##### NETWORK PROGRAMMING

**CSX-354**

**LAB PRACTICALS RECORD**

**COMPUTER SCIENCE AND ENGINEERING**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**Dr. B R AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY**

**JALANDHAR – 144011, PUNJAB (INDIA)**

###### **Submitted To: Submitted By:**

Mr. MANOJ KUMAR SHRAVAN KUMAR

Asst. Professor 13103045

Department of CSE 6th Semester

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.No** | **Programs Name** | **Page No** | **Date** | **Signature** |
| 01. | Client Server Communication using TCP | 2 | 12/01/2016 |  |
| 02. | TCP iterative client and server application to reverse given input | 6 | 19/01/2016 |  |
| 03. | TCP Sockets Date and Time Server | 13 | 09/02/2016 |  |
| 04. | TCP client and server application to transfer a file | 18 | 16/02/2016 |  |
| 05. | UDP client and server application to transfer a file | 24 | 01/03/2016 |  |
| 06. | Creation of one way pipe in single process | 27 | 08/03/2016 |  |
| 07. | To make a Server client for receiving and sending messages using fifo | 29 | 15/03/2016 |  |
| 08. | Program to implement message queue (to transfer a file or any) | 33 | 22/03/2016 |  |
| 09. | To perform Semaphore Operations | 36 | 12/04/2016 |  |
| 10. | DNS Server to resolve a given host name | 42 | 19/04/2016 |  |

**INDEX:**

**PROGRAM-1**

Client Server Communication using TCP

**Server Program :**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

void error(const char \*msg)

{

perror(msg);

exit(1);

}

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

{

int sockfd, newsockfd, portno;

socklen\_t clilen;

char buffer[256];

struct sockaddr\_in serv\_addr, cli\_addr;

int n;

if (argc < 2) {

fprintf(stderr,"ERROR, no port provided\n");

exit(1);

}

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

if (sockfd < 0)

error("ERROR opening socket");

bzero((char \*) &serv\_addr, sizeof(serv\_addr));

portno = atoi(argv[1]);

serv\_addr.sin\_family = AF\_INET;

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

serv\_addr.sin\_port = htons(portno);

if (bind(sockfd, (struct sockaddr \*) &serv\_addr,

sizeof(serv\_addr)) < 0)

error("ERROR on binding");

listen(sockfd,5);

clilen = sizeof(cli\_addr);

newsockfd = accept(sockfd,

(struct sockaddr \*) &cli\_addr,

&clilen);

if (newsockfd < 0)

error("ERROR on accept");

bzero(buffer,256);

n = read(newsockfd,buffer,255);

if (n < 0) error("ERROR reading from socket");

printf("Here is the message: %s\n",buffer);

n = write(newsockfd,"I got your message",18);

if (n < 0) error("ERROR writing to socket");

close(newsockfd);

close(sockfd);

return 0;

}

**Client Program :**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

void error(const char \*msg)

{

perror(msg);

exit(0);

}

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

{

int sockfd, portno, n;

struct sockaddr\_in serv\_addr;

struct hostent \*server;

char buffer[256];

if (argc < 3) {

fprintf(stderr,"usage %s hostname port\n", argv[0]);

exit(0);

}

portno = atoi(argv[2]);

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

if (sockfd < 0)

error("ERROR opening socket");

server = gethostbyname(argv[1]);

if (server == NULL) {

fprintf(stderr,"ERROR, no such host\n");

exit(0);

}

bzero((char \*) &serv\_addr, sizeof(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

bcopy((char \*)server->h\_addr,

(char \*)&serv\_addr.sin\_addr.s\_addr,

server->h\_length);

serv\_addr.sin\_port = htons(portno);

if (connect(sockfd,(struct sockaddr \*) &serv\_addr,sizeof(serv\_addr)) < 0)

error("ERROR connecting");

printf("Please enter the message: ");

bzero(buffer,256);

fgets(buffer,255,stdin);

n = write(sockfd,buffer,strlen(buffer));

if (n < 0)

error("ERROR writing to socket");

bzero(buffer,256);

n = read(sockfd,buffer,255);

if (n < 0)

error("ERROR reading from socket");

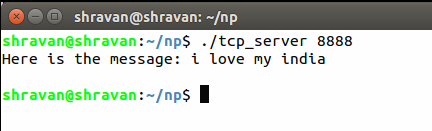
printf("%s\n",buffer);

close(sockfd);

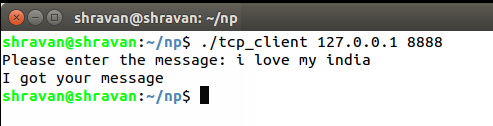
return 0;

}

**Output:**

****

**Figure:Server**



**Figure:Client**

**PROGRAM 2**

Design TCP iterative client and server application to reverse a given input string.

**Server Program :**

#include<stdio.h>

#include<string.h>

#include<sys/un.h>

#include<unistd.h>

#include<pthread.h>

#include<arpa/inet.h>

#include<netinet/in.h>

#define size 1024

#define PORT 9925

char rbuf[size];

void \*retval;

int server\_fd, c\_fd, ret, c\_len, bytes, ret;

struct sockaddr\_in server\_addrobj, c\_addrobj;

void strrev(char \*s)

{

int i, j;

char t;

for(i=0; s[i]!='\0'; i++);

j=0;i--;

while(j<i)

{

t = s[i];

s[i] = s[j];

s[j] = t;

j++;i--;

}

}

int main()

{

int true = 1;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Unlink \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unlink("SOCKET SERVER");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Socket \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

setsockopt(server\_fd, SOL\_SOCKET, SO\_REUSEADDR, &true, sizeof(int));

if(server\_fd == -1)

perror("Socket creation failed!!!\n");

else

printf("Socket created successfully\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Bind \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

server\_addrobj.sin\_family = AF\_INET;

server\_addrobj.sin\_port = htons(PORT);

server\_addrobj.sin\_addr.s\_addr = htonl(INADDR\_ANY);

ret = bind(server\_fd, (const struct sockaddr \*)&server\_addrobj, sizeof(server\_addrobj));

if(ret==-1)

perror("Binding failed !!!!\n");

else

printf("Binding Successfull\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Listen \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ret = listen(server\_fd, 5);

if(ret==-1)

perror("Listening failed!!!\n");

else

printf("Listening successfull\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Accept \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

while(1)

{

bzero(rbuf, size);

printf("\nWaiting for client request.......\n");

c\_len=sizeof(c\_addrobj);

c\_fd = accept(server\_fd, (struct sockaddr \*)&c\_addrobj, &c\_len);

if(c\_fd == -1)

perror("Accept system call failed!!!\n");

else

printf("Connection established successfully \n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Conversation\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bzero(rbuf, size);

while(strncmp(rbuf, "end", 3) != 0 )

{

bzero(rbuf, size);

bytes = recv(c\_fd, rbuf, size, 0);

if(bytes == -1)

perror("Writing by the server failed!!!\n");

strrev(rbuf);

bytes = send(c\_fd, rbuf, sizeof(rbuf), 0);

if(bytes == -1)

perror("\nWriting by the server failed!!!\n");

else

printf("\nString reversed and sent \n");

}

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Conversation Over \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

close(c\_fd);

}

close(server\_fd);

return 0;

}

**Client Program :**

#include<stdio.h>

#include<sys/types.h>

#include<sys/socket.h>

#include<string.h>

#include<sys/un.h>

#include<unistd.h>

#include<arpa/inet.h>

#include<netinet/in.h>

#define size 1024

#define PORT 9925

char rbuf[size], buf[size];

int client\_fd, ret, bytes, ret;

struct sockaddr\_in client\_addrobj;

void \*retval;

int main()

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Socket \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

if(client\_fd == -1)

perror("Socket creation failed !!!\n");

else

printf("Socket created successfully \n\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Naming the socket \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

client\_addrobj.sin\_family = AF\_INET;

client\_addrobj.sin\_port = htons(PORT);

client\_addrobj.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Connect \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ret = connect(client\_fd, (struct sockaddr \*)&client\_addrobj, sizeof(client\_addrobj) );

if(ret == -1)

perror("Connection failed !!!\n");

else

printf("Connection established successfully \n\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Conversation \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bzero(rbuf, size);

while(strncmp(rbuf, "end", 3) != 0 )

{

bzero(rbuf, size);

printf("\nEnter the message from client....\n");

fgets(rbuf, size, stdin);

bytes = send(client\_fd, rbuf, sizeof(rbuf), 0);

if(bytes == -1)

perror("Writing by the client failed!!!\n");

else

printf("String sent\n");

bzero(buf, size);

bytes = recv(client\_fd, buf, size, 0);

if(bytes == -1)

perror("Reading from server failed!!!\n");

else

printf("\nYou got the reversed string from server :\n %s \n", buf);

}

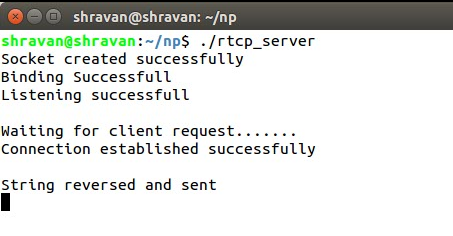
printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Conversation Over \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

close(client\_fd);

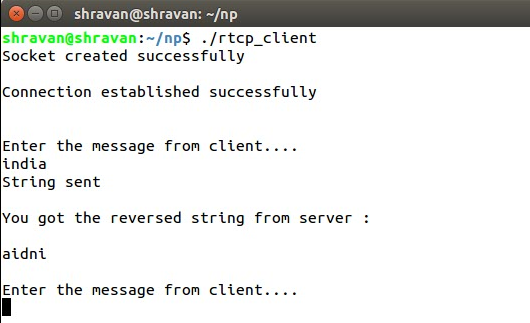
return 0;

}

**OUTPUT :**

****

**Figure:Server**

****

**Figure:Client**

**PROGRAM 3**

Program for date and time server using TCP sockets.

**Server Program :**

#include<stdio.h>

#include<string.h>

#include<sys/un.h>

#include<unistd.h>

#include<pthread.h>

#include<arpa/inet.h>

#include<netinet/in.h>

#include<time.h>

#define size 1024

#define PORT 9925

char str[size];

void \*retval;

int server\_fd, c\_fd, ret, c\_len, bytes, ret;

struct sockaddr\_in server\_addrobj, c\_addrobj;

time\_t tick;

int main()

{

int true = 1;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Unlink \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unlink("SOCKET SERVER");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Socket \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

setsockopt(server\_fd, SOL\_SOCKET, SO\_REUSEADDR, &true, sizeof(int));

if(server\_fd == -1)

perror("Socket creation failed!!!\n");

else

printf("Socket created successfully\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Bind \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

server\_addrobj.sin\_family = AF\_INET;

server\_addrobj.sin\_port = htons(PORT);

server\_addrobj.sin\_addr.s\_addr = htonl(INADDR\_ANY);

ret = bind(server\_fd, (const struct sockaddr \*)&server\_addrobj, sizeof(server\_addrobj));

if(ret==-1)

perror("Binding failed !!!!\n");

else

printf("Binding Successfull\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Listen \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ret = listen(server\_fd, 5);

if(ret==-1)

perror("Listening failed!!!\n");

else

printf("Listening successfull\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Accept \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

while(1)

{

printf("\nWaiting for client request.......\n");

c\_len=sizeof(c\_addrobj);

c\_fd = accept(server\_fd, (struct sockaddr \*)&c\_addrobj, &c\_len);

if(c\_fd == -1)

perror("Accept system call failed!!!\n");

else

printf("Connection established successfully \n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Conversation \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bzero(str, size);

tick=time(NULL);

snprintf(str,sizeof(str),"%s",ctime(&tick));

printf("%s",str);

write(c\_fd,str,100);

bytes = send(c\_fd, str, sizeof(str), 0);

if(bytes == -1)

perror("\nWriting by the server failed!!!\n");

close(c\_fd);

}

close(server\_fd);

return 0;

}

**Client Program :**

#include<stdio.h>

#include<sys/types.h>

#include<sys/socket.h>

#include<string.h>

#include<sys/un.h>

#include<unistd.h>

#include<arpa/inet.h>

#include<netinet/in.h>

#define size 1024

#define PORT 9925

char rbuf[size], buf[size];

int client\_fd, ret, bytes, ret;

struct sockaddr\_in client\_addrobj;

void \*retval;

int main()

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Socket \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

if(client\_fd == -1)

perror("Socket creation failed !!!\n");

else

printf("Socket created successfully \n\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Naming the socket \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

client\_addrobj.sin\_family = AF\_INET;

client\_addrobj.sin\_port = htons(PORT);

client\_addrobj.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Connect \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ret = connect(client\_fd, (struct sockaddr \*)&client\_addrobj, sizeof(client\_addrobj) );

if(ret == -1)

perror("Connection failed !!!\n");

else

printf("Connection established successfully \n\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Conversation \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bzero(buf, size);

bytes = recv(client\_fd, buf, size, 0);

if(bytes == -1)

perror("Reading from server failed!!!\n");

else

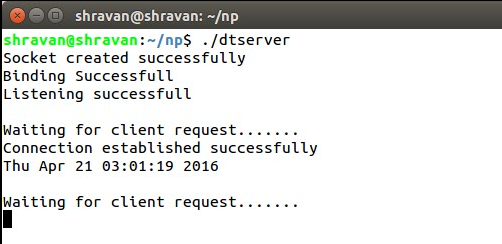
printf("\n %s \n", buf);

close(client\_fd);

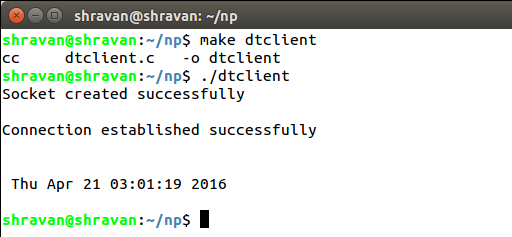
return 0;

}

**OUTPUT :**

****

**Figure:Server**

****

**Figure:Client**

**PROGRAM 4**

Design TCP client and server application to transfer a file.

**Server Program :**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<sys/socket.h>

#include<sys/types.h>

#include<sys/un.h>

#include<unistd.h>

#include<pthread.h>

#include<arpa/inet.h>

#include<netinet/in.h>

#define PORT 7777

#define SIZE 1024

void \* thread\_function(void \*);

int server\_fd, client\_fd, ret, c\_len;

struct sockaddr\_in server\_addrobj, client\_addrobj;

pthread\_t thread\_id;

char buf[SIZE];

FILE \*fp, \*fp1;

size\_t bytes;

int main()

{

int true = 1;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Unlink \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unlink("SOCKET SERVER");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Socket \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

setsockopt(server\_fd, SOL\_SOCKET, SO\_REUSEADDR, &true, sizeof(int));

if(server\_fd == -1)

{

perror("Socket creation failed!!!\n");

exit(0);

}

else

printf("Socket created successfully\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Binding \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

server\_addrobj.sin\_family = AF\_INET;

server\_addrobj.sin\_port = htons(PORT);

server\_addrobj.sin\_addr.s\_addr = htonl(INADDR\_ANY);

ret = bind(server\_fd, (const struct sockaddr \*)&server\_addrobj, sizeof(server\_addrobj));

if(ret == -1)

{

perror("Binding failed!!!!\n");

exit(1);

}

else

printf("Binding successful\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Listening \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ret = listen(server\_fd, 5);

if(ret == -1)

{

perror("Listening failed!!!!\n");

exit(2);

}

else

printf("Listening successful\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Accepting \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

while(1)

{

printf("\nWaiting for client request.......\n");

c\_len = sizeof(client\_addrobj);

client\_fd = accept(server\_fd, (struct sockaddr \*)&client\_addrobj, &c\_len);

if(ret == -1)

{

perror("Accept system call failed!!!!\n");

exit(3);

}

else

printf("Connection established successfully\n");

fp = fopen("abc.txt", "r");

printf("\nSending file .......\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Sending the file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

while(!feof(fp))

{

bytes = fread(buf, sizeof(char), sizeof(buf), fp);

if(bytes < 0)

{

perror("Error");

exit(0);

}

ret = send(client\_fd, buf, sizeof(buf), 0);

if(ret < 0)

perror("Sending failed\n");

}

fclose(fp);

close(client\_fd);

}

close(server\_fd);

return 0;

}

**Client Program :**

#include<stdio.h>

#include<sys/types.h>

#include<sys/socket.h>

#include<string.h>

#include<sys/un.h>

#include<unistd.h>

#include<stdlib.h>

#include<pthread.h>

#include<arpa/inet.h>

#include<netinet/in.h>

#define SIZE 1024

#define PORT 7777

int client\_fd, ret, true=1;

size\_t bytes;

struct sockaddr\_in server\_addrobj, client\_addrobj;

char buf[SIZE];

int main()

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Socket \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

setsockopt(client\_fd, SOL\_SOCKET, SO\_REUSEADDR, &true, sizeof(int));

if(client\_fd == -1)

perror("Socket creation failed !!!\n");

else

printf("Socket created successfully \n\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Binding \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

client\_addrobj.sin\_family = AF\_INET;

client\_addrobj.sin\_port = htons(PORT);

client\_addrobj.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Connecting \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ret = connect(client\_fd, (struct sockaddr \*)&client\_addrobj, sizeof(client\_addrobj) );

if(ret == -1)

perror("Connection failed !!!\n");

else

printf("Connection established successfully \n\n");

FILE \*fp = fopen("abc1.txt", "w");

bzero(buf, SIZE);

printf("\n Receiving file ........\n");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Receiving the file \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

while( bytes = recv(client\_fd, buf, sizeof(buf), 0) > 0)

{

fprintf(fp, "%s", buf);

bzero(buf, SIZE);

}

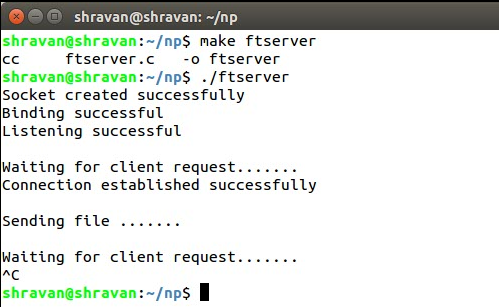
fclose(fp);

close(client\_fd);

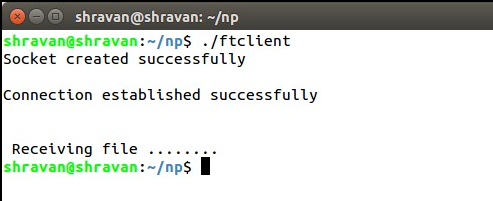
return 0;

}

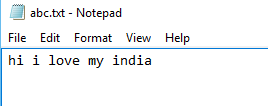
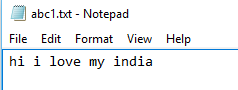
**Output:**

****

**Figure:Server**

****

**Figure:Client**

**** ****

**File to be sent** **Received File**

**PROGRAM 5**

UDP client and server application to transfer a file

**Server Program:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

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

{

int sockfd,newsockfd;

socklen\_t clilen;

char file[100],line[100];

struct sockaddr\_in serv\_addr,cli\_addr;

bzero((char\*) &serv\_addr,sizeof(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(atoi(argv[1]));

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

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

bind(sockfd,(struct sockaddr\*) &serv\_addr,sizeof(serv\_addr));

listen(sockfd,5);

clilen = sizeof(cli\_addr);

if((newsockfd = accept(sockfd,(struct sockaddr\*) &cli\_addr,&clilen))<0) error("Couldn't accept");

recvfrom(newsockfd,file,100,0,(struct sockaddr\*) &cli\_addr,&clilen);

FILE \*fp;

if((fp = fopen(file,"r")) < 0 ) error("Couldn't open file");

printf("Contents of file are:\n");

while(fgets(line,100,fp)!=NULL)

{

printf("%s",line);

}

fclose(fp);

return 0;

}

**Client Program:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

void error(char \*msg)

{

perror(msg);

exit(1);

}

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

{

int sockfd;

char file[100];

printf("Enter the file name.\n");

scanf("%s",file);

FILE\* fp;

if((fp = fopen(file,"r"))<0) error("File Couldn't found");

struct sockaddr\_in serv\_addr;

struct hostent \*server;

bzero((char\*) &serv\_addr,sizeof(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(atoi(argv[2]));

server = gethostbyname(argv[1]);

bcopy((char\*)server->h\_addr,(char\*) &serv\_addr.sin\_addr.s\_addr,server->h\_length);

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

if(connect(sockfd,(struct sockaddr \*) &serv\_addr,sizeof(serv\_addr)) < 0 ) error("Couldn't connect");

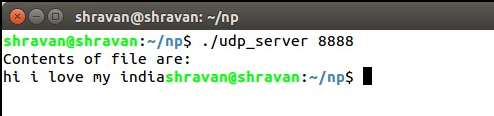
sendto(sockfd,file,strlen(file),0,(struct sockaddr \*) &serv\_addr,sizeof(serv\_addr));

printf("Done sending file. Exiting..");

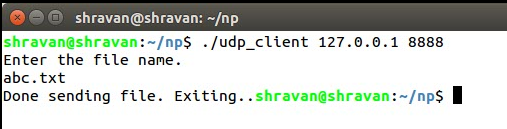
return 0;

}

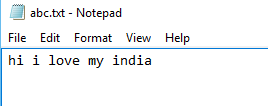
**Output:**

****

**Figure:Server**

****

**Figure:Client**

****

**File to be sent**

**PROGRAM 6**

Creation of a one way pipe in a single process.

**Program :**

#include<stdio.h>

#include<unistd.h>

#include<string.h>

int main()

{

char source[20]="Hello",dest[20];

int fd[2],ret,data\_bytes;

ret=pipe(fd);

if(ret==-1)

perror("Pipe creation failed!!!\n");

else

printf("Pipe created successfully!!!\n");

data\_bytes=write(fd[1],source,strlen(source));

if(data\_bytes==-1)

perror("Error!!\n");

else

printf("No. of bytes wrote = %d\n",data\_bytes);

memset(dest,'\0',sizeof(dest));

data\_bytes=read(fd[0],dest,sizeof(dest));

if(data\_bytes==-1)

perror("Error!!\n");

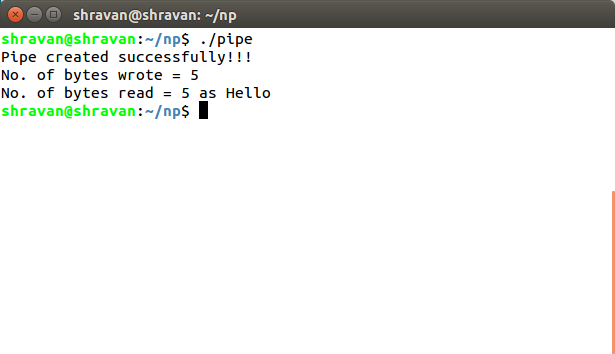
else

printf("No. of bytes read = %d as %s \n",data\_bytes,dest);

return 0;

}

**OUTPUT :**

****

**PROGRAM 7**

To make a Server client for receiving and sending messages using FIFO

**Program For Creating FIFO:**

#include<stdio.h>

#include<fcntl.h>

#include<stdlib.h>

main()

{

int file1,file2;

int fd;

char str[256];

char temp[4]="how";

char temp1[4];

file1 = mkfifo("fifo\_server",0666);

if(file1<0) {

printf("Unable to create a fifo");

exit(-1);

}

file2 = mkfifo("fifo\_client",0666);

if(file1<0) {

printf("Unable to create a fifo");

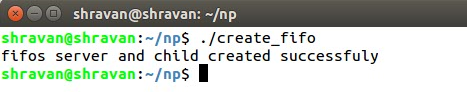
exit(-1);

}

printf("fifos server and child created successfuly\n");

}

**Output:**

****

**Server Program:**

#include<stdio.h>

#include<fcntl.h>

main()

{

FILE \*file1;

int fifo\_server,fifo\_client;

int choice;

char \*buf;

fifo\_server = open("fifo\_server",O\_RDWR);

if(fifo\_server<1) {

printf("Error opening file");

}

read(fifo\_server,&choice,sizeof(int));

sleep(10);

fifo\_client = open("fifo\_client",O\_RDWR);

if(fifo\_server<1) {

printf("Error opening file");

}

switch(choice) {

case 1:

buf="Linux";

write(fifo\_client,buf,10\*sizeof(char));

printf("\n Data sent to client \n");

break;

case 2:

buf="debian";

write(fifo\_client,buf,10\*sizeof(char));

printf("\nData sent to client\n");

break;

case 3:

buf="2.6.32";

write(fifo\_client,buf,10\*sizeof(char));

printf("\nData sent to client\n");

}

close(fifo\_server);

close(fifo\_client);

}

**Client Program:**

#include<stdio.h>

#include<fcntl.h>

#include<stdlib.h>

main()

{

FILE \*file1;

int fifo\_server,fifo\_client;

char str[256];

char \*buf;

int choice=1;

printf("Choose the request to be sent to server from options below");

printf("\n\t\t Enter 1 for O.S.Name \n \

Enter 2 for Distribution \n \

Enter 3 for Kernel version \n");

scanf("%d",&choice);

fifo\_server=open("fifo\_server",O\_RDWR);

if(fifo\_server < 0) {

printf("Error in opening file");

exit(-1);

}

write(fifo\_server,&choice,sizeof(int));

fifo\_client=open("fifo\_client",O\_RDWR);

if(fifo\_client < 0) {

printf("Error in opening file");

exit(-1);

}

buf=malloc(10\*sizeof(char));

read (fifo\_client,buf,10\*sizeof(char));

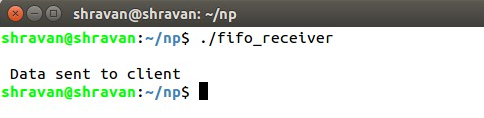
printf("\n \*\*\*Reply from server is %s\*\*\*\n",buf);

close(fifo\_server);

close(fifo\_client);

}

**Output:**

****

**Figure:Server**

****

**Figure:Client**

**PROGRAM 8**

Program to implement message queue (to transfer a file or any)

**Server Program:**

#include<sys/wait.h>

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<fcntl.h>

#include<sys/types.h>

#include<sys/msg.h>

#include<sys/ipc.h>

#include<errno.h>

#include<sys/stat.h>

struct msgbuf

{

long mtype;

char mtext[80];

};

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

{

int msqid;

int msgflg = IPC\_CREAT | 0666;

int key = 1234;

struct msgbuf rcvbuffer;

if(msqid = msgget(key,msgflg) < 0)

perror("msgget");

msgrcv(msqid,&rcvbuffer,80,1,0);

printf("Message received : %s\n",rcvbuffer.mtext);

return 0;

}

**Client Program:**

#include<sys/wait.h>

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<fcntl.h>

#include<sys/types.h>

#include<sys/msg.h>

#include<sys/ipc.h>

#include<errno.h>

#include<sys/stat.h>

struct msgbuf

{

long mtype;

char mtext[80];

};

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

{

int msqid;

int msgflg = IPC\_CREAT | 0666;

int key = 1234;

struct msgbuf sbuf;

if(msqid = msgget(key,msgflg) < 0)

perror("msgget");

sbuf.mtype = 1;

printf("Enter a message:\n");

scanf("%s",sbuf.mtext);

int len = strlen(sbuf.mtext)+1;

msgsnd(msqid,&sbuf,len,IPC\_NOWAIT);

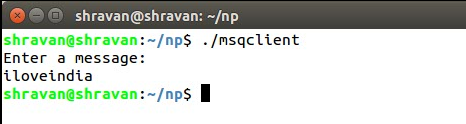
return 0;

}

**Output:**

****

**Figure:Server**

****

**Figure:Client**

**PROGRAM 9**

To perform Semaphore Operations

**Semaphore Server Program:**

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <fcntl.h>

#include <sys/sem.h>

#include <unistd.h>

#include <string.h>

#include <sys/wait.h>

#include <sys/types.h>

#include <errno.h>

#include "common.h"

#define CLIENT\_PATH\_BUFSIZE 255

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

key\_t sem\_key;

int sem\_id;

int sem\_fd;

char client\_exe[CLIENT\_PATH\_BUFSIZE];

int dir\_len;

int i;

struct sembuf sop;

int pid;

int status;

sem\_key = ftok("./sem\_server.c", 42);

// Write the key to a file for children to pick it up

sem\_fd = open(SEM\_KEY\_FILE, O\_WRONLY | O\_TRUNC | O\_EXCL | O\_CREAT, 0644);

if (sem\_fd < 0) {

perror("Could not open sem.key");

exit(1);

}

// Actual write of the key

if (write(sem\_fd, &sem\_key, sizeof(key\_t)) < 0) {

perror("Could not write key to file");

exit(2);

}

// Done with the key

close(sem\_fd);

// Create the semaphore

sem\_id = semget(sem\_key, 1, IPC\_CREAT | IPC\_EXCL | 0600);

if (sem\_id < 0) {

perror("Could not create sem");

unlink(SEM\_KEY\_FILE);

exit(3);

}

if (semctl(sem\_id, 0, SETVAL, 0) < 0) {

perror("Could not set value of semaphore");

exit(4);

}

// Now create some clients

// First create the path to the client exec

getcwd(client\_exe, CLIENT\_PATH\_BUFSIZE);

dir\_len = strlen(client\_exe);

strcpy(client\_exe + dir\_len, "/sem\_client");

printf("%s\n", client\_exe);

for (i = 0; i < 5; ++i) {

if ((pid = fork()) < 0) {

perror("Could not fork, please create clients manually");

}

else if (pid == 0) {

// We're in the child process, start a client

execl(client\_exe, "sem\_client", (char\*)0);

\_exit(127);

}

}

printf("Done creating clients, sleeping for a while\n");

sleep(5);

printf("Increasing sem count\n");

sop.sem\_num = 0;

sop.sem\_op = 1;

sop.sem\_flg = 0;

if (semop(sem\_id, &sop, 1)) {

perror("Could not increment semaphore");

exit(5);

}

// Wait for all children to finish

for (;;) {

// Remove the zombie process, and get the pid and return code

pid = wait(&status);

if (pid < 0) {

if (errno == ECHILD) {

printf("All children have exited\n");

break;

}

else {

perror("Could not wait");

}

}

else {

printf("Child %d exited with status %d\n", pid, status);

}

}

// Delete semaphore and file

if (unlink(SEM\_KEY\_FILE) < 0) {

perror("Could not unlink key file");

}

if (semctl(sem\_id, 0, IPC\_RMID) < 0) {

perror("Could not delete semaphore");

}

exit(0);

}

**Semaphore Client Program:**

#include <stdio.h>

#include <stdlib.h>

#include <sys/sem.h>

#include <fcntl.h>

#include <unistd.h>

#include "common.h"

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

int sem\_fd;

key\_t sem\_key;

int sem\_id;

int i;

struct sembuf sop;

// Recover the sem\_key from file

sem\_fd = open(SEM\_KEY\_FILE, O\_RDONLY);

if (sem\_fd < 0) {

perror("Could not open sem key for reading");

exit(1);

}

// Technically speaking, the read could read less than sizeof(key\_t)

// Which would be wrong.

// But in our case, it is not likely to happen...

if (read(sem\_fd, &sem\_key, sizeof(key\_t)) != sizeof(key\_t)) {

perror("Error reading the semaphore key");

exit(2);

}

// Done getting the semaphore key

close(sem\_fd);

// Now obtain the (hopefully) existing sem

sem\_id = semget(sem\_key, 0, 0);

if (sem\_id < 0) {

perror("Could not obtain semaphore");

exit(3);

}

for (i = 0; i < 5; ++i) {

sop.sem\_num = 0;

sop.sem\_op = -1;

sop.sem\_flg = SEM\_UNDO;

printf("Client #%d waiting\n", getpid());

semop(sem\_id, &sop, 1);

printf("Client #%d acquired. Sleeping\n", getpid());

sleep(1);

printf("Client #%d releasing\n", getpid());

sop.sem\_op = 1;

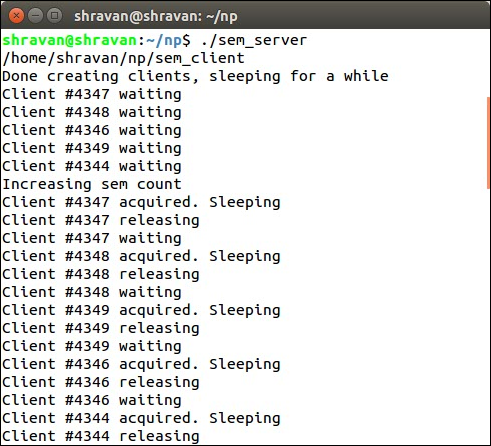
semop(sem\_id, &sop, 1);

}

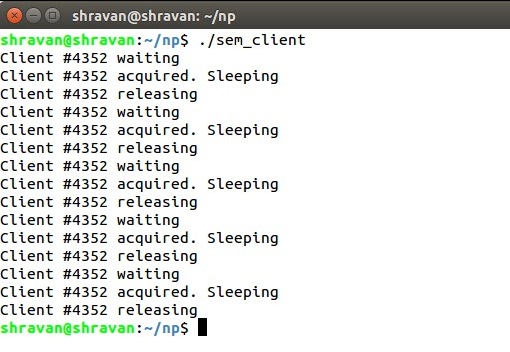
exit(0);

}

**Output:**

****

**Figure:Server**

****

**Figure:Client**

**PROGRAM 10**

DNS Server to resolve a given host name:

**Program:**

#include<stdio.h>

#include<netdb.h>

#include<arpa/inet.h>

#include<netinet/in.h>

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

{

char h\_name;

int h\_type;

struct hostent\* host;

struct in\_addr h\_addr;

if(argc!=2)

{

fprintf(stderr,"USAGE:nslookup\n");

}

if((host=gethostbyname(argv[1]))==NULL)

{

fprintf(stderr,"(mini)nslookup failed on %s\n",argv[1]);

}

h\_addr.s\_addr=\*((unsigned long\*)host->h\_addr\_list[0]);

printf("\nIP ADDRESS=%s\n",inet\_ntoa(h\_addr));

printf("\nHOST NAME=%s\n",host->h\_name);

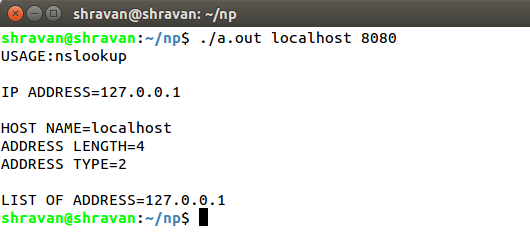
printf("ADDRESS LENGTH=%d\n",host->h\_length);

printf("ADDRESS TYPE=%d\n",host->h\_addrtype);

printf("\nLIST OF ADDRESS=%s\n",inet\_ntoa(h\_addr\_list[0]));

}

**Output:**

****