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>

/**
 This function is used to send file descriptor over Unix domain socket
 You can use this function with file descriptor return
 by any one of below functions
 1 .  socketpair();
 2 .  socket(AF_UNIX,...);
 3 .  socket(AF_LOCAL,...);
 @param socket file_descriptor_of_sender
 @param fd_to_send
*/
```

SHARE 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);
```

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))
	{

	}
}
```

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 ");
```

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 );
```

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);
	}
```

RAW SOCKETS

```
void print_ipheader(struct iphdr* ip)
{
	cout<<"------------------------\n";
	cout<<"Printing IP header....\n";
	cout<<"IP version:"<<(unsigned int)ip->version<<endl;
	cout<<"IP header length:"<<(unsigned int)ip->ihl<<endl;

	cout<<"Type of service:"<<(unsigned int)ip->tos<<endl;
	cout<<"Total ip packet length:"<<ntohs(ip->tot_len)<<endl;
	cout<<"Packet id:"<<ntohs(ip->id)<<endl;
	cout<<"Time to leave :"<<(unsigned int)ip->ttl<<endl;
	cout<<"Protocol:"<<(unsigned int)ip->protocol<<endl;
	cout<<"Check:"<<ip->check<<endl;
	cout<<"Source ip:"<<inet_ntoa(*(in_addr*)&ip->saddr)<<endl;
	//printf("%pI4\n",&ip->saddr );
	cout<<"Destination ip:"<<inet_ntoa(*(in_addr*)&ip->daddr)<<endl;
	cout<<"End of IP header\n";
	cout<<"------------------------\n";
}
```

RAW SOCKET SERVER

```
	if(argc<2)cout<<"Enter protocal in arguments";
```

```
        int rsfd=socket(AF_INET,SOCK_RAW,atoi(argv[1]));
        perror("socket");
        int optval=1;
        setsockopt(rsfd, IPPROTO_IP, SO_BROADCAST, &optval, sizeof(int));//IP_HDRINCL
        cout<<"opt"<<endl;
                struct sockaddr_in client;
        client.sin_family=AF_INET;
        client.sin_addr.s_addr=inet_addr("127.0.0.1");

                char buff[]="hello";
        client.sin_addr.s_addr=INADDR_ANY;

        unsigned int client_len=sizeof(client);
        cout<<"sending"<<endl;
  sendto(rsfd,buff,strlen(buff)+1,0,(struct sockaddr*)&client,sizeof(client));
  perror("send");
```

RAW SOCKET CLIENT

```
        if(argc<2)cout<<"Enter protocol in arguments\n";
        int rsfd=socket(AF_INET,SOCK_RAW,atoi(argv[1]));
        if(rsfd==-1)custom_perror("socket")
                char buf[BUF_LEN];
        struct sockaddr_in client;
        socklen_t clilen=sizeof(client);
        cout<<"receive"<<endl;
        recvfrom(rsfd,buf,BUF_LEN,0,(sockaddr*)&client,(socklen_t*)clilen);
        perror("recv");
         struct iphdr *ip;
  ip=(struct iphdr*)buf;
        cout<<(buf+(ip->ihl)*4)<<endl;
```

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];
}
```

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)
        {

        }
```

LIBPCAP

```
#include<pcap.h>
#include<stdio.h>
#include<stdlib.h> // for exit()
#include<string.h> //for memset

#include<sys/socket.h>
#include<arpa/inet.h> // for inet_ntoa()
#include<net/ethernet.h>
#include<netinet/ip_icmp.h>   //Provides declarations for icmp header
#include<netinet/udp.h>   //Provides declarations for udp header
#include<netinet/tcp.h>   //Provides declarations for tcp header
#include<netinet/ip.h>    //Provides declarations for ip header

void process_packet(u_char *, const struct pcap_pkthdr *, const u_char *);
void process_ip_packet(const u_char * , int);
void print_ip_packet(const u_char * , int);
void print_tcp_packet(const u_char *  , int );
void print_udp_packet(const u_char * , int);
void print_icmp_packet(const u_char * , int );
void PrintData (const u_char * , int);

FILE *logfile; //to store the output
struct sockaddr_in source,dest;
int tcp=0,udp=0,icmp=0,others=0,igmp=0,total=0,i,j;

int main()
{
   pcap_if_t *alldevsp , *device;
   pcap_t *handle; //Handle of the device that shall be sniffed

   char errbuf[100] , *devname , devs[100][100];
   int count = 1 , n;

   // get the list of available devices
   printf("Finding available devices ... ");
   if( pcap_findalldevs( &alldevsp , errbuf) )
   {
      printf("Error finding devices : %s" , errbuf);
      exit(1);
   }
   printf("Done");

   //Print the available devices
   printf("\nAvailable Devices are :\n");
   for(device = alldevsp ; device != NULL ; device = device->next)
   {
```

```
        printf("%d. %s - %s\n" , count , device->name , device->description);
        if(device->name != NULL)
        {
            strcpy(devs[count] , device->name);
        }
        count++;
    }

    //Ask user which device to sniff
    printf("Enter the number of the device you want to sniff : ");
    scanf("%d" , &n);
    devname = devs[n];

    //Open the device for sniffing
    printf("Opening device %s for sniffing ... " , devname);
    handle = pcap_open_live(devname , 65535 , 1 , 0 , errbuf);

    if (handle == NULL)
    {
        fprintf(stderr, "Couldn't open device %s : %s\n" , devname , errbuf);
        exit(1);
    }
    printf("Done\n");

    logfile=fopen("log.txt","w");
    if(logfile==NULL)
    {
        printf("Unable to create file.");
    }

    //Put the device in sniff loop
    pcap_loop(handle , 0 , process_packet , NULL);
    return 0;
}

void process_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *buffer)
{
    int size = header->len;

    //Get the IP Header part of this packet , excluding the ethernet header
    struct iphdr *iph = (struct iphdr*)(buffer + sizeof(struct ethhdr));
    ++total;
    switch (iph->protocol) //Check the Protocol and do accordingly...
    {
        case 1:  //ICMP Protocol
            ++icmp;
            print_icmp_packet( buffer , size);
            break;

        case 2:  //IGMP Protocol
            ++igmp;
            break;

        case 6:  //TCP Protocol
            ++tcp;
            print_tcp_packet(buffer , size);
            break;
```

```
        case 17: //UDP Protocol
            ++udp;
            print_udp_packet(buffer , size);
            break;

        default: //Some Other Protocol like ARP etc.
            ++others;
            break;
    }
    printf("TCP : %d   UDP : %d   ICMP : %d   IGMP : %d   Others : %d   Total : %d\r", tcp , udp ,
icmp , igmp , others , total);
}

void print_ethernet_header(const u_char *Buffer, int Size)
{
    struct ethhdr *eth = (struct ethhdr *)Buffer;

    fprintf(logfile , "\n");
    fprintf(logfile , "Ethernet Header\n");
    fprintf(logfile , "   |-Destination Address : %.2X-%.2X-%.2X-%.2X-%.2X-%.2X \n", eth-
>h_dest[0] , eth->h_dest[1] , eth->h_dest[2] , eth->h_dest[3] , eth->h_dest[4] , eth->h_dest[5] );
    fprintf(logfile , "   |-Source Address      : %.2X-%.2X-%.2X-%.2X-%.2X-%.2X \n", eth-
>h_source[0] , eth->h_source[1] , eth->h_source[2] , eth->h_source[3] , eth->h_source[4] , eth-
>h_source[5] );
    fprintf(logfile , "   |-Protocol            : %u \n",(unsigned short)eth->h_proto);
}

void print_ip_header(const u_char * Buffer, int Size)
{
    print_ethernet_header(Buffer , Size);

    unsigned short iphdrlen;

    struct iphdr *iph = (struct iphdr *)(Buffer  + sizeof(struct ethhdr) );
    iphdrlen =iph->ihl*4;

    memset(&source, 0, sizeof(source));
    source.sin_addr.s_addr = iph->saddr;

    memset(&dest, 0, sizeof(dest));
    dest.sin_addr.s_addr = iph->daddr;

    fprintf(logfile , "\n");
    fprintf(logfile , "IP Header\n");
    fprintf(logfile , "   |-IP Version        : %d\n",(unsigned int)iph->version);
    fprintf(logfile , "   |-IP Header Length  : %d DWORDS or %d Bytes\n",(unsigned int)iph->ihl,
((unsigned int)(iph->ihl))*4);
    fprintf(logfile , "   |-Type Of Service   : %d\n",(unsigned int)iph->tos);
    fprintf(logfile , "   |-IP Total Length   : %d  Bytes(Size of Packet)\n",ntohs(iph->tot_len));
    fprintf(logfile , "   |-Identification    : %d\n",ntohs(iph->id));
    fprintf(logfile , "   |-TTL      : %d\n",(unsigned int)iph->ttl);
    fprintf(logfile , "   |-Protocol : %d\n",(unsigned int)iph->protocol);
    fprintf(logfile , "   |-Checksum : %d\n",ntohs(iph->check));
    fprintf(logfile , "   |-Source IP        : %s\n" , inet_ntoa(source.sin_addr) );
    fprintf(logfile , "   |-Destination IP   : %s\n" , inet_ntoa(dest.sin_addr) );
}
```

```
void print_tcp_packet(const u_char * Buffer, int Size)
{
    unsigned short iphdrlen;

    struct iphdr *iph = (struct iphdr *)( Buffer  + sizeof(struct ethhdr) );
    iphdrlen = iph->ihl*4;

    struct tcphdr *tcph=(struct tcphdr*)(Buffer + iphdrlen + sizeof(struct ethhdr));

    int header_size =  sizeof(struct ethhdr) + iphdrlen + tcph->doff*4;

    fprintf(logfile , "\n\n***********************TCP Packet*************************\n");

    print_ip_header(Buffer,Size);

    fprintf(logfile , "\n");
    fprintf(logfile , "TCP Header\n");
    fprintf(logfile , "   |-Source Port      : %u\n",ntohs(tcph->source));
    fprintf(logfile , "   |-Destination Port : %u\n",ntohs(tcph->dest));
    fprintf(logfile , "   |-Sequence Number    : %u\n",ntohl(tcph->seq));
    fprintf(logfile , "   |-Acknowledge Number : %u\n",ntohl(tcph->ack_seq));
    fprintf(logfile , "   |-Header Length      : %d DWORDS or %d BYTES\n" ,(unsigned int)tcph-
>doff,(unsigned int)tcph->doff*4);
    fprintf(logfile , "   |-Urgent Flag          : %d\n",(unsigned int)tcph->urg);
    fprintf(logfile , "   |-Acknowledgement Flag : %d\n",(unsigned int)tcph->ack);
    fprintf(logfile , "   |-Push Flag            : %d\n",(unsigned int)tcph->psh);
    fprintf(logfile , "   |-Reset Flag           : %d\n",(unsigned int)tcph->rst);
    fprintf(logfile , "   |-Synchronise Flag     : %d\n",(unsigned int)tcph->syn);
    fprintf(logfile , "   |-Finish Flag          : %d\n",(unsigned int)tcph->fin);
    fprintf(logfile , "   |-Window         : %d\n",ntohs(tcph->window));
    fprintf(logfile , "   |-Checksum       : %d\n",ntohs(tcph->check));
    fprintf(logfile , "   |-Urgent Pointer : %d\n",tcph->urg_ptr);
    fprintf(logfile , "\n");
    fprintf(logfile , "                        DATA Dump                         ");
    fprintf(logfile , "\n");

    fprintf(logfile , "IP Header\n");
    PrintData(Buffer,iphdrlen);

    fprintf(logfile , "TCP Header\n");
    PrintData(Buffer+iphdrlen,tcph->doff*4);

    fprintf(logfile , "Data Payload\n");
    PrintData(Buffer + header_size , Size - header_size );

    fprintf(logfile , "\n###########################################################");
}

void print_udp_packet(const u_char *Buffer , int Size)
{

    unsigned short iphdrlen;

    struct iphdr *iph = (struct iphdr *)(Buffer +  sizeof(struct ethhdr));
    iphdrlen = iph->ihl*4;
```

```
    struct udphdr *udph = (struct udphdr*)(Buffer + iphdrlen  + sizeof(struct ethhdr));

    int header_size =  sizeof(struct ethhdr) + iphdrlen + sizeof udph;

    fprintf(logfile , "\n\n***********************UDP Packet*************************\n");

    print_ip_header(Buffer,Size);

    fprintf(logfile , "\nUDP Header\n");
    fprintf(logfile , "   |-Source Port      : %d\n" , ntohs(udph->source));
    fprintf(logfile , "   |-Destination Port : %d\n" , ntohs(udph->dest));
    fprintf(logfile , "   |-UDP Length       : %d\n" , ntohs(udph->len));
    fprintf(logfile , "   |-UDP Checksum     : %d\n" , ntohs(udph->check));

    fprintf(logfile , "\n");
    fprintf(logfile , "IP Header\n");
    PrintData(Buffer , iphdrlen);

    fprintf(logfile , "UDP Header\n");
    PrintData(Buffer+iphdrlen , sizeof udph);

    fprintf(logfile , "Data Payload\n");

    //Move the pointer ahead and reduce the size of string
    PrintData(Buffer + header_size , Size - header_size);

    fprintf(logfile , "\n#####################################################################");
}

void print_icmp_packet(const u_char * Buffer , int Size)
{
    unsigned short iphdrlen;

    struct iphdr *iph = (struct iphdr *)(Buffer  + sizeof(struct ethhdr));
    iphdrlen = iph->ihl * 4;

    struct icmphdr *icmph = (struct icmphdr *)(Buffer + iphdrlen  + sizeof(struct ethhdr));

    int header_size =  sizeof(struct ethhdr) + iphdrlen + sizeof icmph;

    fprintf(logfile , "\n\n***********************ICMP Packet*************************\
n");

    print_ip_header(Buffer , Size);

    fprintf(logfile , "\n");

    fprintf(logfile , "ICMP Header\n");
    fprintf(logfile , "   |-Type : %d",(unsigned int)(icmph->type));

    if((unsigned int)(icmph->type) == 11)
    {
        fprintf(logfile , "  (TTL Expired)\n");
    }
    else if((unsigned int)(icmph->type) == ICMP_ECHOREPLY)
    {
        fprintf(logfile , "  (ICMP Echo Reply)\n");
```

```
    }

    fprintf(logfile , "   |-Code : %d\n",(unsigned int)(icmph->code));
    fprintf(logfile , "   |-Checksum : %d\n",ntohs(icmph->checksum));
    //fprintf(logfile , "   |-ID       : %d\n",ntohs(icmph->id));
    //fprintf(logfile , "   |-Sequence : %d\n",ntohs(icmph->sequence));
    fprintf(logfile , "\n");

    fprintf(logfile , "IP Header\n");
    PrintData(Buffer,iphdrlen);

    fprintf(logfile , "UDP Header\n");
    PrintData(Buffer + iphdrlen , sizeof icmph);

    fprintf(logfile , "Data Payload\n");

    //Move the pointer ahead and reduce the size of string
    PrintData(Buffer + header_size , (Size - header_size) );

    fprintf(logfile , "\n#####################################################################");
}

void PrintData (const u_char * data , int Size)
{
   //u_char *ptr=(u_char *)data;
 //  const char* S1 = reinterpret_cast<const char*>(data);
  // fprintf(logfile,"%s\n",S1);
    int i , j;
    for(i=0 ; i < Size ; i++)
    {
        if( i!=0 && i%16==0)   //if one line of hex printing is complete...
        {
            fprintf(logfile , "         ");
            for(j=i-16 ; j<i ; j++)
            {
                if(data[j]>=32 && data[j]<=128)
                    fprintf(logfile , "%c",(unsigned char)data[j]); //if its a number or alphabet

                else fprintf(logfile , "."); //otherwise print a dot
            }
            fprintf(logfile , "\n");
        }

        if(i%16==0) fprintf(logfile , "   ");
            fprintf(logfile , " %02X",(unsigned int)data[i]);

        if( i==Size-1)  //print the last spaces
        {
            for(j=0;j<15-i%16;j++)
            {
              fprintf(logfile , "   "); //extra spaces
            }

            fprintf(logfile , "         ");

            for(j=i-i%16 ; j<=i ; j++)
            {
```

```
            if(data[j]>=32 && data[j]<=128)
            {
              fprintf(logfile , "%c",(unsigned char)data[j]);
            }
            else
            {
              fprintf(logfile , ".");
            }
          }

          fprintf(logfile ,  "\n" );
        }
    }
}


//usage:
g++ filename.cpp -o file -lpcap
sudo ./file
-------------------------------------------------------------------------------------------------------------------
-------
                                RPC
-------------------------------------------------------------------------------------------------------------------
-------
  .x file

  struct arg{

};
program ADD_PROG{
          version ADD_VERS{
                    return_type fun(arg)=1;
          }=1;
}=0x23451111;

usage:
rpcgen -a -C file.x
make -f Makefile.file
./file_server
./file_client localhost args
```