Experiment 7-a:Stop and wait
Program:
client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
typedef struct packet{
 char data[1024];
}Packet;
typedef struct frame{
 int frame_kind; //ACK:0, SEQ:1 FIN:2
 int sq_no;
 int ack;
 Packet packet;
}Frame;
int main(int argc, char **argv[]){
 if (argc != 2){
printf("Usage: %s <port>", argv[0]); exit(0);
}
int port = atoi(argv[1]);
int sockfd;
struct sockaddr_in serverAddr;
char buffer[1024];
socklen_t addr_size;
int frame_id = 0;
Frame frame_send;
Frame frame_recv;
int ack_recv = 1;
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
memset(&serverAddr, '\0',
sizeof(serverAddr));
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(port);
serverAddr.sin_addr.s_addr =
inet_addr("127.0.0.1");
while(1){
if(ack_recv == 1){
frame_send.sq_no = frame_id;
frame_send.frame_kind = 1;
frame_send.ack = 0;
printf("Enter Data: ");
scanf("%s", buffer);
24
strcpy(frame_send.packet.data, buffer);
sendto(sockfd,&frame_send, sizeof(Frame),
0, (struct
sockaddr*)&serverAddr,
sizeof(serverAddr));
printf("[+]Frame Send\n");
}
int addr_size = sizeof(serverAddr);
int f_recv_size = recvfrom(sockfd,
&frame_recv, sizeof(frame_recv), 0 ,(struct
sockaddr*)&serverAddr, &addr_size);
if( f_recv_size > 0 && frame_recv.sq_no ==
0 && frame_recv.ack ==
frame_id+1){
printf("[+]Ack Received\n");
ack_recv = 1;
}else{
printf("[-]Ack Not Received\n");
ack_recv = 0;
}
frame_id++;
}
close(sockfd);
return 0;
```

Stop and wait

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <unistd.h>
#include <arpa/inet.h>
typedef struct packet{
 char data[1024];
}Packet;
typedef struct frame{
 int frame_kind; //ACK:0, SEQ:1 FIN:2
 int sq_no;
 int ack;
 Packet packet;
}Frame;
int main(int argc, char** argv){
if (argc != 2){
printf("Usage: %s <port>", argv[0]);
exit(0);
25
}
int port = atoi(argv[1]);
int sockfd;
struct sockaddr_in serverAddr, newAddr;
char buffer[1024];
socklen_t addr_size;
int frame_id=0;
Frame frame_recv;
Frame frame_send;
sockfd = socket(AF_INET, SOCK_DGRAM,
0);
memset(&serverAddr, '\0',
sizeof(serverAddr));
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(port);
serverAddr.sin_addr.s_addr =
inet_addr("127.0.0.1");
bind(sockfd, (struct
sockaddr*)&serverAddr,
sizeof(serverAddr));
addr_size = sizeof(newAddr);
while(1){
int f_recv_size = recvfrom(sockfd,
&frame_recv, sizeof(Frame), 0, (struct
sockaddr*)&newAddr, &addr_size);
if (f_recv_size > 0 &&
frame_recv.frame_kind == 1 &&
frame_recv.sq_no ==
frame_id){
printf("[+]Frame Received: %s\n",
frame_recv.packet.data);
frame_send.sq_no = 0;
frame_send.frame_kind = 0;
frame_send.ack = frame_recv.sq_no + 1;
sendto(sockfd, &frame_send,
sizeof(frame_send), 0, (struct
sockaddr*)&newAddr, addr_size);
printf("[+]Ack Send\n");
}else{
printf("[+]Frame Not Received\n");
}
frame_id++;
}
close(sockfd);
return 0;
}
```

FTP

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <string.h>
int main()
{
 FILE *fp;
 int csd, n, ser, s, cli, cport, newsd;
 char name[100], rcvmsg[100], rcvg[100],
fname[100];
 struct sockaddr_in servaddr;
 printf("Enter the port address\n");
 scanf("%d", &cport);
 csd = socket(AF_INET, SOCK_STREAM, 0);
 if (csd < 0)
 {
 printf("Error....\n");
 exit(0);
 }
 else
 printf("Socket is created\n");
 servaddr.sin_family = AF_INET;
 servaddr.sin_addr.s_addr =
htonl(INADDR_ANY);
 servaddr.sin_port = htons(cport);
 if (connect(csd, (struct sockaddr
*)&servaddr, sizeof(servaddr)) < 0)
 printf("Error in connection\n");
 else
 printf("connected\n");
 printf("Enter the existing file name: ");
 scanf("%s", name);
 printf("Enter the new file name: ");
 scanf("%s", fname);
 fp = fopen(fname, "w");
 send(csd, name, sizeof(name), 0);
 while (1)
 {
 s = recv(csd, rcvg, 100, 0);
 rcvg[s] = '\0';
 if (strcmp(rcvg, "error") == 0)
 printf("File is not available\n");
38
 if (strcmp(rcvg, "completed") == 0)
 {
 printf("\nFile is transferred........\n");
 fclose(fp);
 close(csd);
 break;
 }
 else
 fputs(rcvg, stdout);
 fprintf(fp, "%s", rcvg);
 }
 return 0;
}
```

FTP server.c

```
#include <stdio.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
int main()
{
 FILE *fp;
 int sd, newsd, ser, n, a, cli, pid, bd, port,
clilen;
 char name[100], fileread[100],
fname[100], ch, file[100], rcv[100];
 struct sockaddr_in servaddr, cliaddr;
 printf("Enter the port address\n");
 scanf("%d", &port);
 sd = socket(AF_INET, SOCK_STREAM, 0);
 if (sd < 0)
 printf("Cant create\n");
 else
 printf("Socket is created\n");
 servaddr.sin_family = AF_INET;
 servaddr.sin_addr.s_addr =
htonl(INADDR_ANY);
 servaddr.sin_port = htons(port);
 a = sizeof(servaddr);
 bd = bind(sd, (struct sockaddr
*)&servaddr, a);
 if (bd < 0)
 printf("Cant bind\n");
 else
 printf("Binded\n");
 listen(sd, 5);
 clilen = sizeof(cliaddr);
39
 newsd = accept(sd, (struct sockaddr
*)&cliaddr, &clilen);
 if (newsd < 0)
 {
 printf("Cant accept\n");
 }
 else
 printf("Accepted\n");
 n = recv(newsd, rcv, 100, 0);
 rcv[n] = '\0';
 fp = fopen(rcv, "r");
 if (fp == NULL)
 {
 send(newsd, "error", 5, 0);
 close(newsd);
 }
 else
 {
 while (fgets(fileread, 100, fp))
 {
 if (send(newsd, fileread, sizeof(fileread), 0)
< 0)
 {
 printf("Can’t send file contents\n");
 exit(0);
 }
 sleep(1);
 }
 send(newsd, "completed", 9, 0);
 return (0);
 }
}
```

UDP Exp
Client

```
1. Create UDP socket
2. Send a request for time to the server
3. Receive the time from the server
4. Display the result
5. Server
1. Create a UDP socket
2. bind the port and address to the socket
3. while (1)
3.1 Receive time request from the client
3.2 create a child process using fork
If child process
3.2.1 Use time and time functions to find out cuurent time
3.2.2 Send the time as a string to the client
3.2.3 Exit
4. end of while
```

Stop and wait

```
1.Start the program
2. Generate a random number that gives the total number of frames to be transmitted.
3. Transmit the first frame
4. Receive the acknowledgement for the first frame
5. Transmit the next frame
6. Find the remaining frames to be sent.
7. If an acknowledgement is not received for a particular frame, retransmit that frame alone again.
8. Repeat the steps 5 to 7 till the number of remaining frames to be sent becomes zero
9. Stop the program.
```

Distance vector
Bellman Ford algoithm is applied.
Let dy (y) be the cost of the least cost path from node x to node y. Then Bellman Ford equation states that
da(y) = min [c(x,v) + di(y)]
where v is a neighbour of node x.
d.(y) is the cost of the least cost path from v to y. c(x,v) is the cost from r to neighbour v. The least cost path has a value equal to minimum of c(x,) + d(y) over all its neighbours v. The solution of Bellman Ford equation provides entries in node x's forwarding table.

Distance vector (DV) algorithm

```
At each node r
Initialization:
for all destinations y in N:
D, (y) = c(x,y) /* if y is not a neighbour of x, then c(x,y) = 00 */
for each neighbour w,
send distance vector D, = ( D(y): y in N] to w
loop:
for each y in N:
D‹(y) = min [ c(x,v) + Dw(y) ]
If D(y) changed for any destination y
send distance vector D‹ = (D(y) : y in N) to all neighbours.
```

FTP
Steps involved in writing the Server Process:

```
1. Create a socket using socket() system call with address family AF_INET, type SOCK_STREAM and default protocol.
2.Bind server's address and port using bind() system call.
3. Wait for client connection to complete accepting connections using accept ) system call.
4. Receive the Clients file using rec() system call.
5. Using *gets(char *str, int n, FILE *stream) function, we read a line of text from the specified stream and stores it into the string pointed to by str. It stops when either (n-1) characters are read, or when the end-of-file is reached.
6. On successful execution i.e. when file pointer reaches end of file, file transfer "completed" message is sent by the server to the accepted client connection using newsd, socket file descriptor.
```

Steps involved in writing the Client Process:

```
1. Create a socket system call with address family AF_INET, type SOCK_STREAM and default protocol.
2.Enter the client port id
3.Fill in the internet socket address structure (with server information).
4.Connect to the server address using connect system call.
5.Read the existing and new file name from user.
6.Send existing file to server using send system call
7.Receive feedback from server "Completed", regarding file transfer completion
8. Write "File is transferred" to standard output screen.
9.Close the socket connection and file pointer
```

UDP
Client.c:

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <stdlib.h>
main(int argc, char *argv[])
{
 int i, j, n;
 int sock_fd;
 struct sockaddr_in servaddr;
 char buff[100];
 if (argc != 3)
 {
 fprintf(stderr, "Usage: ./client
IPaddress_of_server port\n");
 exit(1);
 }
 if ((sock_fd = socket(AF_INET,
SOCK_DGRAM, 0)) < 0)
 {
 printf("Cannot create socket\n");
 exit(1);
 }
 bzero((char *)&servaddr,
sizeof(servaddr));
 servaddr.sin_family = AF_INET;
 servaddr.sin_port = htons(atoi(argv[2]));
 inet_pton(AF_INET, argv[1],
&servaddr.sin_addr);
 n = sendto(sock_fd, "", 1, 0, (struct
sockaddr *)&servaddr, sizeof(servaddr));
 if (n < 0)
 {
 perror("error in sending");
 exit(1);
 }
 if ((n = recvfrom(sock_fd, buff,
sizeof(buff), 0, NULL, NULL)) == -1)
 {
 perror("read error from server:");
 exit(1);
 }
 printf(" the current time of the system
is %s\n", buff);
}
```

UDP
Server.c:

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <stdlib.h>
#include <time.h>
main(int argc, char *argv[])
{
 int n;
 int sock_fd;
 int i, j, k;
 int childpid;
 char buffer[100];
 time_t curtime;
 struct sockaddr_in servaddr, cliaddr;
 int len = sizeof(cliaddr);
 if (argc != 2)
 {
 fprintf(stderr, "Usage: ./server port\n");
 exit(1);
 }
 if ((sock_fd = socket(AF_INET, SOCK_DGRAM,
0)) < 0)
 {
 printf("Cannot create socket\n");
 exit(1);
 }
 bzero((char *)&servaddr, sizeof(servaddr));
 servaddr.sin_family = AF_INET;
 servaddr.sin_port = htons(atoi(argv[1]));
 servaddr.sin_addr.s_addr =
htonl(INADDR_ANY);
 if (bind(sock_fd, (struct sockaddr *)&servaddr,
sizeof(servaddr)) < 0)
 {
 perror("bind failed:");
 exit(1);
 }
 while (1)
 {
 if ((n = recvfrom(sock_fd, buffer,
sizeof(buffer), 0, (struct sockaddr *)&cliaddr,
&len)) == -
1)
 {
 perror("size not received:");
22
 exit(1);
 }
 childpid = fork();
 if (childpid == 0)
 {
 time(&curtime);
 sprintf(buffer, "= %s", ctime(&curtime));
 n = sendto(sock_fd, buffer, sizeof(buffer), 0,
(struct sockaddr *)&cliaddr,
sizeof(cliaddr));
 if (n < 0)
 {
 perror("error in sending");
 exit(1);
 }
 exit(1);
 }
 }
}
```

Program:Distance vector

```
#include <stdio.h>
struct node
{
 unsigned dist[20];
 unsigned from[20];
} rt[10];
int main()
{
 int costmat[20][20];
 int nodes, i, j, k, count = 0;
 printf("\n Enter the number of nodes : ");
 scanf("%d", &nodes);
 printf("\n Enter 999 for infinity");
 printf("\n Enter the cost matrix :\n");
 for (i = 0; i < nodes; i++)
 {
 for (j = 0; j < nodes; j++)
 {
 scanf("%d", &costmat[i][j]);
 costmat[i][i] = 0;
 rt[i].dist[j] = costmat[i][j];
 rt[i].from[j] = j;
 }
 }
 do {
 count = 0;
 for (i = 0; i < nodes; i++)
 for (j = 0; j < nodes; j++) for (k = 0; k <
nodes; k++) if (rt[i].dist[j] > costmat[i][k] +
rt[k].dist[j])
 {
 rt[i].dist[j] = rt[i].dist[k] + rt[k].dist[j];
 rt[i].from[j] = k;
 count++;
 }
 } while (count != 0);
 for (i = 0; i < nodes; i++)
 {
 printf("\n\n For router %d\n", i + 1);
 for (j = 0; j < nodes; j++)
 {
 printf("\t\nnode %d via %d Distance %d ",
j + 1, rt[i].from[j] + 1, rt[i].dist[j]);
 }
 }
36
 printf("\n\n");
```