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);

}

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)

{

}