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);

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 );

**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)

{

}

**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);