HEADER FILES

```
#include<time.h>
#include<stdio.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>
#include<sys/select.h>
#include<pthread.h>
#include<signal.h>
#include<stdlib.h>
#include<fcntl.h>
#include<sys/shm.h>
#include<unistd.h>
#include<sys/un.h>
#include<netinet/ip.h>
#include<arpa/inet.h>
#include<pcap.h>
#include<errno.h>
#include<netinet/if_ether.h>
#include<net/ethernet.h>
#include<netinet/ether.h>
#include<netinet/udp.h>
#include<sys/ipc.h>
#include<sys/msg.h>
```

SHARED MEMORY

```
      int state=1;
      key_t h=ftok(".",state++);   // value of state should on every
program where this share memory is used
      int shmid=shmget(h,sizeof(int),IPC_CREAT|0666);
      share_memory=shmat(shmid,(const void*)0,0);
```

SEMAPHORE

```
      void sem_wait(int semid)
      {
            struct sembuf sb;
            sb.sem_num=0;
            sb.sem_op=-1;
            sb.sem_flg=0;
            if((semop(semid,&sb,1))==-1)
            {
                  perror("\nFailed to acquire semaphore.");
                  exit(0);
            }
      }
```

```
      void sem_try_wait(int semid)
      {
            struct sembuf sb;
            sb.sem_num=0;
            sb.sem_op=-1;
            sb.sem_flg=IPC_NOWAIT;;
            return semop(semid,&sb,1);
      }

      void sem_signal(int semid)
      {
            struct sembuf sb;
            sb.sem_num=0;
            sb.sem_op=1;
            sb.sem_flg=0;
            if((semop(semid,&sb,1))==-1)
            {
                  perror("\nFailed to release semaphore.");
                  exit(0);
            }
      }

      int state=1;
      key_t h=ftok(".",state++);   // value of state should on every
program where this semaphore is used
      int sem_id;
      if((sem_id=semget(h,1,0666|IPC_CREAT))==-1)
      {
            printf("error in creation semaphore\n");
            exit(0);
      }

      int semaphore_value=1;

      if((semctl(sem_id,0,SETVAL,semaphore_value))==-1)
      {
            printf("error to set value\n");
      }

      (OR)

#define sname "/mysem"

sem_t *sem = sem_open(sname, O_CREAT, 0644, 0);
sem_t *sem = sem_open(sname,1);

sem_wait(sem);
sem_post(sem);

-------------------------------------------------------------------------
-----------------------------------------------------
                        MSG QUEUE
-------------------------------------------------------------------------
-----------------------------------------------------
      struct mymsg
      {
            long type;
            char msg[20];
```

```
};

struct mymsg msg1;
key_t  key;
int    mqpid;
int    ret;
int    len;
system("touch f1.txt");
if((key=ftok("f1.txt",'B')) == -1)
{
      perror("key");
      exit(1);
}
if((mqpid=msgget(key,0644|IPC_CREAT))==-1)
{
      perror("Key");
      exit(1);
}

if(msgsnd( mqpid ,&msg1 ,len+1 , 0) == -1)
{
      perror("msgsnd");
      exit(1);
}

memset(msg1.msg,'\0',sizeof(msg1.msg));
if(msgrcv( mqpid , &msg1 , sizeof(msg1.msg),1 ,0) == -1)
{
      perror("msgrcv");
      exit(1);
}
```

FIFO

```
char name[50];
if(mkfifo(name,0666)==-1)
{
      perror("mkfifo()1");
      exit(1);
}
if((wfd=open("./wellknownfifo",O_WRONLY))==-1)
{
      perror("open()");
      exit(1);
}


write(wfd,buffer,sizeof(buffer));


char buffer[50];
if(mkfifo("./wellknownfifo",0666)==-1)
{
      perror("mkfifo()");
      exit(1);
}
```

```
      if((rfd=open("./wellknownfifo",O_RDONLY))==-1)
      {
            perror("open()");
            exit(1);
      }

      read(rfd,buffer,50);
```

MKFIFO

```
 #include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <string.h>

  int fd;
  mkfifo("fifo1.fifo",0666);
  fd=open("./fifo1.fifo",O_RDONLY);
```

POLL

```
      int size;
      struct pollfd fds[size];
      fds[i]=open(" ", 0666);
      fds[i].events=POLLIN;

      int ret=poll(fds, size, timeout);
      if(fds[i].revents & POLLIN)
      {

      }
```

To know pid of a program by knowing its name

```
int fd = fileno(popen("pidof ./S", "r"));
char s[1000];
      read(fd, &s, 1000);
      X = atoi(s);
int fd = fileno(popen("pidof ./P2.exe", "r"));
      char s[1000];
      read(fd, &s, 1000);
      X = atoi(s);
```

pthread

```
      void do_thread_service(void *arg)
      {
            int *args= (int*)arg ;

      }

      pthread_t t_service;
      if(pthread_create(&t_service,NULL,(void*)&do_thread_service
,(void*)args)!=0)
      perror("\npthread_create ");
```

SELECT

```
      fd_set readset;
      FD_ZERO(&readset);

      int max=-1;

      for(i=0;i<no_of_file_descriptors;i++)
      {
            FD_SET(fd[i], &readset);
            if(fd[i]>max)
            max=fd[i];
      }


      struct timeval t;
      t.tv_sec=3;
      t.tv_usec=100;
      int rv = select(max + 1, &readset, NULL, NULL, &t);

      if (rv == -1)
      {
            perror("select");
      }

      else if (rv == 0)
      {
            printf("Timeout occurred!\n");
      }

      else
      {
            int i;
            // check for events
            for(i=0;i<no_of_file_descriptors;i++)
            if (FD_ISSET(fd[i], &readset))
            {

            }
      }
```

CONNECTION ORIENTED SERVER (usage -: "./a.out port_no")

```
      if(argc!=2)
      printf("\n usage ./a.out port_no");

      int sfd;
      struct sockaddr_in serv_addr,cli_addr;
      socklen_t cli_len;
      int port_no=atoi(argv[1]);

      if((sfd = socket(AF_INET,SOCK_STREAM,0))==-1)
      perror("\n socket ");
      else printf("\n socket created successfully");

      bzero(&serv_addr,sizeof(serv_addr));

      serv_addr.sin_family = AF_INET;
      serv_addr.sin_port = htons(port_no);
      serv_addr.sin_addr.s_addr = INADDR_ANY;

      int opt=1;
      setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,
&opt, sizeof(opt));

      if(bind(sfd,(struct sockaddr *) &serv_addr,sizeof(serv_addr))==-1)
      perror("\n bind : ");
      else printf("\n bind successful ");

      listen(sfd,10);

      cli_len=sizeof(cli_addr);
      int nsfd;
      if((nsfd = accept(sfd , (struct sockaddr *)&cli_addr ,
&cli_len))==-1)
      perror("\n accept ");
      else printf("\n accept successful");
      //break after exec in child
```

CONNECTION ORIENTED CLIENT (usage -: "./a.out port_no")

```
      if(argc!=2)
      printf("\n usage ./a.out port_no");

      int sfd;
      struct sockaddr_in serv_addr;
      int port_no=atoi(argv[1]);

      bzero(&serv_addr,sizeof(serv_addr));
```

```
      if((sfd = socket(AF_INET , SOCK_STREAM , 0))==-1)
      perror("\n socket");
      else printf("\n socket created successfully\n");

      serv_addr.sin_family = AF_INET;
      serv_addr.sin_port = htons(port_no);
      //serv_addr.sin_addr.s_addr = INADDR_ANY;
      inet_pton(AF_INET,"127.0.0.1", &serv_addr.sin_addr);

      if(connect(sfd , (struct sockaddr *)&serv_addr ,
sizeof(serv_addr))==-1)
      perror("\n connect : ");
      else printf("\nconnect succesful");
```

CONNECTION LESS SERVER (usage -: "./a.out port_no")

```
      if(argc!=2)
      printf("\n usage ./a.out port_no");

      int sfd;
      struct sockaddr_in serv_addr,cli_addr;
      socklen_t cli_len;
      int port_no=atoi(argv[1]);

      if((sfd = socket(AF_INET,SOCK_DGRAM,0))==-1)
      perror("\n socket ");
      else printf("\n socket created successfully");

      bzero(&serv_addr,sizeof(serv_addr));

      serv_addr.sin_family = AF_INET;
      serv_addr.sin_port = htons(port_no);
      serv_addr.sin_addr.s_addr = INADDR_ANY;

      if(bind(sfd,(struct sockaddr *) &serv_addr,sizeof(serv_addr))==-1)
      perror("\n bind : ");
      else printf("\n bind successful ");

      cli_len = sizeof(cli_addr);

      fgets( buffer , 256 , stdin );
      sendto(sfd , buffer , 256 , 0 , ( struct sockaddr * ) &cli_addr ,
cli_len);
      recvfrom(sfd , buffer , 256 , 0 , ( struct sockaddr * ) &cli_addr ,
& cli_len );
```

CONNECTION LESS CLIENT (usage -: "./a.out port_no")

```
      if(argc!=2)
      printf("\n usage ./a.out port_no");
```

```
      int sfd;
      struct sockaddr_in serv_addr;
      int port_no=atoi(argv[1]);
      char buffer[256];

      bzero(&serv_addr,sizeof(serv_addr));

      if((sfd = socket(AF_INET , SOCK_DGRAM , 0))==-1)
      perror("\n socket");
      else printf("\n socket created successfully\n");

      serv_addr.sin_family = AF_INET;
      serv_addr.sin_port = htons(port_no);
      serv_addr.sin_addr.s_addr = INADDR_ANY;

      socklen_t serv_len = sizeof(serv_addr);

      fgets( buffer , 256 , stdin );
      sendto(sfd , buffer , 256 , 0 , ( struct sockaddr * ) &serv_addr ,
serv_len);
      recvfrom(sfd , buffer , 256 , 0 , ( struct sockaddr * ) &serv_addr
, & serv_len );
```

GETPEERNAME (usage: only after accept; only on nsfd)

```
      #include <sys/types.h>
      #include <sys/socket.h>
      #include <netinet/in.h>
      #include <arpa/inet.h>
      #include <stdio.h>


{
   int s;
   struct sockaddr_in peer;
   int peer_len;

   peer_len = sizeof(peer);

   if (getpeername(s, &peer, &peer_len) == -1) {
      perror("getpeername() failed");
      return -1;
   }

      /* Print it.    */
   printf("Peer's IP address is: %s\n", inet_ntoa(peer.sin_addr));
   printf("Peer's port is: %d\n", (int) ntohs(peer.sin_port));

}
```

PASSING ARGUMENTS THROUGH EXEC

```
--------------------------------------------------------------------------
-------------------------------------------------
string msg;
char **arg=new char*[2];
arg[0]=strdup(msg.c_str());
arg[1]=NULL;
int c=fork();
if(c>0);
else if(c==0)
{
      if(execvp("./s",arg)==-1)
      cout<<eroor"<<endl;
      exit(1);
}


//retrieving in child

int main(int argc, char const *argv[])
{
      string info=argv[argc];
}

------------------ ---------------- ---------------------
```

UNIX SOCKET CONNECTION ORIENTED SERVER(usage -: "./a.out")

```
--------------------------------------------------------------------------
-------------------------------------------------------------
      #define ADDRESS  "mysocket"

      int  usfd;
      struct sockaddr_un userv_addr,ucli_addr;
      int userv_len,ucli_len;

      usfd = socket(AF_UNIX , SOCK_STREAM , 0);
      perror("socket");

      bzero(&userv_addr,sizeof(userv_addr));

      userv_addr.sun_family = AF_UNIX;
      strcpy(userv_addr.sun_path, ADDRESS);
      unlink(ADDRESS);
      userv_len = sizeof(userv_addr);

      if(bind(usfd, (struct sockaddr *)&userv_addr, userv_len)==-1)
      perror("server: bind");

      listen(usfd, 5);

      ucli_len=sizeof(ucli_addr);

      int nusfd;
      nusfd=accept(usfd, (struct sockaddr *)&ucli_addr, &ucli_len);
```

UNIX SOCKET CONNECTION ORIENTED CLIENT (usage -: "./a.out")

```
-----------------------------------------------------------------------
-------------------------------------------------------------

      #define ADDRESS     "mysocket"

      int usfd;
      struct sockaddr_un userv_addr;
      int userv_len,ucli_len;

      usfd = socket(AF_UNIX, SOCK_STREAM, 0);

      if(usfd==-1)
      perror("\nsocket  ");

      bzero(&userv_addr,sizeof(userv_addr));
      userv_addr.sun_family = AF_UNIX;
      strcpy(userv_addr.sun_path, ADDRESS);

      userv_len = sizeof(userv_addr);

      if(connect(usfd,(struct sockaddr *)&userv_addr,userv_len)==-1)
      perror("\n connect ");

      else printf("\nconnect succesful");
```

SEND_FD AND RECV_FD

```
-----------------------------------------------------------------------
-----------------------------------------------

int send_fd(int socket, int fd_to_send)
 {
  struct msghdr socket_message;
  struct iovec io_vector[1];
  struct cmsghdr *control_message = NULL;
  char message_buffer[1];
  /* storage space needed for an ancillary element with a paylod of
length is CMSG_SPACE(sizeof(length)) */
  char ancillary_element_buffer[CMSG_SPACE(sizeof(int))];
```

```
  int available_ancillary_element_buffer_space;

  /* at least one vector of one byte must be sent */
  message_buffer[0] = 'F';
  io_vector[0].iov_base = message_buffer;
  io_vector[0].iov_len = 1;

  /* initialize socket message */
  memset(&socket_message, 0, sizeof(struct msghdr));
  socket_message.msg_iov = io_vector;
  socket_message.msg_iovlen = 1;

  /* provide space for the ancillary data */
  available_ancillary_element_buffer_space = CMSG_SPACE(sizeof(int));
  memset(ancillary_element_buffer, 0,
available_ancillary_element_buffer_space);
  socket_message.msg_control = ancillary_element_buffer;
  socket_message.msg_controllen =
available_ancillary_element_buffer_space;

  /* initialize a single ancillary data element for fd passing */
  control_message = CMSG_FIRSTHDR(&socket_message);
  control_message->cmsg_level = SOL_SOCKET;
  control_message->cmsg_type = SCM_RIGHTS;
  control_message->cmsg_len = CMSG_LEN(sizeof(int));
  *((int *) CMSG_DATA(control_message)) = fd_to_send;

  return sendmsg(socket, &socket_message, 0);
 }




 int recv_fd(int socket)
 {
  int sent_fd, available_ancillary_element_buffer_space;
  struct msghdr socket_message;
  struct iovec io_vector[1];
  struct cmsghdr *control_message = NULL;
  char message_buffer[1];
  char ancillary_element_buffer[CMSG_SPACE(sizeof(int))];

  /* start clean */
  memset(&socket_message, 0, sizeof(struct msghdr));
  memset(ancillary_element_buffer, 0, CMSG_SPACE(sizeof(int)));

  /* setup a place to fill in message contents */
  io_vector[0].iov_base = message_buffer;
  io_vector[0].iov_len = 1;
  socket_message.msg_iov = io_vector;
  socket_message.msg_iovlen = 1;

  /* provide space for the ancillary data */
  socket_message.msg_control = ancillary_element_buffer;
  socket_message.msg_controllen = CMSG_SPACE(sizeof(int));
```

```
  if(recvmsg(socket, &socket_message, MSG_CMSG_CLOEXEC) < 0)
   return -1;

  if(message_buffer[0] != 'F')
  {
   /* this did not originate from the above function */
   return -1;
  }

  if((socket_message.msg_flags & MSG_CTRUNC) == MSG_CTRUNC)
  {
   /* we did not provide enough space for the ancillary element array
*/
   return -1;
  }

  /* iterate ancillary elements */
   for(control_message = CMSG_FIRSTHDR(&socket_message);
       control_message != NULL;
       control_message = CMSG_NXTHDR(&socket_message,
control_message))
  {
   if( (control_message->cmsg_level == SOL_SOCKET) &&
       (control_message->cmsg_type == SCM_RIGHTS) )
   {
    sent_fd = *((int *) CMSG_DATA(control_message));
    return sent_fd;
   }
  }

  return -1;
 }
```

UNIX SOCKET CONNECTION LESS SERVER (usage - : "./a.out")

```
     #define ADDRESS  "mysocket"

     int  usfd;
     struct sockaddr_un userv_addr,ucli_addr;
     int userv_len,ucli_len;

     usfd = socket(AF_UNIX , SOCK_DGRAM , 0);
     perror("socket");

     bzero(&userv_addr,sizeof(userv_addr));

     userv_addr.sun_family = AF_UNIX;
     strcpy(userv_addr.sun_path, ADDRESS);
     unlink(ADDRESS);
     userv_len = sizeof(userv_addr);
```

```
     if(bind(usfd, (struct sockaddr *)&userv_addr, userv_len)==-1)
     perror("server: bind");

     fgets( buffer , 256 , stdin );
     sendto(sfd , buffer , 256 , 0 , ( struct sockaddr * ) &ucli_addr
,  ucli_len);
     recvfrom(sfd , buffer , 256 , 0 , ( struct sockaddr * )
&ucli_addr , & uscli_len );
```

UNIX SOCKET CONNECTION LESS CLIENT (usage -: "./a.out")

```
     #define ADDRESS     "mysocket"

     int usfd;
     struct sockaddr_un userv_addr;
     int userv_len,ucli_len;

     usfd = socket(AF_UNIX, SOCK_DGRAM, 0);

     if(usfd==-1)
     perror("\nsocket  ");

     bzero(&userv_addr,sizeof(userv_addr));
     userv_addr.sun_family = AF_UNIX;
     strcpy(userv_addr.sun_path, ADDRESS);

     userv_len = sizeof(userv_addr);

     fgets( buffer , 256 , stdin );
     sendto(sfd , buffer , 256 , 0 , ( struct sockaddr * )
&userv_addr ,  userv_len);
     recvfrom(sfd , buffer , 256 , 0 , ( struct sockaddr * )
&userv_addr , & userv_len );
```

SOCKET PAIR (usage -: "./a.out")

```
     int usfd[2];
     if(socketpair(AF_UNIX,SOCK_STREAM,0,usfd)==-1)
     perror("socketpair ");

     int c=fork();

     if(c==-1)
     perror("\nfork ");

     else if(c>0)
     {
          close(usfd[1]);
```

```
}

else if(c==0)
{
      close(usfd[0]);
      dup2(usfd[1],0);
      execvp(file_name,args);
}
```