browser
     + Datagram socket: Online gaming, video streaming
     + Raw socket: custom protocol development or network diagnostics
  3. High level socket process flow
     + At least two process involved
       1. Client: request service
       2. Server: facilitate requested service
     + List of steps in TCP Server socket
       1. Creates endpoint
       2. Bind to ip, port and endpoint
       3. Listen for client
       4. Accept client request
       5. Read data/Write data
       6. Close connection
     + List of steps in TCP client socket
       1. Creates endpoints
       2. Connect server
       3. Write data
       4. Read data
       5. Close
     + List of steps in UDP server/Client
       1. Create end points use by both clients/server
       2. Bind to port, ip and endpoint used only by server
       3. Close socket

1. Socket address structure
   1. Network communication is performed mainly using the default standard called TCP/IP protocol stack.
   2. To access the underlying IP protocol structure name **sockaddr\_in** comes under **library <netinet/in.h>** in UNIX/LINUX case. ***windows have different case***.
   3. Structure proto type

struct sockaddr\_in {

sa\_family\_t sin\_family;//address family

in\_port\_t sin\_port;//TCP port address(16bits)

struct in\_addr sin\_addr;//IPV4 address(32 bits)

char sin\_zero[8];//padding not used(Reserved)

};

* 1. Example of socket address implemented using C.

Struct sockaddr\_in server;//creating object for sockaddr structure

memset(&server, '0', sizeof(server));//optional initializing 0 to structure address

server.sin\_family=AF\_INET;//assign family as internet domain

server.sin\_addr.s\_addr=htonl(INADDR\_ANY);//assigning ipv4 address default eth.

server.sin\_port=htons(PORT);//assign port to application

* 1. Socket family
     + AF\_INET//for internet domain v4
     + AF\_INET6//for internet domain ipv6
     + AF\_UNIX//for unix domain socket
     + AF\_PACKET//RAW packet at network layer
     + AF\_RAW//Raw network protocol access
  2. Port
     + Used to identify the application and 16 bits’ number
  3. Sin\_addr
     + Internet IP protocol address
  4. Htons() byte manipulation function for network byte odering for short integer
  5. INADDR\_ANY it will automatically select IP configured at active interface.

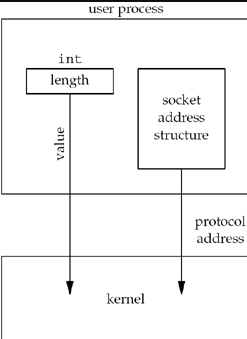
1. Value result argument
   * + In socket programming to pass value and receive the result in same parameter when message is passed between the two process with common function and arguments know as value result argument. Basically it refers to the arguments(parameter) that are both input(value) and output(result).
     + Example using c:
       1. int test(int b, int \*b);
       2. here for 2nd parameter we can send value and receive the result from the same parameter known as value result arguments.
     + Let understand value result argument based on the socket calls
     + Step: To understand we have to know the types of call that will pass the value from **process to kernel** and kernel to the process.
     + Socket call that will pass argument from process to kernel are.
       1. Bind(),connect(),sendto()

Struct sockaddr\_in server;

Connect(fd,(SA\*)&server,sizeof(server));

//here information will fill to server.

* + - 1. Since kernel is passed both pointer and size of what the pointer points to. Kernel know exactly knows how much data to copy from the process into kernel e.g diagram



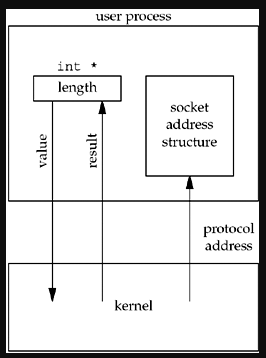
* + - Socket call that will pass argument from kernel to process are
      1. Accept(),recvfrom() and getsockname()
      2. Eg using c

Struct sockaddr\_un client;

Socklen\_t len;

Len=sizeof(client);

Getpeername(fd,(SA\*)&client,&len);//len may have changed.



* + - The above diagram will demonstrate the concept of value result diagram in network programming
    - In the above two example we try to send address structure from the kernel to process and receiving address structure from kernel to process and send address structure may vary received address structure which is achieved by using the length parameter as integer to pointer as term as value result argument.

1. Byte ordering and Manipulation function
   1. Byte order: it is the order of digits in system, which refers to how data is organized and transmitted across the network.
   2. Two ordering technique are
      * Host Byte order
      * Network Byte order
        1. Host Byte Oder:
           1. Refers to way data is stored in memory on a particular system
           2. Ordering technique can be Big-endian or Little-endian depending on the architecture of processor.
           3. Big-endian: Most significant byte(MSB) of data stored first(at lowest memory address).
           4. Little-endian: The least significant byte(LSB) is stored first(at lowest memory address).
           5. Example: 0x12345678

Big-endian stores it as: 0x12 0x34 0x56 0x78

Little-endian:0x78 0x56 0x34 0x12

* + - 1. Network Byte order
         1. Is a standard format used for data transmission over network
         2. Ordering technique is always big-endian.
         3. This is because to make uniformity for network protocol(like TCP/UDP/IP) to communicate regardless of host machine byte order.
  1. Byte ordering functions: Used for conversion.
     + htons()
     + htonl()
     + ntohs()
     + ntohl()
  2. example:
  3. header file <arpa/inet.h> for hotnl,hotns,ntohl,ntohs.
  4. Byte manipulation functions

1. Fork and exec function
   1. Fork()
      * This system call is used to duplicate the calling process.
      * New process created by fork() is known as the **child** process
      * Calling process is known as **parent** process
      * Once after fork() both the process can continue executing code but the return value is different between them to identify themselves.
      * Return Value:
        1. In parent process:
           1. Returns the Process id(PID) of the child process which is positive value
        2. In child process:
           1. Fork() returns 0
        3. In case of error
           1. Fork() returns -1
      * Fork() key points to understand
        1. Fork duplicate which means it copies the parent process to create the new process and inherits from parent as follows
           1. Parents memory
           2. File descriptor
           3. And other resources
        2. PID and PPID
           1. PID: a unique value to identify the child process
           2. PPID: the parent process ID
      * Syntax for fork
        1. Header file #include<unistd.h>
        2. Example fork() implementation using c
   2. D

// Online C compiler to run C program online

#include <stdio.h>

#include<unistd.h>

int main() {

printf("fork() demo started!!!\n");

int val=100;

pid\_t pid=fork();//creating child process(dup)

//now checking the client and parent process

if(pid<0)

{

printf("Fork failed fork()-->[%d]\n",pid);

return -1;

}

else if(pid==0)

{

printf("---Child code execution start---\n");

val=val+val;

printf("I am chiled process\n");

printf("Parent PID:[%d] and Child PID:[%d]\n",getpid(),pid);

printf("val at client[%d]",val);

printf("\n---Child code execution end---\n");

}

else

{

val=val\*90;

printf("---Parent code execution start---\n");

printf("I am parent process\n");

printf("Parent PID[%d]and Child PID[%d]\n",getpid(),pid);

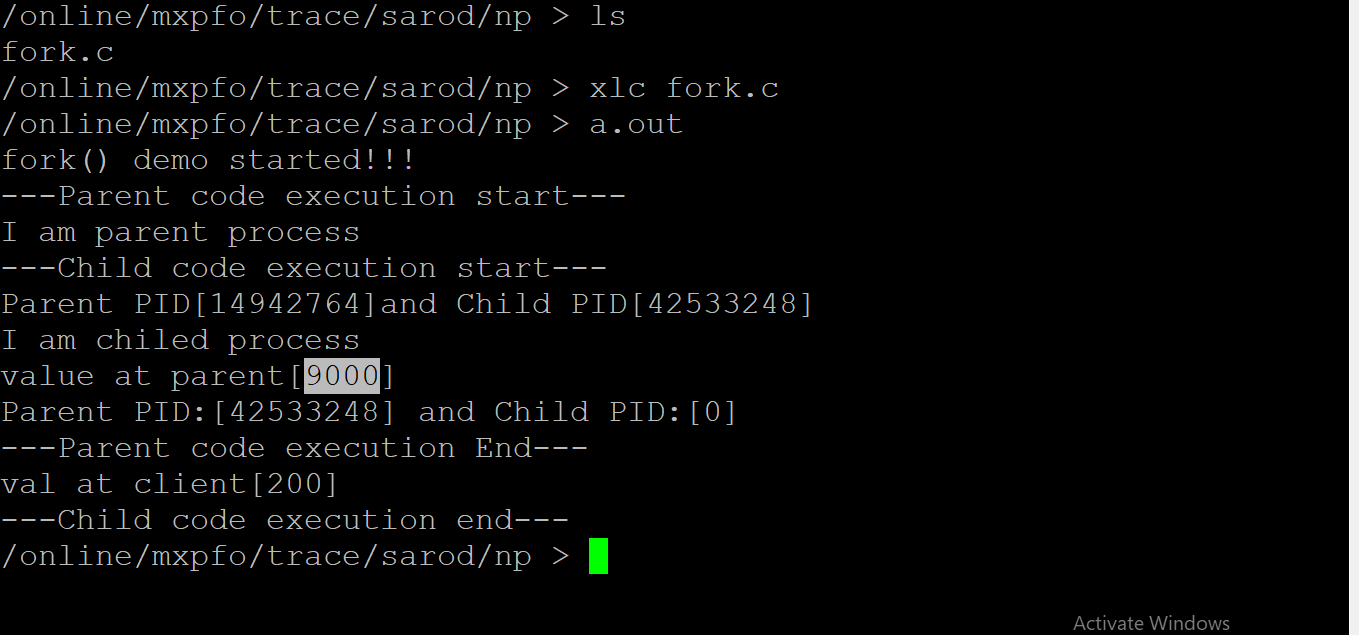
printf("value at parent[%d]\n",val);

printf("---Parent code execution End---\n");

}

return 0;

}



1. Exec() functions
   1. Exec() family of function in c is used to replace the current process image with a new one.
   2. When a process calls exec() function the process is replaced with a new program and new program start the executions and calling process ceases to exist in its original form.
   3. This is basically used after the fork() function to replaced child process with a different program maintain child and parent relationship and running child as separate program to perform different tasks.
   4. Exec functions has different variant slightly different in how argument are passed.
   5. Header file #include<unistd.h>
   6. Parameter definition
      * 1. Path: the path to the program to be executed
        2. Arg0,…,argN: the argument for the new program, arg0 is the name of program, the last argument must be a NULL pointer to terminate argument list
        3. Argv[]: an array of pointer to arguments used for execv() and execvp()
        4. Envp(): an array of environment variable for execle()
        5. Return value:
           1. Successful does not return , the current process replaced by new program
           2. Failure: -1
      * Execl(): executes a program with a list of arguments

Int execl(const char\*path,const char\*arg0,…,(char\*)NULL);

Example: execl(“/bin/ls”,”ls”,”-l”,NULL);

* + - Execp(): executes a program with a list of arguments and searches for it in directories listed in PATH environment variable.

Int execp(const char\*path,const char\* arg0,…,(char\*)NULL);

Note: Similar to execl() but serach for the program in directory listed path.

* + - Execv(): executes a program with array of arguments

Int execv(const char\*path,char \*const argv[]);

Example:

Char \* argv[]={“bin/ls”,”-l”,NULL);

Execv(“/bin/ls”,argv);

* + - Execvp(): executes a program with an array of arguments and searches for it in directories listed in PATH.

Int execvp(const char \*file,char \*const argv[]);

Example:

Char \* argv[]={“ls”,”-l”,NULL);

Execvp(“ls”,argv);

* + - Execle(): executes a program with a process with the new program and don not return if successful and if error return -1

Int execle(const char \* path,char \* arg0, …,(char\*) NULL,char \*const envp[]);

* + - Simple Example

#include <stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/wait.h>

int main() {

execl("/bin/ls","ls","-l",NULL);

printf("If error come this line is print\n");

return 0;

}

1. Concurrent Server
   1. It is the technique to write the server that can handle multiple client simultaneously at any given point of time.
   2. This is important in network programming to ensure that server can server many clients efficiently without making them to wait for each other request to wait.
   3. Technique used for Concurrency
      * Using of fork() function under UNIX
      * Using of pthreads
      * Non-Blocking, I/O Multiplexing
      * Asynchronous I/O
   4. Concurrent Server using fork
   5. Demo program for concurrent server

Pid\_t pid;

Int fd,confd;

fd=socket(…);

bind(fd,…);

listen(fd,5);

for(;;)

{

confd=accept(fd,…);//process block in call waiting client request

if((pid=fork())==0) // wants to create new process as child to handle equest

{

**close(fd)**;//child closing listen socket.

processRequest();

sendResponse();

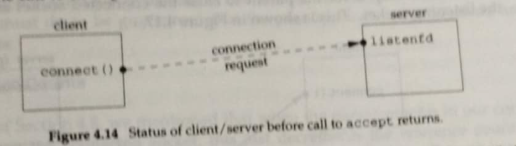
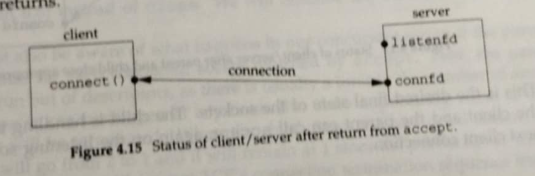
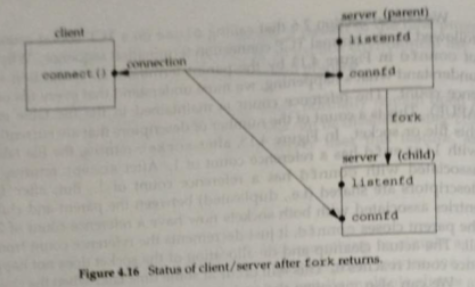
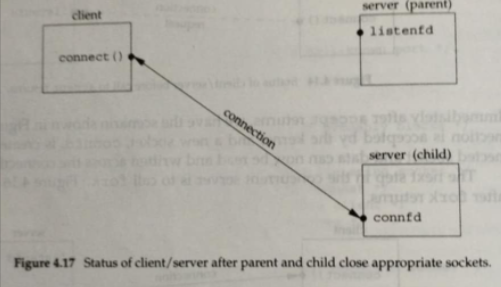
**close(confd);**//child closing connected socket

exit(0);

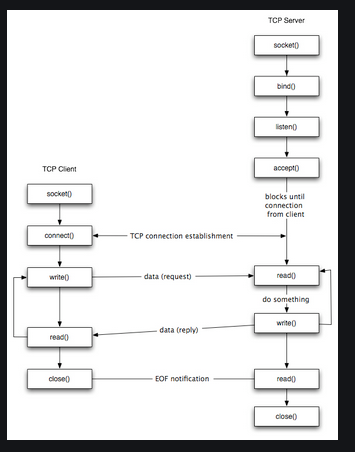
}

close(confd);//parent closing connected socket

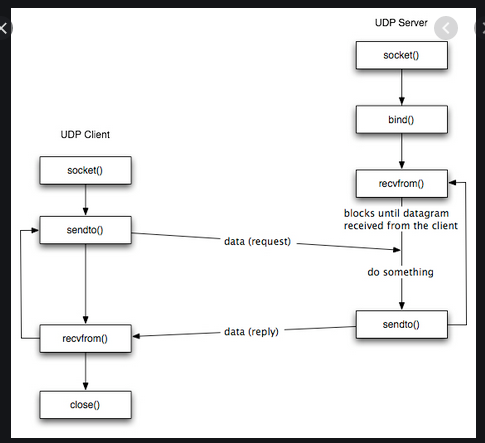
}

* 1. High level process flow for concurrent server
     + When connection is established accepts returns, the server call fork () and child process service the client.
     + The parent process will wait for another client request to process.
     + The parent will close the connected socket since now child will process the request and response for each connection.
     + Here why we have closed the listen and connected socket by parent and child. This is because the kernel will maintain the count of open socket in its table
       1. When socket() returns the listen fd counts is set to 1
       2. When accept() returns the connected socket set to 1
       3. And after fork() both listen and connected socket count is duplicated and set to 2 there for parent just decrementing the count from 2 to 1 in case of connected socket.
       4. However server need one listening socket to accept the connection from client.
  2. High level diagrammatical representation for concurrent operation
  3. 
  4. 
  5. 
  6. 

1. **TCP Client/Server socket flow**



1. **UDP Client/Server socket flow**



1. **Socket System calls**
   1. **socket()(use by TCP/UDP both)**

#include<sys/socket.h>

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

* socket() function is used to create an endpoint for communication between two process over a network.
* It allow for sending and receiving the data over network by access different underlying communication protocol like tcp/ip,ipx/spx etc.
* Commonly used by client/server for communication
* Socket function consists of 3 parameters as follows
* First parameter **“family”:**
  + First parameter basically is a family of communication protocol like Internet domain(IPV4/IPV6), UNIX domain etc.
  + Types of family(Domain)
    - AF\_INET: for IPV4 protocol
    - AF\_INET6: for IPV6 protocol
    - AF\_LOCAL/AF\_UNIX: Unix domain protocol
    - AF\_ROUTE: Routing sockets (Accessing route kernel routing table
    - AF\_KEY: Key socket(Cryptographic activities
* Second parameter **“type”**
  + SOCK\_STREAM: streaming socket used by TCP/SCTP
  + SOCK\_DGRAM: datagram socket used by UDP
  + SOCK\_SEQPACKET: used by SCTP
  + SOCK\_RAW: for access raw network data
* Third parameter **“protocol”**
* IPPROTO\_TCP: transport layer TCP protocol
* IPPROTO\_UDP: transport layer UDP protocol
* IPPROTO\_SCTP: transport layer SCTP protocol
* We can set this value as zero to select protocol based on second parameter.
* Return value:
  + Error -1
  + Return new socket descriptor on successful creation of endpoint.
* This call is used by both clients/server for creating endpoint(socket)
* Example in c:

#include<sys/socket.h>

Int fd=socket(AF\_INET,SOCK\_STREAM,IPPROTO\_TCP);

* 1. **bind()(use by TCP/UDP both)**
     + bind function is used to assign the local protocol address to socket.
     + In case of internet domain, the protocol address can be either 32 bits IPV4 address or IPV6 address with 16 TCP/UDP port number.
     + Actually bind() map between IP,PORT and endpoint for server so that kernel knows which interface address and well know port the process is listening for incoming connection from the client.

#include<sys/socket.h>

Int bind(int fd,const struct sockaddr\*myaddr,socklen\_t addrlen);

* + - First parameter **“fd”**
      1. Endpoint created through socket system call
    - Second parameter **“struct sockaddr\*”**
      1. It is address structure of type \* which will supposed to send IP address,TCP port number and address family
      2. This parameter should be passed as point to structure of type sockaddr
    - Third len “addrlen”
      1. The length of the structure that we are passing to kernel for mapping purpose between communication protocol and endpoint
    - Bind function is called by server to map the server endpoint with specific interface address and port number so that client can contact to the application
    - If bind will not use than kernel will allocate ephemeral port which is difficult for server to contact as we don’t know the port until application run and changes time to time.
    - Client normally do not use bind it let kernel to allocate the port for it.
    - Return value
      1. Success 0
      2. Failed -1
    - Example with c

#include<sys/socket.h>

Struct sockaddr\_in server;

Server.sin\_family=AF\_INET;

Server.sin\_port=htons(PORT);

Server.sin\_addr.s\_addr=INADDR\_ANY; //or can use inte\_addr(IP)

Int v=bind(fd,(struct sockaddr\_in\*)&server,sizeof(server));

* 1. **Listen()(use by TCP server only)**
     + This function is used by server
     + By default when socket is created as active socket but when listen is used it set the socket to passive indicating that it should accept the connection request from the client.
     + Listen will set the socket from CLOSED to LISTEN state of TCP and generally used in case of TCP socket.
     + Once client execute connect, and server after accept TCP 3-way handshake will be completed and will set TCP state to ESTABLISHED state and data transfer can be performed.
     + This call is executed after socket() and bind() function in case of TCP server socket.
     + Successful return 0 and failed as -1
     + After calling listen TCP Server use the accept() function to accept incoming client connection request and creating a new socket connection for each client.
     + The first parameter in listen is socket descriptor return after successful socket() system call
     + The second parameter is the backlog which is use for handling the connection request from client.
     + The backlog is divided into two part
       1. Incomplete queue
          1. This is the client request of the client where client SYN packet has been received and server has send client acknowledgement of it SYN and Server SYN packet and waiting for its acknowledgement.
       2. Completed queue
          1. If all TCP 3-way handshake has been completed and in established state and ready for data transfer.
       3. Backlog of 5 means it can handle total of 8 connection request. Logic is 1.5 \* backlog
       4. Never set backlog value as 0 connection will be never accepted
       5. Backlog is cumulative value of incomplete and complete connection of client request. Maximum number of connection allowed in the queue.
       6. Some system might not accept high value backlog can throw the error.
       7. Example in c

Fd=socket(…)

Bind(fd,…);

int listen(int fd,int backlog);

int confd=listen(fd,5);

1. **Connect()(use by TCP client only)**
   1. Call by client and used to establish connection with remote server
   2. This call will initiate TCP-3 way hand shake and completion performed by server after accept() call.
   3. Connect() function is blocking in call by default until connection is established or failed
   4. Once connection is established client can send/receive over the connection
   5. We can use select() system call to make it to non-blocking call
   6. Header file#include<sys/socket.h>
   7. Function prototype

Int connect(int sockfd,const struct sockaddr\*,socklen\_t addrlen);

* + - Sockfd: socket descriptor return by socket() system call to represent endpoint. It represent the local socket you want to connect
    - Sockaddr\*: pointer to socket address structure , this address contain the remote server IPV4 address and port number that we want to connect to.
    - Addrlen: the size of address structure sockaddr\_in eg sizeof(struct sockaddr\_in)
  1. Under successful execution of this function
     + 0 for success and connection will get established ready for transfer of data
     + -1 an error
  2. Pseudo code

fd=socket(…);

struct sockaddr\_in server;

//setting address to structure

server.sin\_faily=AF\_INET;

server.sin\_port=htons(9999);

server.sin\_addr.s\_addr=inet\_addr(“127.0.0.1”)//loop back address

if(connect(fd,(strcut sockaddr\*)&server,sizeof(server))==-1){print(“error”);}

1. **Accept() (used by TCP server)**
   1. This function is used by server to accept the incoming connection requested by client after the listen call.
   2. This call in conjunction with connect() completes the TCP 3 way handshake for connection establishment.
   3. Once both the call completed successfully we say client/server is connected and can start with data transfer back and forth between them.
   4. This call is only used by TCP server socket
   5. Pseudo code in c

Int accept(int sockfd,strcut sockaddr\*addr,socklent\_t \*addrlen);

* Fd: from the socket system call rest you know
* Addr: this is sockaddr structure as pointer which will fill with requesting client information like IP, PORT and Family information. It is pass as pointer to strcucture
* Addrlen: it is pointer to address structure len this is value result argument. Under successful call it will return the actual length size of the client address structure.
* Return value
  + Success new socket descriptor will be return which will represent the specific connection between client and server and used to send and receive data over socket.
  + -1 in error

1. **Input/output call for TCP & UDP socket**
   1. This input output model is used for sending and receiving the data to and from the socket.
   2. This call has separate function for reading data from kernel buffer and sending data to kernel buffer for transmission
   3. Based on socket and type and requirement this call can differ.
      * TCP I/O call
      * Read() system call
        1. This system call is used to read the data from the kernel buffer to application buffer.
        2. Pseudo code in c
        3. Ssize\_t read(int fd,void\*buff,size\_t count);
           1. Fd: file descriptor from which you want to read data. This can be as follows:
           2. Regular file, socket, pipe or another file descriptor.
           3. Buf: a pointer to the buffer where the data read will be stored
           4. Count: maximum number of bytes of data to read from the descriptor
           5. Return value

On success: number of bytes actually read. This value can be less than count if available data is less

On failure: -1

* + - * 1. Code in c
        2. Ssize\_t readbytes=read(fd,buffer,100);
        3. It will reads up to the count and if end of file is reached return 0
    - Write() system call
      1. This system call is used to write data to kernel buffer for sending information at network
      2. Pseudo code
         1. Ssize\_t write(int fd,const void\*buff,size\_t count);
         2. Fd: file descriptor to write
         3. Buff: holds the information you want to write
         4. Count: number of bytes you want to write from the buffer.
      3. Return value:
         1. Success: Number of bytes actually written, number may be less than count if less data is supposed to write
         2. Failure:-1
      4. Code in c
         1. Ssize\_t byteswritten=write(fd,sendbuff,strlen(message));
    - UDP I/O call
      1. Sendto()/recvfrom() system call are used to send and receive message over datagram based socket like (UDP)
      2. This is basically used for transferring message with connectionless protocol like UDP
      3. No connection has to establish with this function
      4. With this operation we supply both sender and receiver address for each individual message.
    - Sendto()
      1. Ssize\_t sendto(int sockfd,const void\*buf,size\_t len,int flags,const struct socaddr\*dest\_addr,socklen\_t addrlent);
         1. Sockfd: socket descriptor from socket() return
         2. Buf: A pointer to the buffer containing data you want to send
         3. Len: the length of data in bytes you send from buffer
         4. Flags: use for special operation and usually to handle OOB data eg

MSG\_DONTWAIT: non-blocking send

MSGNOSIGNL: prevent sending signal from remote peer if disconnected.

* + - * 1. Return value:

Success: number of bytes send

Failure: -1

* + - * 1. Example in c

Ssize\_t r=sendto(fd,buf,strlen(buf),0,(struct sockaddr\*)&server,sizeof(server));

* + - Recvfrom()
      1. Ssize\_t recvfrom(int sockfd,void\*buff,size\_t len,int flags,struct sockaddr\*src\_addr,socklen\_t \*addrlen);
         1. Fd: socket descriptor after socket() return
         2. Buf: A pointer to buffer where you want to store receive data
         3. Len: maximum number of bytes to receive
         4. Flags: handling OOB data normally used 0

MSG\_DONTWAIT: non-blocking receive

* + - * 1. Src\_addr: the structure where the source address will be store. It will contain ip and port address of sender
        2. Addrlen: pointer to the type addr structure which will actually give the size of the source address.
        3. Return value

Success: number of bytes received which may be less than the len if the sender sends few bytes

Failure: -1

* + - 1. Pseudo c code
         1. Int len=sizeof(client);
         2. Ssize\_t bytesrecv=recvfrom(fd,buffer,sizeof(buffer),(struct sockaddr\*)&client,&len);
  1. **Close() & shutdown()**
     + Close()
       1. This function is used to close the a socket when communication is complete in order to free the occupied resources.
       2. Int close(int sockfd);
       3. Sockfd: socket descriptor to close
       4. Return value:
          1. Success:0
          2. Fail=-1
          3. Close(fd);
     + Shutdown()
       1. Is use to disable further send or receive on a socket for the effective ending of socket in well controlled manner
       2. Int shutdown(int sockfd,int how);
          1. Sockfd: socket descriptor after socket() return
          2. How: specify what to disable

SHUT\_RD: disable reading

SHUT\_WR: disable writing

SHUT\_RDWR: disable both reading and writing

* + - * 1. Return value:

Success=0

Fail: -1

Shutdown(fd,SHUT\_WR);

1. Descriptor passing
   1. It is the technique of passing an open file between the process with in UNIX system which has couple of ways
      * Between related process
        1. Passing from parent to child by using fork or exec function
      * Between unrelated process (Stream/Datagram any can be used)
        1. Server crate UNIX domain socket and bind pathname to it allowing client to connect it
        2. Here client will request descriptor to open and server can pass back open file descriptor to client.
        3. To open descriptor any UNIX domain descriptor creating function can be used e.g:
           1. Open(),pipe(),mkfifo(),socket() or accept().
        4. Sending process creates a msghdr structure containing descriptor to pass. As per POSIX descriptor should be send as ancillary data(It is meta data is a mechanism to send that is not actual data but use for handling communication like socket)
        5. For sending and receiving descriptor two function are used
           1. Sendmsg()
           2. Recvmsg()
           3. Prototype in C.

#include <sys/socket.h>

ssize\_t sendmsg(int sockfd, const struct msghdr \*msg, int flags);

ssize\_t recvmsg(int sockfd, struct msghdr \*msg, int flags);

* + - * 1. Structure msghdr

void \*msg\_name;

socklen\_t msg\_namelen;

struct iovec \*msg\_iov;

size\_t msg\_iovlen;

void \*msg\_control;

size\_t msg\_controllen;

int msg\_flags;

};

Msg\_name,msg\_namelen:Basically used for UDP to specify destination address

Msg\_iov,msg\_iovlen:describe actual data to be sent

Msg\_control,msg\_controlen: describe ancillary data.

* 1. C-Prototype example

#include <sys/socket.h>

#include <unistd.h>

#include <string.h>

#include <stdio.h>

void send\_fd(int socket, int fd\_to\_send) {

struct msghdr msg = {0}; //short way of initializing structure to zero

struct cmsghdr \*cmsg;

char buf[CMSG\_SPACE(sizeof(fd\_to\_send))];//assigning amount of memory need for anicillary data(OOB).

memset(buf, 0, sizeof(buf));

struct iovec io = {.iov\_base = "FD", .iov\_len = 2};//initializing iovec, with len 2 bytes

msg.msg\_iov = &io;//assign iov

msg.msg\_iovlen = 1; // len of io

msg.msg\_control = buf; //ancillary data to send

msg.msg\_controllen = sizeof(buf);//len of ancillary data

cmsg = CMSG\_FIRSTHDR(&msg); //this function is used to check if control message is available return NULL or cmsghdr structure

cmsg->cmsg\_level = SOL\_SOCKET;// define socket level option or control msg

cmsg->cmsg\_type = SCM\_RIGHTS; //allow to send open file descriptor

cmsg->cmsg\_len = CMSG\_LEN(sizeof(fd\_to\_send));//calculate size of ancillary data

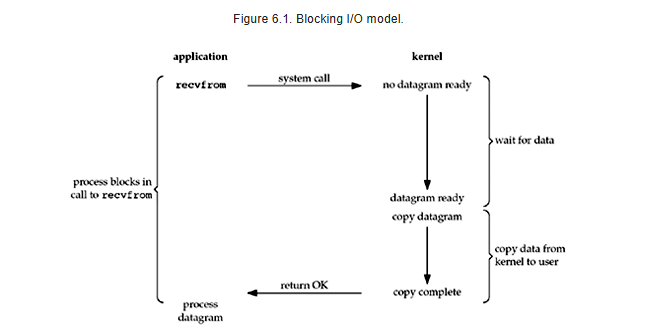
memcpy(CMSG\_DATA(cmsg), &fd\_to\_send, sizeof(fd\_to\_send));//copying

if (sendmsg(socket, &msg, 0) < 0){error..}

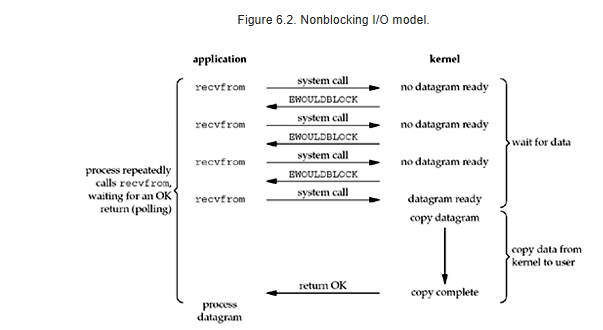
}//end function

CMSG\_DATA(cmsg): this macro retrieves a pointer to the data portions of control message(cmsg).this pointer actually points where is actual ancillary data is stored.

1. I/O models
   1. In network programming(Socket) I/O models refers to how Input and output operation i.e. reading and writing to and from the socket is performed.
   2. Actually this I/O models allows us the different technique to deals with socket operation based on our requirement and can design the system accordingly.
   3. The I/O models has direct impact on the system design and performances and sophisticated implementation.
   4. I/O operation actually refers to reading and writing data to and from the memory
   5. Blocking I/O model



* + - Basic model where the calling process will be blocked until read returns
    - By default, all the socket is block in nature.
    - Explaining Figure
      1. Process call recevfrom() as system call and process will be not return until datagram is ready.
      2. Once datagram ready it is copied from kernel to application buffer and return ok to process
      3. There process will be blocked during the whole life time of recvfrom call.
  1. Non-Blocking, I/O model
     + Now let try to explain diagram same as above.



* + - UNIX function for setting socket to non-blocking mode
      1. #include<fcntl.h>

Fcntl()

int flags = fcntl(fd, F\_GETFL, 0); //reading current fd flag

fcntl(fd, F\_SETFL, flags | O\_NONBLOCK);//setting to nonblk

pipe is doing bitwise or and ensuring that the non-blocking flag is added to exising flag

* + - 1. Ioctl()

#include<sys/ioctl.h>

Int mode=1,val;//1=nonblocking 0=blocking

Int confd;

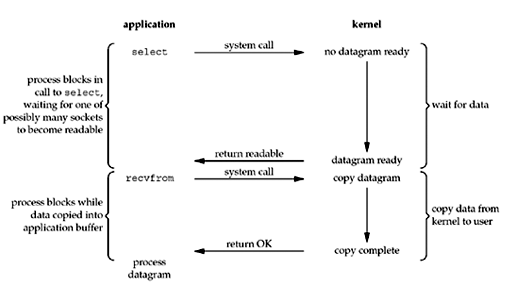
Confd=accept(fd,…);

val=ioctl(confd,FIONBIO,(char\*)&mode);//FIONBIO is req flag

if(val<0){print(“error”);}

**Note: Fcntl is more specific to file related operation to handle where ioctl is device specific.**

* 1. Multiplexing I/O model



* + - Select() system call
    - #include<sys/select.h>
    - #include<sys/time.h>
    - Int select(int maxfdp1,fd\_set \*readset,fd\_set \*writeset,fd\_set \*exceptset,const struct timeval \*timeout);
    - Return 0 if descriptor is ready else -1

Struct timeval{

long tv\_sec;

long tv\_usec;//microset

};

* + - Fd\_set() macro
      1. FD\_ZERO(fd\_set \*set);//clear the descriptor set
      2. FD\_SET(int fd,fd\_set \*set);//adding file descriptor for monitoring and interested on particular event to track

FD\_SET(sock, &readfds); // Adds 'sock' to the readfds set

FD\_CLR(sock, &readfds); // Removes 'sock' from the readfds set

if (FD\_ISSET(sock, &readfds)) { // The socket 'sock' is ready for reading }

* + - C proto type of select

Int confd, max\_sd;

Fd\_set readfds;

int client\_socket[MAX\_CLIENTS] = {0};//for handling connection

Socket()

Bind()

Listen()

While(TRUE)

{

//clearing socket set

FD\_ZERO(&readfds);

FD\_SET(server\_fd, &readfds);//adding to structure readfds

// Wait for an activity

select(max\_sd + 1, &readfds, NULL, NULL, NULL);

//adding client connection request to connection array

if (FD\_ISSET(server\_fd, &readfds)) {

client\_socket[i]=accept();//keeping inside array

if (FD\_ISSET(sd, &readfds)) {

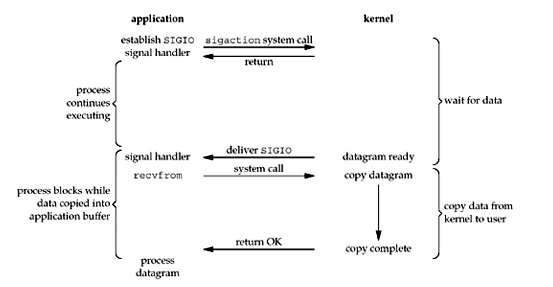
read()//reading data descriptor ready for reading operation

}

}

}

* 1. Signal driven I/O model



* + - 1. Enable socket for signal-driven I/o and install signal handler using sigaction system call.
      2. Return from sigaction call is immediate process will not block
      3. When datagram is ready SIGIO signal is generated by signal handler to calling process,they continue with data reading with block call.
      4. C prototype

#include<fcntl.h>//for setting sock operation

#include<signal.h>//for signal handler

#include<unistd.h>//getpid()

Void my\_signalHandler(int sig)

{

(void)sig;//suppressing unused variable warning

Printf(“Your Signal will be trapped here and ready “);

}

Int main(…)

{

//setting up SIGO signal handler

Struct sigaction sa;

sa.sa\_handler = sigio\_handler

sa.sa\_flags = 0;

sigemptyset(&sa.sa\_mask);

**if (sigaction(SIGIO, &sa, NULL) == -1) {error}**

//now followed by all the socket call

//now set the process as the owner for socket for SIGIO

fcntl(server\_fd, F\_SETOWN, **getpid()**) == -1) {error;}

//enabling ascynchronous I/O notification

int flags = fcntl(server\_fd, F\_GETFL, 0);

if (fcntl(server\_fd, F\_SETFL, flags | O\_ASYNC) == -1) {e;}

//no error read data

While()

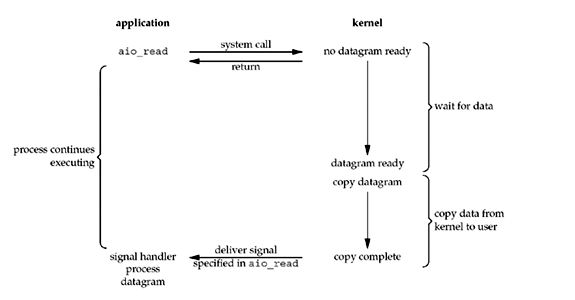
{

Connection accept();

Read/write

}

* 1. Asynchronous I/O model



* + - Its part of POSIX asynchronous read
    - Allow non-blocking file or socket I/O operation
    - Here I/O operation can continue executing without waiting for I/O operation to complete
    - On completion of operation signal is deliver and can handle read and write operation separately
    - Exaple in C proto

#include<aio.h>

struct aiocb aio\_cb;

char buffer[BUFFER\_SIZE];

memset(&aio\_cb, 0, sizeof(struct aiocb));//you know tell me.

//buffer initialization for aio

aio\_cb.aio\_fildes = fd;//setting descriptor

aio\_cb.aio\_buf = buffer;//buffer to store the read data

aio\_cb.aio\_nbytes = BUFFER\_SIZE;//no of bytes to read

aio\_cb.aio\_offset = 0;//offset in the file to start reading

//instating the asynchronous read

if (aio\_read(&aio\_cb) == -1) {error;}

//aio\_read ok now waiting for aio to complete

while (aio\_error(&aio\_cb) == EINPROGRESS) {

print(“Reading in progress…./n”);

sleep(1)//giving breathe space to process to check what is going on

}

//checking final status of read operation

int err = aio\_error(&aio\_cb);//getting error satus

if(err!=0){error exit;}

else

//read the data

ssize\_t bytes\_read = aio\_return(&aio\_cb);

print(“Reading complete asyn%d”,bytes\_read);

//showing read data

Print(“Data read%s”,buffer);//it is copied to app buffer

1. Socket options
   * + Getsockopt() and setsockopt() are system call in socket programming are used to set the options associated with socket.
     + Return value is 0 for success and -1 for failure
     + Header file #include<sys/typesh> and #include<sys/socket.h>
     + C proto type for socket option
   1. Getsockopt()
      * Get the socket option set for the protocol or socket

int getsockopt(int sockfd, int level, int optname, void \*optval, socklen\_t \*optlen);

**sockfd**: socket descriptor to change behavior of socket

**level**: protocol level where you want to perform operation eg:

SOL\_SOCKET: at socket level you want to perform operation

IPPROTO\_TCP: tcp level

IPPROTO\_IP: at ip level

Optname: behavior you want to set or get eg:

SO\_RCVBUFF: checking or setting receive buffer size

SO\_REUSEADDR: when socket is in time\_wait state without waiting to finish it

You can reuse address.

**Optval**: A pointer to the value to be set for the option

**Optlen**: len of optval

* 1. Setsockopt()
     + Set the behavior you want for the socket or protocol

int setsockopt(int sockfd, int level, int optname, const void \*optval, socklen\_t optlen);//same above

* 1. Proto type code

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netinet/tcp.h>

int main(int arg, char\*\* argv)

{

int fd=socket(AF\_INET,SOCK\_STREAM,0);

int optval=1;//1 means enable option

int recvBuffSize=0;

int tcpkeepen;

socklen\_t tcplen=sizeof(tcpkeepen);

socklen\_t buflen=sizeof(recvBuffSize);

//getting socket option

if(getsockopt(fd,SOL\_SOCKET,SO\_RCVBUF,&recvBuffSize,&buflen)<0)

{

printf("Error on receiving size\n");

}

printf("Current Buffer size is[%d]\n",recvBuffSize);

if(getsockopt(fd,SOL\_SOCKET,SO\_KEEPALIVE,&tcpkeepen,&tcplen)<0)

;

printf("TCP flag value[%d]\n",tcpkeepen);

close(fd);

return 0;

}

//further reference with socketopt

int idle = 60; // Seconds of idle time before sending keepalive probes

int interval = 10; // Interval between keepalive probes

int cnt = 5; // Number of keepalive probes to send before giving up

setsockopt(sock, IPPROTO\_TCP, TCP\_KEEPIDLE, &idle, sizeof(idle));

setsockopt(sock, IPPROTO\_TCP, TCP\_KEEPINTVL, &interval, sizeof(interval));

setsockopt(sock, IPPROTO\_TCP, TCP\_KEEPCNT, &cnt, sizeof(cnt));

* 1. Fcntl()
     + More specific for file operation
     + int fcntl(int fd, int cmd, ...);
     + fd: file descriptor
     + cmd: command specifying the operation to perform eg locking,get/set flags
     + …:optional depending on the command
     + C prototype

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#include <errno.h>

#include<time.h>

int main(int arg, char\*\*argv[])

{

struct timespec req;

req.tv\_sec =60; // 1 second

req.tv\_nsec = 500000000;

int fd=open("c.txt",O\_WRONLY | O\_CREAT, 0777);

if(fd<0)

{

printf("Error on operning file [%d]\n",errno);

}

struct flock lock;

lock.l\_type = F\_WRLCK; // Write lock=F\_WRLCK,read lock=F\_RDLCK

lock.l\_whence = SEEK\_SET;//lock from start of file

lock.l\_start = 0;//start from begining of file

lock.l\_len = 0; // Lock the entire file

lock.l\_pid=getpid();//setting lock owner

//try to seting lock

if(fcntl(fd,F\_SETLK,&lock)<0)

printf("Error creating Lock[%d]\n",errno);

else

printf("Lock acquire successfully\n");

nanosleep(&req, NULL);

//now unlocking fine

lock.l\_type=F\_UNLCK;

fcntl(fd,F\_SETLK,&lock);

close(fd);

return 0;

}

* 1. Ioctl()
     + More specific to device control and operation
     + It allow sending custom command to device driver to control hardware or software behavior that will not covered by standard system call
     + int ioctl(int fd, unsigned long request, ...);
     + fd: file descriptor you want to perform operation
     + request:A device dependent request code that will tell the specific operation to perfrom
     + …:optional flag dependent on command type you are requesting
     + C prototype

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ioctl.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <sys/socket.h>

#include<net/if.h>//for ifreq struct for ip address

int main(int arg, char\*\*argv)

{

int fd;

struct ifreq ifr;

fd=socket(AF\_INET,SOCK\_STREAM,0);

strncpy(ifr.ifr\_name,"en3",IFNAMSIZ);//filling structure

if(ioctl(fd,SIOCGIFADDR,&ifr)<0)

;

struct sockaddr\_in \*ipaddr=(struct sockaddr\_in\*)&ifr.ifr\_addr;//type cast

printf("IP address retreive from interface[%s]\n",inet\_ntoa(ipaddr->sin\_addr))

return 0;

}

* + - Note: SIOCGIFADDR: is the request code used to get ip addressed assign to specific interface in our case en3.
    - CDROMEJECT: ejecting cdrom
    - CDROMCLOSETRAY:used to close cdrom
    - command eg:
    - const char \*device = "/dev/cdrom";
    - fd = open(device, O\_RDONLY | O\_NONBLOCK);
    - if (ioctl(fd, CDROMEJECT) < 0) you have to open cdrom before executing it.

1. **Daemon process**
   1. A daemon is a background process that runs on a computer without direct user intervention
   2. It basically performs below task
      * 1. Initiated at the system startup and runs continuously until system is shuts down
        2. Provide services like managing hardware devices, handling network request, running scheduled task or logging system information.
        3. Daemon does not attach with terminal session meaning that when terminal is exit it will not get logout
        4. Every Daemon process has suffix d attached to it e.g sshd, httpd etc.
   3. Example of common daemons process
      * 1. Sshd: managing incoming SSH connections
        2. Httpd: serves web pages for https request
        3. Cron: cron daemon handling cron job
        4. Syslogd: syslog daemon handling messaging logging
        5. Ntpd: network time protocol daemon sync time
   4. Working principal of Daemon
      * 1. Initialization: daemons are launched by system init process eg init,system or launchd and start all daemons from the configuration file from /etc/rc.d🡪 entry will be here for daemon and start during system boots
        2. Detach from terminal: parent executes for() to provide the services and terminate itself letting child to continue.
        3. Run in background: daemons continue to run in background to handle task like waiting for incoming connection or perform specified action
        4. Logging: daemons log the log like error, information debug into the location /var/log/syslog or based of OS location can be changed or can set custom log file location
        5. Termination: Basically daemon runs till system running however administrator can force stop the daemon by executing command like systemctl or system.
   5. Advantages of Daemon
      * 1. Contentious operation: background running without user interaction
        2. Automation: automate task like logging, monitoring , scheduling and maintain services
        3. Reliability: Daemon ensure that key service are always available
        4. Resources efficiency: Consume minimum resources like memory and processor (performance optimization)
        5. Scalability and fault tolerance: Daemons are built to handle multiple clients, recover from failure and can scaled if needed
2. Syslog Daemon
   1. Syslog Daemon is the background process responsible performing below task
      * 1. Logging of system message
        2. Logging System events
        3. Logging user’s application logs
   2. Syslog daemons collects process and collect log messages from various resources like kernel, system services and user’s applications
   3. Functions of Syslog Daemon
      * 1. Collecting logs: Syslog daemon will be listening for collecting the log from different source like kernel, user application, network etc. it collect that information and send to specific file for logging or to remote log server if configured.
        2. Log prioritizing and filtering: it perform based on below severity

Emergency(highest level)

Alert

Critical

Error

Warning

Notice

Information

Debug(lowest priority)

* + - 1. Storing of log to define location
         1. /var/log/syslog or /var/log/message (general message)
         2. /var/log/auth.log(authentication log)
         3. /var/log/kern.log(kernel logs)
         4. /var/log/cron(cron job logs)
      2. Log rotation: old log archiving facility
      3. Common syslog Daemons
         1. Syslogd: traditional syslog performs

Listening for log

Writes them to log file specified

And can forward to remote server

Config location: /etc/syslog.conf

* + - 1. Rsyslog: an enhanced version

More advance filtering and remote logging by TLS and integration with database

Default for many UNIX or LINUX system

Config loc: /etc/rsyslog.conf

* + - 1. Many other are their like journal, syslog-ng(FYR explore by you)
      2. Syslog example:
      3. Jan 1 17:06:24 hostname phantom: [1234456.68900] [INFO] Network interface eth0 up
      4. Jan 1 17:06:24 hostname kernal: [1234456.68900] [INFO] Network interface eth0 up
         1. Red: timestamp when event occurred
         2. Black: hostname generating message
         3. Blue: A timestamp for system uptime
         4. Yellow: Log level or severity
         5. Green: actual message logged

1. Syslog function
   1. Syslog() is a function in c used to send log message to the system log and handle by syslog daemon
   2. It allows application to log message of different severity level(log,info,degug,error…) and can routed local or remote server
   3. Header file #include<syslog.h>
   4. Function prototype
   5. Void syslog (int priority,const char \*format,…);
   6. Priority: it is the combination of two-part log facility and log level(severity)
   7. Log facility: it defines the types of program or subsystem generating the log eg:
      * 1. LOG\_USER: user level message
        2. LOG\_KERN: kernel message
        3. LOG\_MAIL: mail system message
        4. LOG\_DAEMON: daemon process message
        5. LOG\_AUTH: Security server message doing authorization eg user verification
        6. LOG\_SYSLOG: internal syslog message
        7. LOG\_LOCAL0 to LOG\_LOCAL7: custom facilities for application specific message
   8. Log Levels
      * 1. LOG\_EMERG: emergency message
        2. LOG\_ALERT: message where action has to be taken immediately
        3. LOG\_CRIT: critical condition message
        4. LOG\_ERR: error message (exception message)
        5. LOG\_WARNING: waring message
        6. LOG\_NOTICE: Normal but significant conditions
        7. LOG\_INFO: general informative message
        8. LOG\_DEBUG: Debugging message
   9. Format: it defines the format of the log how it will be organized as string to dump into the log file.
   10. …: this is a variable list of argument how you want to format your message according to the format string.
   11. Eg open**(“ApplicationName”,** LOG\_PID | LOG\_CONS, LOG\_USER);
   12. Syslog(LOG\_INFO,”This is pushing of log message”);
   13. Closelog()//closing connection after logging message.
   14. **C program code**

#include<stdio.h>  
#include<syslog.h>//syslog  
int main(int arg,char\*\* argv[])  
{

Int errno=-1;  
        printf("Syslog Function Testing !!!\n");  
        openlog("syslog",LOG\_PID | LOG\_CONS, LOG\_USER);  
        syslog(LOG\_INFO,"Logging info level message");

If(errno<0)

{  
        syslog(LOG\_ERR,"Logging error message",errno);

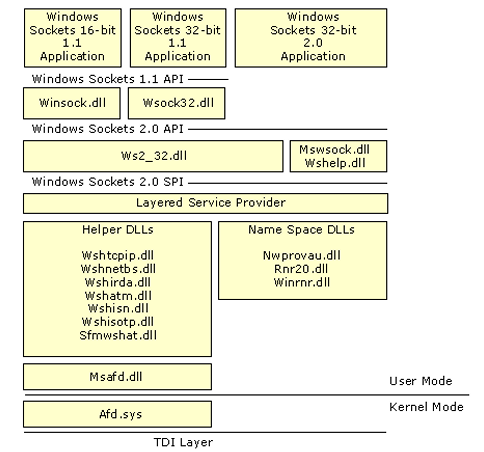
}  
        closelog();  
        return 0;  
}

**----------------END of Chapter-22 Hours------------------------**

**3.Winsock Programming**

3.1 Introduction to Winsock architecture

* Winsock in short is window socket, a programming interface and a set of network services used for creating network based application.
* Basically it was design to implement network based application which can communicate across the internet by accessing underlying core communication protocol like TCP/UDP/IP.
* It facilitates to create client/server paradigm application for sending/receiving message across the network like web browsing, file transfer, chat application etc.
* It supports both connectionless and connectionless communication (UDP, TCP)
* Winsock is primarily design for TCP/IP communication stack where this does not fit in case of UNIX where it supports both UNIX and INTERNET family for communication.
* Winsock play and intermediate system between user application and underlying communication protocols.
* **Winsock Architecture 2.0**



* Winsock 2.0 is a Windows Open System Architecture(**WOSA**)a compliant interface allow backend or front end application service to communicate
* **Interface included by Winsock 2.0**
  + Winsock 1.1 API: API for access underlying protocol for 32/16 bits’ application
  + Winsock 2.0 API: It extend 1.1 to provide additional services other than TCP/IP like
    - Netware
    - Apple Talk
    - Support real time multimedia communication including QOS
    - DNS, Novel directory service is supported to access multiple name space
    - Protocol independent multipoint and multicast.
  + Winsock 2.0 TSP (Transport service provider)
    - TSP is software component responsible for
      * Network connection establishment, management, termination
      * TCP and UDP are basically used for communication between process.
      * TSP enable Winsock API to work with various network protocols and handles the details of those protocol for network communication
      * There are several TSP for every protocol that takes part in communication
      * E.g. let create an end point how the flow looks like
        + Application call socket (AF, SOCK\_STREAM, IPPROTO\_TCP);
        + App()->Winsock1.1/Winsock2->TSP-TCP will be called since we are trying to create TCP socket. Which means transport service provider for TCP will be called same way for other.
      * TSP is responsible for creating meta data from protocol eg packet composing(construction), error checking and network address.
  + Layered Service Providers
    - Layer service provider is an extension of TSP sits between Application and TSP to add extra functionality like encryption, compression, monitoring.
    - Use for modifying transport service thereby offer more advance feature without modifying underlying protocol (think as wrapper function)
  + Winsock 2.0 Name resolution provider
    - Responsible for resolving domain name e.g. [www.cosmos.com.np](http://www.cosmos.com.np) into ip address
    - This service that works with Winsock API to convert human readable domain name to network address.
* Winsock contain multiple DLLs for completing particular task listed below

|  |  |
| --- | --- |
| Winsock DLLs | Description |
| Winsock.dll | 16 bits Winsock 1.1 |
| Winsock32.dll | 32 bits Winsock 1.1 |
| Ws2\_32.dll | Main Winsock dll for version 2 |
| MsWsock.dll | Extension dll of Winsock and provides service that not part of Winsock |
| Ws2help.dll | Platform specific utilities. Supply operating system specific code which is not part of Winsock. This component assist Winsock. Transport protocol such as TCP, ATM and IrDA have DLLs that supply provide necessary program code to support winsock |
| Wshtcpip.dll | Helper for TCP |
| Wshnetbs.dll | Helper for NetBIOS |
| Wshirda.dll | Helper for IrDA |
| Wshatm.dll | Helper for ATM |
| Wshisn.dll | Helper for Novel Netware |
| Wshisotp.dll | Helper for OSI transport |
| Sfmwshat.dll | Helper for Macintosh |
| Nwprovau.dll | Name resolution provider for IPX |
| Rnr20.dll | Main name resolution |
| Winrnr.dll | LDAP name resolution |
| Msafd.dll | Winsock Interface to Kernel |
| Afd.sys | Winsock interface for TDI transport protocol(Transport Driver Interface) |

* + Winsock DLL
* DLL in short is Dynamic Link Library which is a file that contains code and data that can be used by multiple application program simultaneously.
* DLL allow modular programming by providing reusable functions, procedures and resources that can be shared across different application
* Why DLL?
  + Shared Libraries: Multiple program can load and use same DLL thereby reducing memory usage and redundancy
  + Modularization: Can divide the huge task into sub-task e.g. for handling network connection can use net.dll for database connection db.dll etc.
  + Efficiency: Since DLL are load into when only needed they help to conserve system resources.
  + Dynamic Linking: DLL files are link to application at runtime rather than compile time however static library extension .lib is in compile time linking
  + Some Example of DLL
    - Kernel32.dll: for managing process, memory, files
    - User32.dll: Handle user interface operation like windows button, mouse events
    - Gdi32.dll: This deals with drawing operation in windows
* Benefits of DLL
  + Code Reusability: Shared between program
  + Reduced Size: All command function is centralized in single DLL and all application needing that can be managed by single dll
  + Easier Update: Any update can be automatically done at DLL without recompiling or redistributing entire application (Off course you have to compile DLL if any update made and can replace only the specific DLL)
* Issues with DLL
  + Version Conflicts: if different application requires different version of same dll than it is trouble giver.
  + Missing DLLs: if required DLL are missing or corrupted then your application fail to run
* **Types of DLL in windows**
  + **Static DLLs**
    - It is the process of including all of the code from library directly into the application executable file at compile time and known as static library
    - Extension .lib
  + Characteristics
    - Compile time binding: all code bound to application during the time of application compilation
    - Application Bundle with code: In static library the code is copied to application during compile time making the application self-contained.
    - No runtime dependencies: There no runtime dependencies in static DLL because all the required library or code included into executable file during compile time
  + Drawbacks
    - Larger Executable: Application **become larger since it includes all the necessary code in executable** application
    - No Sharing: Here sharing library is not possible because each program will get copy for library separately
    - Updates: If library is updated the application must be recompiled.
  + **Dynamic DLLs**
    - It refers to loading and lining the library at run time rather than compile time
    - File extension of dynamic link library .dll
    - It is **loaded into the memory** when application needed it
  + Characteristics
    - Runtime Binding: The application load DLL at runtime (Explicitly or implicitly) and bind to function it needs
      * Explicit: Program itself load the DLL when needed(Manually)
      * Implicit: It refers to automatic loading of DLL when program starts which means application is preconfigured to load the DLL.
    - External Dependencies: The application depends on the presence of the DLL on the System at runtime, if it not their program fails to load
    - Shared libraries: only instance of library is load into memory and can be shared by multiple application thereby saving memory and disk space
    - Runtime loading: the application can choose when and how they want to load the DLL using function like LoadLibrary() and GetProcAddress()
  + Benefits
    - Smaller Executable: Because code is not embedded into executable
    - Code Reusability: DLLs can be shared among multiple application
    - Easier update: if we update any dll only dll need to recompiled
    - On demand loading: We can choose to load specific code only from DLL.
  + Drawbacks
    - Dependency: Checks for run time dependency not found system fails to start
    - DLL problem: if multiple application uses different version of DLL which leads to conflict known as DLL hell.
    - Slight overhead: Compare to static library during runtime it introduce small amount of computing overhead
* **Comparison between Static library (.lib) and Dynamic library(.dll)**

|  |  |  |
| --- | --- | --- |
| Features | Static DLL | Dynamic DLL |
| Library linking | Compile time | Run time |
| Size of exe | Larger(include all needed code at compile time from library) | Smaller(does not include DLL code) |
| Runtime dependency | None | Yes requires DLL at run time |
| Code sharing | No each app will have their own copy | Yes DLL is shared to multiple application |
| Update | Requires application re-compilation | Application does not need recompilation however if DLL is updated then it Is compiled |
| Loading mechanism | Code is embedded into exe | Can be loaded explicitly(by application or implicitly(at starting) |
| Extension | .lib | .dll |

* + **window socket and Blocking I/O**
* Windows socket API is a programming interface that allow user application to access the communication protocol like TCP/IP, IPX/SPX, AppleTalk etc.
* Winsock is a window specific implementation for design network based application
* Winsock allow to send/receive message across the network or within same system.
* Winsock basic operation for socket (**you know below topic**)
  + Socket creation
  + Binding
  + Listening and Accepting connection
  + Connection
  + Data communication sending/receiving data: send()/recv() or sento() and recvfrom()
  + Closing socket
* **Windows blocking, I/O**
  + Blocking will not return to calling function unless data is available for read. Which means it will wait until data is available for read and wait until data arrives
  + Writing to socket will also be block unless all data is written successfully to line
  + C prototype for blocking I/O

Char recvmsg[100];

SOCKET confd,fd;

WSAData wsadata;

WSAStartup(MAKEWORD(2,2,&wsadata));

Socket(…);

Struct sockaddr\_in server;

Bind(..)

Listen(…);

Confd=accept(…);

While(TRUE)

{

Int r=recv(confd,recvmsg,sizeof(recvmsg),0);//**blocking in call**

int s=send(…);//**Blocking in call unless all data is written to kernel buff**.

**}**

**//closing of socket and cleanup function goes here**

* + **Window socket extension**
* Window socket extension basically refers to additional features or enhancement done to the basic Winsock API.
* Why extension was done by Windows
  + To provide advance networking features
  + Performance improvement
  + And support modern protocols
* What are the new features introduced by windows socket through Windows socket API extension?
  + Multicasting support
  + IPV6 support
  + Asynchronous I/O operation
* Some of the key extension done at Winsock API
  + Asynchronous I/O
    - Introduced from Winsock2 thereby enabling non-blocking socket communication
    - Function used for putting socket into non-blocking mode
      * WSAAsynSelect()
      * WSAEventSelect() or WSAOverlapped()
  + Multicast support
    - Enable sending data to multiple recipient in a network for this winsock2 has introduced multicast groups and internet group management protocol(IGMP)
  + Quality of service(QoS)
    - Allow application to specify how network traffic should be prioritized
    - Useful for real time data like audio, VoIP or video conferencing apps to ensure low latency and high quality connection
  + IPV6 support
  + Socket filtering
    - Allow to filter the network traffic based on protocol and allowing to intercept and process packet before sending to actual application
  + Zero-copy networking
    - This allows to transfer the data directly from network to application buffer without copying data to kernel buffer.
  + DLL support
    - Winsock allow to load socket related extension via DLLs
  + Socket security
    - Support Secure socket layer
    - Transport layer security
  + **Setup and Cleanup function**
* Setup function
  + API used to initialize Window socket library before calling any Winsock function call
  + This function is the initial step in Winsock programming before calling any other Socket call eg socket(),bind(),listen(),recv(),send() etc.
  + Syntax

Int WSAStartup( WORD wVersionRequested, LPWSADATA lpWSAData);

wVersionRequested: this specify the version of Winsock you want to use which is of type word(16bits) and can be used as follows

MAKEWORD(2,2);//you want to use winsock version 2.2 minor and major version

lpWSAData: this is a pointer to WSAData structure that will be filled with information related to Winsock that has been installed to operating system once successfully executed.

If success return 0 in failed return SOCKET\_ERROR

* + **WSAData structure prototype**

***Typedef struct***

***{***  
*WORD wVersion;//Winscok version your app is requesting during WSAStartup()*

*WORD wHighVersion;//Highest version of Winsock supported*

*char szDescription[WSADESCRIPTION\_LEN+1];//Description of Winsock impl..*

*unsigned short iMaxSockets;//Max num of socket that system can handle*

*unsigned short iMaxUdpDg;//maximum size of UDP datagram*

*char \*lpVendroInfo;//pointer to vendor specific information use for debug*

***}WSADATA;***

C prototype code

WSAData wsaData;

Int result=WSAStartup(MAKEWORD(2,2),&wsaData);

If(result<0)

Printf(“error…”);

Else

{

//lets print version supported

Printf(“LowVersion[%d]\n”,(int)LOWORD(wsaData.wVersion));

Printf(“High version[%d]\n”,(int)HIWORD(wsaData.wVersion);

//you can also fetch additional data from structure wsaData;

}

**Note:**

**MAKEWORD()**: this macro is used to crate 16 bit word from two separate 8 bit values typically representing major or minor version number of software or protocol

**WORD**: unsigned short integer(2 bytes)

* + **LPWSADATA:** it is long pointer of type WSADATA structure which is pass in WSAStartup() function to check if application is using correct version
* Cleanup function
  + WSACleanup() function is used to clean up the Winsock library after application has finished using it.
  + It is necessary to call WSACleanup() for each WSAStartup()
  + Syntax:

Int WSACleanup(void);//does not take any parameters

Return Value:

0 if successful operation

SOCKET\_ERROR if the functions fails

* + Purpose of WSACleanup()
    - Release Winsock resource for reuse
    - Allow further Winsock initialization
  + C prototype

WSAData wsaData;

Int result=WSAStartup(MAKEWORD(2,2,&wsaData))

//all the socket call for reading and writing…

//finished for I/O operation

//releasing resources

closesocket(fd);

Colsesocket(confd);

WSACleanup();//final call in winsock

NOTE: closesocket(): want to close the socket once done with it

WSACleanup(): is used to cleanup the Winsock library after your

Application is finished using Winsock function. Releases global resources

Allocated by Winsock library after WSAStartup()

* + **Function for handling blocking I/O**
* WSAIsBlocking(): this function is used to check if call to socket has blocked in mode, it is part of Winsock1.1 but deprecated in Winsock2

int WSAIsBlocking(void);

Return Nonzero: A blocking call in progress

Zero: No blocking call in progress

C prototype

//initializing winsock

WSAStartup(…);

If(WSAIsBlocking())

Printf(“blocking call in progress);

Else

Printf(“no blocking call in progress);

* WSACancelBLockingCall(): Cancel a call to window socket that might be blocked and deprecated in Winsock2

int WSACancelBlockingCall(void);

return type 0 success else SOCKET\_ERROR

C prototype

WSAStartup(…)

Sock=socket(…)

Filling socket address struct

Strcut sockaddr\_in server;

Server.sin\_family=AF\_INET……..

Result=connect(…)

Result=recv(…)

Flag= WSACancelBlockingCall();

If(Flag==SOCKET\_ERROR)//handle it

Closesocket()

WSACleanup()

* WSASetBlockingHook(): User define entry point or function with winsock that will called when a socket is blocked. Deprecated in Winsock2

FARPROC WSASetBlockingHook(FARPROC lpBlockFunc);

lbBlockFunc: A pointer to user defined function that will be set as new blocking hook.

Returns: Return to previous blocking hook function

Else Null will be return on fail

C prototype

//creating custom blocking hook function

BOOL FAR PASCAL myBlockingHookFunction()

{

Printf(“Custom Blocking Hooking is ready for setting);

Sleep(500);

Return false;//return false to allow the blocking call to continue

}

Int main(){

WSAData wsadata;

FARPROC prevHook;

WSAStartup(…);

//setting blocking hook custome

prevHook= WSASetBlockingHook((FARPROC) myBlockingHookFunction);

if(prevHook==NULL)

error handle it

else

set the custom function

sock=socket(…)

//setup address

Server.sin\_family=AF\_INET;…..

Connect(…);

//once connected restoring original blocking hook

WSASetBlockingHook(prevHook);  
}

**Function replaced by modern calls**

1. Non-Blocking socket

Ioctlsocket() or WSAEventSelect()

1. Asynchronous I/O

WSAAsynSelect() or WSAEventSelect()

Overlapped I/O

* WSAUnhookBlockingHook(): This function is used to remove a previously set blocking hook function that was installed using WSASetBlockingHook()

Deprecated in winsock2

Syntax:

int WSAUnhookBlockingHook(void);

Return 0 success or SOCKET\_ERROR on failed

C prototype.

BOOL FAR PASCAL myBlockingHookFunction()

{

Printf(“Custom Blocking Hooking is ready for setting);

Sleep(500);

Return false;//return false to allow the blocking call to continue

}

Int main()

{

WSAStartup(…)

prevHook= WSASetBlockingHook((FARPROC) myBlockingHookFunction);

if(repvHook==SOCKET\_ERROR)//handle it

sock=socket(…)

fill address struct

server.sin\_family=AF\_INET;…..

connect(…)

if (WSAUnhookBlockingHook() == 0)

printf(“custom blocking remove successfully”);

else

printf(“Error Removing it..”)

//Cleaning

Closesocket(…)

WSACleanup();

{

}

* **Making socket to Non-blocking mode (Easy way)**

Int PASCAL FAR ioctlsocket(SOCKET soc, long cmd,u\_long FAR \* argp);

Portotype in c

Soc=socket(…);

Unsigned long sockmode=1;//marking socket to non-blocking

If(ioctlsocket(sock,FIONBIO,(u\_long FAR\*)&sockmode)==SOCKET\_ERROR)

Print error

Else //socket set for nonblocking

* + Asynchronous Database function
* **WSAAsyncGetServByName():** Asynchronous form of **getservbyname()**

WSAAsyncGetServByName(HWND hWnd, u\_int wMsg, const char\* lpServiceName,

const char\* lpProtocolName, struct servent\* lpServEnt, int nBufLen);

***hWnd***:Handle to window that will receive message

***wMsg***: the message to be poste to window when operation complete

***lpServiceName:***service name eg:http

***lpProtoclName***: protocol name eg:tcp

***lpServEnt***: pointer to servent structure to be filled with result

***nBuflen***: size of buffer

**struct servent {**

char \*s\_name; //official name of service eg “http”

char \*\*s\_aliases; //list of alternative name of service eg.”www”

int s\_port; //port number of service network byte order

char \*s\_proto; //protocol use by service “udp”,”tcp”

};

Example:

struct servent\*serv

UINT res = WSAAsyncGetServByName(hWnd, wMsg, "http", "tcp", &serv, sizeof(serv));

//modern call

struct servent \*getservbyname(const char \*name, const char \*proto);

const char \*service\_name = "http";

const char \*protocol = "tcp";

struct servent \*service = getservbyname(service\_name, protocol);

printf("Service: %s\n", service->s\_name);

printf("Port: %d\n", ntohs(service->s\_port));

printf("Protocol: %s\n", service->s\_proto);

* **WSAAsyncGetServByPort():** Asynchronous form of **getservbyproto()** used to retrieve service information by port number

UINT WSAAsyncGetServByPort(HWND hWnd, u\_int wMsg, int port, const char \*proto,

truct servent \*lpServEnt, int nBufLen);

***hWnd***:Handle to window that will receive message

***wMsg***: the message to be poste to window when operation complete

***port:***port number in network byte order

***\*proto***: protocol eg “tcp”,”udp”

***` lpServEnt***: pointer to servent structure to service info

***nBuflen***: size of buffer

int port = htons(80);

char \*proto = "tcp";

UINT asyncResult = WSAAsyncGetServByPort(hWnd, wMsg, port, proto, &serv, sizeof(serv));

**Morden function**

struct servent \*getservbyproto(const char \*proto, const char \*proto\_name);

const char \*protocol = "tcp";//protocol name

const char \*protocol\_name = NULL;// often use null

struct servent \*service = getservbyproto(protocol, protocol\_name);

printf("Protocol: %s\n", service->s\_proto);

printf("Service: %s\n", service->s\_name);

printf("Port: %d\n", ntohs(service->s\_port));

**Note servent structure already explain above**

* **WSAAsyncGetProtoByName():** asynchronous form of **getprotobyname()**

UINT WSAAsyncGetProtoByName(HWND hWnd, u\_int wMsg, const char \*name, struct protoent \*lpProtoEnt, int nBufLen);

***hWnd***:Handle to window that will receive message

***wMsg***: the message to be poste to window when operation complete

***\*name:***the name of protocol like “tcp”,”udp”

***`*** lpProtoEnt: pointer to protent structure to store protocol information

***nBuflen***: size of buffer

**struct protoent {**

char \*p\_name; //official name of protocol eg “tcp”,”udp”

char \*\*p\_aliases; //list of alias for the protocol

int p\_proto; //protocol number eg IPPROTO\_TCP,IPPROTO\_UDP

};

Example:

struct protoent proto;

const char \*protocol\_name = "tcp";

UINT asyncResult = WSAAsyncGetProtoByName(hWnd, wMsg, protocol\_name, &proto, sizeof(proto));

**Modern Function**

struct protoent \*getprotobyname(const char \*name);

const char \*protocol\_name = "tcp";//protocol name

struct protoent \*protocol = getprotobyname(protocol\_name);

printf("Protocol Name: %s\n", protocol->p\_name);

printf("Protocol Number: %d\n", protocol->p\_proto);

* **WSAAsyncGetProtoByNumber():** asynchronous form of **getprotobynumber()**

UINT WSAAsyncGetProtoByNumber(HWND hWnd, u\_int wMsg, int protocolNumber, struct protoent \*lpProtoEnt, int nBufLen);

***hWnd***:Handle to window that will receive message

***wMsg***: the message to be poste to window when operation complete

***protocon number:***protocol number we want to query eg IPPROTO\_TDP,IPPROTO\_UDP

***`*** lpProtoEnt: pointer to protent structure to store protocol information

***nBuflen***: size of buffer

***Example***:

struct protoent proto;

int protocolNumber = IPPROTO\_TCP;//protocol number for tcp or can use 6

UINT asyncResult = WSAAsyncGetProtoByNumber(hWnd, wMsg, protocolNumber, &proto, sizeof(proto));

**Modern Function:**

struct protoent \*getprotobynumber(int proto);

int protocolNumber = IPPROTO\_TCP;//protocol number for TCP

struct protoent \*protocol = getprotobynumber(protocolNumber);

printf("Protocol Name: %s\n", protocol->p\_name);

printf("Protocol Number: %d\n", protocol->p\_proto);

* **WSAAsyncGetHostByName():** asynchronous form of **gethostbyname**

UINT WSAAsyncGetHostByName(HWND hWnd, u\_int wMsg, const char \*name, struct hostent \*lpHostEnt, int nBufLen);

***hWnd***:Handle to window that will receive message

***wMsg***: the message to be poste to window when operation complete

***name:*** the string representing domain name or hostname eg([www.cnet.com](http://www.cnet.com) or phantom)

***`*** lpHostEnt: pointer to hostent structure to store protocol information

***nBuflen***: size of buffer

**struct hostent {**

char \*h\_name; //official name of host

char \*\*h\_aliases; // list of alias for host

short h\_addrtype; //Address type eg AF\_INT for IPV4

short h\_length;//length of address

char \*\*h\_addr\_list; //list of ip address associated with host

};

Example:

struct hostent host;

const char \*hostname = "www.cnet.com";

UINT asyncResult = WSAAsyncGetHostByName(hWnd, wMsg, hostname, &host, sizeof(host));

**Modern Function**

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

const char \*hostname = "www.cnet.com";

struct hostent \*host = gethostbyname(hostname);

printf("Official name: %s\n", host->h\_name);

* **WSAAsyncGetHostByAddr():**asynchronous form of **gethostbyaddr()**

UINT WSAAsyncGetHostByAddr(HWND hWnd, u\_int wMsg, const char \*addr int addrLen, int type, struct hostent \*lpHostEnt, int nBufLen);

***hWnd***:Handle to window that will receive message

***wMsg***: the message to be poste to window when operation complete

***addr:***pointe to IP address in network byte order

**addrLen**: length of the IP address(Usually IPV4

**type**://address family eg AF\_INET for IPV4

***`*** lp lpHostEnt: pointer to hostent structure to store protocol information

***nBuflen***: size of buffer

***Example***:

struct hostent host;

const char \*ipAddress = "127.0.0.1";//ip address to resolve

struct in\_addr addr;

addr.s\_addr = inet\_addr(ipAddress);

UINT asyncResult = WSAAsyncGetHostByAddr(hWnd, wMsg, (const char \*)&addr, sizeof(addr), AF\_INET, &host, sizeof(host));

**Modern Function:**

struct hostent \*gethostbyaddr(const void \*addr, socklen\_t len, int type);

const char \*ip = "8.8.8.8";

inet\_pton(AF\_INET, ip, &addr)//converting string IP to binary form

struct hostent \*host = gethostbyaddr((const char \*)&addr, sizeof(addr), AF\_INET);

printf("Host name: %s\n", host->h\_name);//printing host name

* + Asynchronous I/O functions
* Asynchronous I/O is a method of processing operation in non-blocking mode
* This method of I/O operation allow program to continue executing other task while waiting for I/O operation to complete.
* This has significant impact on performance improvement especially in I/O bound application.
* WSAAsynSelect()
  + It is a function in the Windows Socket API that allows user to perform asynchronous operation for network socket operation in Windows.
  + It is used when user want to notify the application regarding various events like: readiness for reading or writing without blocking the application
  + WSAAsynSelect() enables asynchronous socket notification by associating a socket with a message window(Basically a message loop in a GUI or Console)
  + And specifying which events you want to be notified about.
  + This is done by posting Windows messages to be a specified window when certain socket events occur. E.g reading , writing.
* Syntax

Int WSAAsynSelect( Socket s, HWND hWnd, u\_int wMsg,long lEvent)

S=socket descriptor that will receive asynchronous notification

HWnd: window handle to receive message, generally this is the handle to a window that process message

wMsg: The message to post to window when an event occurs

lEvent: The bitmask that specifies the events you want to be notified about some of the example are:

FD\_READ: Data is available to read

FD\_WRITE: The socket is ready for writing

FD\_OOB: OOB data is available

FD\_ACCEPT: A connection is being accepted on listening socket

FD\_CLOSE: The socket has been closed.

Return value 0 on success else SOCKET\_ERROR.

**//Window Procedure to handle socket events Only for your Clarity**

LRESULT CALLBACK WndProc(HWND hWnd,UINT message, WPARAM wParam,LPARAM lParam)

{

Switch(message)

{

Case WM\_SOCKET:

SOCKET s=(SOCKET)wParam;

Long event=lParam;

//now checking for read,write,close event

If(event & FD\_READ)//if socket ready to read

{

Int r=recv(s,buffer,sizeof(buffer),0);

}

If(event & FD\_WRITE)//if data is ready to write

{

Int b=send(s,data,sizeof(data),0);

}

If(event & FD\_CLOSE)//socket close

{

Closesocket(s);

}

}

//now main sock program include<window.h>//hWnd

Int main(int arg,char \*\* argv)

{

Struct sockaddr\_in client;

WNDCLASS wc;//struct create GUI application using win32 API

HWND hWnd;//window handle(return hWnd) to handle control(Button)

WSAStartup(…);

Socket(…);

Client.sin\_family=AF\_INET……….

Connect(…)

**//Now Registering Socket for Asynchronous notification**

Wc.lpfnWndProc=WndProc**;//**window procedure to process message

Wc.hInstance=GetModuleHandle(NULL);

Wc.lpszClassName=”WocketWindow”;

Wc.style=CS\_VENDRAW | CS\_HREDRAW;

If(!RegisterClass(&wc)

{

//error in registering

}

//now creating hidden window to receive socket message

hWnd=CreateWindow(“SocketWindow”,”Socket

Notification”,0,0,0,0,0,0,0,0,wc.hInstance,Null);//for creating window

If(hWnd==NULL)

Error in window creating

}

**//Now associate socket to window for receiving notification**

WSAAsynSelect(sock,hWnd,WM\_SOCKET, FD\_READ | FD\_WRITE |FD\_CLOSE);

**//Now running message loop to handle the socket notification**

MSG msg;//structure to store message define in <window.h>

While(GetMessage(&msg,NULL,0,0)

{

//Two function comes under win32 API

TranslateMessage(&msg);//for processing key board input message

DispatchMessage(&msg);//It is responsible for sending message to

Appropriate window procedure for processing our case **WndProc()**

}

}

* Let check **select()** call in windows as well should not be issue in understanding(UNIX)
* Select() function is use to monitor multiple socket to check if it is ready for event such as readable, mwritable or exceptions
* Header file #include<winsock2.h>
* Syntax

Int select(int nfds,fd\_set \*readfds,fd\_set \*writefds,fd\_set\*exceptfds,cont struct timeval\*timeout);//same as unix please got through it

* Discussing of FD\_SET
  + Use of FD\_SET is a data structure used to represent a set of descriptor in case of select()
  + We can use several macro to manipulate on fd\_set
  + FD\_ZERO(fd\_set \*set);//clear all the set
  + FD\_SET(int fd,fd\_set \*set);// Adds socket descriptor to set
  + FD\_CLR(int fd,fd\_set \*set);//remove socket descriptor from set
  + FD\_ISSET(int fd,fd\_set \*set);//to check if socket is in set

Lets deploy it in Server Socket to handle multiple client and read/write operation

#include<stdio.h>

#include<winsock2.h>

#include<windows.h>

#include<stdlib.h>

#define MAX\_CLIENTS 10

**Int main(int a,char \*\* argv)**

**{**

WSAData wsadata;

SOCKET servsock,clisock[MAX\_CLIENTS];

Fd\_set readfds;

Int max\_sd,sd,event,consock,valread,addrlen;

//implement all socket call including strcut sock\_addr\_in

Socket(…)

Bind(….)

Listen(…)

//now clearing client socket array

Memset(clisock,0,sizeof(clisock));

Addrlen=sizeof(cliaddr);//sruct sockaddr\_in

While(TRUE)

{

//clearing readfd set and add the server socket

FD\_ZERO(&readfds);

FD\_SET(servsock,&readfds);

Max\_sd=servsock;

**//now adding client socket to set**

For(int i=0;i<MAX\_CLIENTS;i++)

{

Sd=**clisock[i];**

If(sd>0)

FD\_SET(sd,&readfds);

If(sd>max\_sd)

Max\_sd=sd;

}

**//now wait for activities or specific event to occur**

Event=select(max\_sd+1,&readfds,NULL,NULL,NULL);//interest in read

If(FD\_SET(servsock,&readfds)

{

//connection request from client

Confd=accept(…);

Continue;

}

//now adding new socket to array of client socket

For(i=0;i<MAX\_CLIETNS;i++)

Clisock[i]=confd;

//checking client socket for data

For(i=0;i<MAX\_CLIETNS,i++)

Sd=clisock[i]

If(FD\_ISSET(sd,&readfds)

{

Int r=recv(…);//reading done based on notification

}

}

**}**

* WSACancelAsynRequest()
  + Function in Winsock API to cancel the asynchronous request
  + Previously initiated by WSAAsynSelect() function
  + It basically allow you to cancel the request in specific socket thereby
  + Preventing any further notification from being sent to the window procedure or event
* Syntax
  + Int WSACancelAsyncRequest(DWORD dwContext);
  + DwContext:dwContext parameter that was passed in WSAAsynSelect() basically the socket descriptor you are using
  + C proto type

**Note**:Other calls in WSAAsynSelect() I have not included here!!!!

WSAAsynSelect(sock,hWnd,WM\_SOCKET,FD\_READ |FD\_WRITE|FD\_CLOSE0;

WSACancelAsncRequest((DWORD)sock);

* WSARecvEx()
  + This Is the function provided by Winsock API used for receiving data from socket
  + It is extended version of WSARecv() offering enhanced function and performance when working with large message.
  + Deprecated in modern version of winsock
  + Syntax
  + Int WSARecvEx(SOCKET s,LPVOID lpBuffer,DWORD dwBufferLength,LPWORD lpNuberofBytesRecvd,LPDWORD lpFlags);

**s**:socket descriptor

**lpbuffer**:A pointer to the buffer where the incoming data will be stored

**dwBufferLength**: the size in bytes of lpbuffer

**lpNumberOfBytesRecvd**: a Pointer to variable that receives actual number of bytes received by call

**lpFlags**: typicall set to 0 (changing the behavior of function eg MSG\_PEEK if needed)

**Example**

#define BUFFER\_SIZE 1024

Char buffer[BUFFER\_SIZE];

DWORD bytesRev=0;

DWORD flags=0;

Sock call

Socket(…)

Connect(…)

Int r=WSARecvEx(sock,buffer,BUFFER\_SIZE,&bytesRev,&flags);

* + Error Handling Function
* WSASetLastError()
  + Function in window socket API that retrieves the most recent error code that was sent by a Winsock function
  + Equivalent to errno in UNIX
  + Syntax
  + Int WSAGetLastError(void)
  + No input parameter
  + Return Value
    - Function returns an integer representing the error code of the most Winsock function was failed.
  + Common Error function
    - WSAEINPROGRESS: Blocking operation is in currently progress
    - WSAEINTGAL: invalid argument was passed
    - WSAENOTSOCK: The descriptor is not socket
    - WSAENOTCONN: the socket was not connected
    - WSAENTDOWN: the network is down etc.
  + Example

If(WSAStartup(MAKEWORD(2,2),&wsadata)!=0)

{

Printf(“Error in Initializing WINSOCK API[%d]\n”,WSAGetLastError());

}

* WSAGetLastError()
  + This function allows user to explicitly set the last error code for a Winsock function
  + It is useful if you need specific error code after performing socket operation
* Syntax
  + Void WSASetLastError(int iError);
  + iError: error code that you want to set as the last error and this should be one of the error code defined in the Winsock error codes eg>WSAEINVAL,WSAEWOULDBLOCK etc
  + Return value none
* Example

WSASetLastError(WSAEINTVAL);//set an invalid argument error

//now checking last error calling WSAGetLastError

Int error=WSAGeLastError();

Printf(“Last Error Code%d\n”,error);

* + Using Nonblocking socket

3.11 Non-blocking with connect

* Socket are by default blocking in nature which means the socket will not return unless data will available for read for user
* Non-Blocking with socket with connect means it will immediately return to calling function and not wait for the connection to complete.
* In order to set socket as non-blocking we will be using select() call for it and monitor the socket to see when the connection attempt has either failed or success.
* Select() call syntax you know
* Example

#include<stdio.h>

#include<winsock2.h>

#include<windows.h>

#define PORT 9999

#define IP “127.0.0.1”

Int main()

{

WSAData wsadata;

SOCKET sock;

Struct sockaddr\_in server;

Fd\_set writefds;

Struct timeval timeout;

Int result;

WSAStartup(…);

Sock=socket(…);

U\_long mode=1;//enabling non-blocking mode

Result=ioctlsocket(sock,FIONBIO,&mode);

If(result!=0)

Error

//fill address structure in sockaddr\_in object i.e server

//using select

FD\_ZERO(&writefds);

FD\_SET(sock,&writefds);

Timeout.tv\_sec=5;//5 second timeout

Timeout.tv\_usec=0;

Result=select(0,NULL,&writefds,NULL,&timeout);

If(result>0)

{

If(FD\_ISSET(sock,&writefds))

{

Int error;

Int len=sizeof(error);

If(getsockopt(sock,SOL\_SOCKET,SO\_ERROR,(char\*)&error,&len)==0)

{

If(error==0)

//connection established successfully

//process data

Else

//connection establishment failed

}

}

}

}

Explanation:

Select(): is used to check if the socket is ready for writing which return either connection is established or failed

FD\_SET(sock,&writefds):set socket to be checked if available for writing

Timeout.tv\_sec=5: waits for 5 seconds for the connection to complete.

* + select in conjunction with accept
* select in conjunction with accept will allows user to handle incoming connections on a server socket in a non-blocking mode
* the select function help to monitor multiple socket for certain event like read,write and exception without blocking the program.
* It is useful when you want your server application to **accept new connection** without blocking call **and handle multiple client simultaneously**
* Select() system call you know
* Example

1. Header file needed as in select with non-blocking file
2. All the function like above in only below code is to be added however the call is with client i.e connect call now we will check for accept call

Fd\_set readfds;

Struct timeval timeout;

SOCKET sock,confd;

Int result;

sock=Socket(…);

Bind(…);

Listen(…);

//now calling select()

While(1)

{

FD\_ZERO(&fdreadfds);//initializing to zero

FD\_SET(sock,&readfds);//monitoring listen socket for incoming connection

//setting timeout value

Timeout.tv\_sec=5;

Timeout.tv\_usec=0;

Result=select(0,&readfds,NULL,NULL,&timeout);

//check error return at Result

If(FD\_SET(confd,&readfds)//checking if listening socket ready to accept con

{

//ok listening socket is ready for reading incoming connection

Confd=accept(sock,(struct sockaddr\_in\*)&client,&clientlen);

//checking error

}

Send(…);

Recv(…);

}

* + **Windows Specific Socket Call**.

LPWSAPROTOCOL\_INFO lpproto;

GROUP aa=0;

DWORD flag;

**SOCKET sock=WSASocket(AF\_INET,SOCK\_STREAM,IPPROTO\_TCP,&lpproto,aa,flag);**

SOCKET a = WSASocket(AF\_INET, SOCK\_STREAM, 0, NULL, 0, NULL);**//100% ok**

**SOCKET WSASocket(**

int af, //famil AF\_INET

int type, //SOCK\_STREAM

int protocol, //IPPROTO\_TCP

LPWSAPROTOCOL\_INFO lpProtocolInfo, //NULL unless we need to provide specific protocol

//details

GROUP g, //reserved parameter need to set to 0

DWORD dwFlags //to control socket creationg eg WSA\_FLAG\_OVERLAPPED//create ovelap

**);**

**int WSABind(**

SOCKET s,//descriptor

const struct sockaddr \*name,//address strcuture

int namelen,//len of address strcuture

LPWSAOVERLAPPED lpOverlapped,//optional buffer for overlapped socket

LPWSAOVERLAPPED\_COMP//LETION\_ROUTINE lpCompletionRoutine//flag for specific

//socket operation

**);**

**Int WSAListen(**

Socket s,

Int backlog

**);// you know**

**SOCKET WSAAPI WSAAccept(**

SOCKET s, //you know

struct sockaddr \*addr,//you know

int addrlen, //you know

LPWSAOVERLAPPED lpOverlapped, //pointer to overlapped struct for asynchronous I/O

DWORD dwFlags;//flag that specify additional behavior

**);**

**Int WSAConnect(**

SOCKET s,

Const struct sockaddr\*name,

Int namelen,

LPWSABUF lpTransmitterBuffer,//optional buffer for transmitting data during connection

LPWSABUF lpReceiverBuffer,//optional buffer for receiving data during the connection

LPQOS lpQos,//optional Qos information

LPQos lpSqos//optional Qos information for the connection

);

**int WSAAPI WSARecv(**

SOCKET s,

LPWSABUF lpBuffers,//pointer to an array of WSABUF structures

DWORD dwBufferCount,//The number of WSABUF structure in the array(size)

LPDWORD lpNumberOfBytesRecvd,//pointer to number of bytes received

LPDWORD lpFlags,//flags modified the behavior of the functions

LPWSAOVERLAPPED lpOverlapped,//pointer to overlapped struct for asynchronous I/O

//if null call is synchronous

LPVOID lpCompletionRoutine lpCompletionRoutine//Completion routine for asy I/O

)

**int WSAAPI WSASend(**

SOCKET s,

LPWSABUF lpBuffers,//pointer to an array of WSABUF structures

DWORD dwBufferCount,//The number of WSABUF structure in the array(size)

LPDWORD lpNumberOfBytesSent,//pointer to number of bytes received

LPDWORD lpFlags,//flags modified the behavior of the functions

LPWSAOVERLAPPED lpOverlapped,//pointer to overlapped struct for asynchronous I/O if null

// the call is synchronous

LPVOID lpCompletionRoutine lpCompletionRoutine//Completion routine for asynchronous

)

3.13 select with recv()/recvfrom() and send()/sendto()

1. Server socket

Bind socket to port 8888 on localhost

\*/

#include<io.h>

#include<stdio.h>

#include<winsock2.h>

#pragma comment(lib,"ws2\_32.lib") //Winsock Library

int main(int argc , char \*argv[])

{

WSADATA wsa;

SOCKET s , new\_socket;

struct sockaddr\_in server , client;

int c;

char \*message;

printf("\nInitialising Winsock...");

if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)

{

printf("Failed. Error Code : %d",WSAGetLastError());

return 1;

}

printf("Initialised.\n");

//Create a socket

if((s = socket(AF\_INET , SOCK\_STREAM , 0 )) == INVALID\_SOCKET)

{

printf("Could not create socket : %d" , WSAGetLastError());

}

printf("Socket created.\n");

//Prepare the sockaddr\_in structure

server.sin\_family = AF\_INET;

server.sin\_addr.s\_addr = INADDR\_ANY;

server.sin\_port = htons( 8888 );

//Bind

if( bind(s ,(struct sockaddr \*)&server , sizeof(server)) == SOCKET\_ERROR)

{

printf("Bind failed with error code : %d" , WSAGetLastError());

}

puts("Bind done");

//Listen to incoming connections

listen(s , 3);

//Accept and incoming connection

puts("Waiting for incoming connections...");

c = sizeof(struct sockaddr\_in);

new\_socket = accept(s , (struct sockaddr \*)&client, &c);

if (new\_socket == INVALID\_SOCKET)

{

printf("accept failed with error code : %d" , WSAGetLastError());

}

puts("Connection accepted");

//Reply to client

message = "Hello Client , I have received your connection. But I have to go now, bye\n";

send(new\_socket , message , strlen(message) , 0);

getchar();

closesocket(s);

WSACleanup();

return 0;

}

1. Client Socket

\*

Create a TCP socket

\*/

#include<stdio.h>

#include<winsock2.h>

#pragma comment(lib,"ws2\_32.lib") //Winsock Library

int main(int argc , char \*argv[])

{

WSADATA wsa;

SOCKET s;

struct sockaddr\_in server;

printf("\nInitialising Winsock...");

if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)

{

printf("Failed. Error Code : %d",WSAGetLastError());

return 1;

}

printf("Initialised.\n");

//Create a socket

if((s = socket(AF\_INET , SOCK\_STREAM , 0 )) == INVALID\_SOCKET)

{

printf("Could not create socket : %d" , WSAGetLastError());

}

printf("Socket created.\n");

server.sin\_addr.s\_addr = inet\_addr("74.125.235.20");

server.sin\_family = AF\_INET;

server.sin\_port = htons( 80 );

//Connect to remote server

if (connect(s , (struct sockaddr \*)&server , sizeof(server)) < 0)

{

puts("connect error");

return 1;

}

puts("Connected");

return 0;

}

/\*

Create a TCP socket

\*/

#include<stdio.h>

#include<winsock2.h>

#pragma comment(lib,"ws2\_32.lib") //Winsock Library

int main(int argc , char \*argv[])

{

WSADATA wsa;

SOCKET s;

struct sockaddr\_in server;

char \*message;

printf("\nInitialising Winsock...");

if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)

{

printf("Failed. Error Code : %d",WSAGetLastError());

return 1;

}

printf("Initialised.\n");

//Create a socket

if((s = socket(AF\_INET , SOCK\_STREAM , 0 )) == INVALID\_SOCKET)

{

printf("Could not create socket : %d" , WSAGetLastError());

}

printf("Socket created.\n");

server.sin\_addr.s\_addr = inet\_addr("74.125.235.20");

server.sin\_family = AF\_INET;

server.sin\_port = htons( 80 );

//Connect to remote server

if (connect(s , (struct sockaddr \*)&server , sizeof(server)) < 0)

{

puts("connect error");

return 1;

}

puts("Connected");

//Send some data

message = "GET / HTTP/1.1\r\n\r\n";

if( send(s , message , strlen(message) , 0) < 0)

{

puts("Send failed");

return 1;

}

puts("Data Send\n");

return 0;

}

1. ===============================END of Course======================================

**Appendix-1: Select System Call Demo**

#include "stdafx.h"

#include<winsock2.h>//winsock

#include<Windows.h>//win

#include<stdlib.h>//exit

#include<process.h>//for dos command

#pragma comment(lib, "Ws2\_32.lib")

#define PORT 8888

#define BUFFSIZE 200

/\*Function for Handling Asynchronous connection and I/O operation\*/

int handleConnection(SOCKET fd);

int main(int arg, char\*\* argv[])

{

SOCKET fd, confd;//variable socket descriptro created to store connected and non connected socket.

struct sockaddr\_in serv,cli;//two structure of type internet domain is created

WSADATA wsd; //Wsadata structure is created to store the information return from kernel after successfuly execution of WSAStartup()

int ret,b,l,result,len;//deinfe variable of type int

DWORD ver = MAKEWORD(2, 2);//creating the variable to store version,makeword will convert the int into DWROD datatypes need to use in first parameter at WSAStartup()

ULONG NONBlock = 0;//seting socket to non blocking mode

if (ret = WSAStartup(ver, &wsd) != 0)//now we initalize the DLL and check for the version supplied on it if success will return value on wsadata strucute else return error.

{

printf("Error In Library initialization\n");

printf("Error Received[%s]", WSAGetLastError());

}

else{ printf("Successfully loaded library\n");

printf("Value on WSAStructure[%s] and Version supported", wsd.szDescription);

printf("Value of SZSsystemStatus[%c]", wsd.szSystemStatus);

}

serv.sin\_family = AF\_INET;

serv.sin\_port = htons(PORT);

serv.sin\_addr.S\_un.S\_addr = htonl(INADDR\_ANY);

fd = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if (fd > 0){ printf("End Point Created Successfully value is[%d]\n", fd); }

else{ printf("Error in Creating Socket\n"); }

if( b = bind(fd, (struct sockaddr\*)&serv, sizeof(serv))==0)

{

printf("Bind Successful");

}

else{ printf("Not able to Bind\n"); exit(-15); }

if (l = listen(fd, 10) != -1)

{

//printf("Program Listening at prot[%d] IP[%s]\n", PORT, inet\_ntoa(serv.sin\_addr));

}

if (ioctlsocket(fd, FIONBIO, &NONBlock) == SOCKET\_ERROR)//seting socket to nonblocking mode.

{

printf("Soket not able to set as NonBlocking\n");

}

else{ printf("Ioctl operation on socket seting non block Ok\n"); }

len = sizeof(cli);

while (TRUE)

{

result = handleConnection(fd);

//result = 1;

if (result>0)

{

int s = 0;

char msg[] = "Hello this is select server";

confd = accept(fd, (struct sockaddr\*)&cli, &len);

r = recv(fd, msg, 0, 100);

s = send(fd, msg,0, sizeof(msg));

}

else if (result == 0){

printf("Select time out\n");

}

else if (result == -1){

WSACleanup();

printf("Error in select\n",WSAGetLastError());

}

}

return 0;

**}//main loop end**

int handleConnection(SOCKET fd)

{

struct timeval timeout;

struct fd\_set rfd;

timeout.tv\_sec = 5;

timeout.tv\_usec = 0;

FD\_ZERO(&rfd);//Initialize readdescriptor to zero

FD\_SET(fd, &rfd);//set descriptor for read notification.

/\*Note:value return by select one error -1 <0 if timeout 0 =0 data ready greater than 0 >0\*/

//return select(0, &rfd, 0, 0, 0);// we set time val zero which will not return unless the descriptor is ready

return select(0, &rfd, 0, 0, &timeout);

3.14 send and receiving data over connection