

```
if( n < 0)
{
perror("error in matrix 1 sending");
exit(1);
}
// SENDING MATRIX 1
n = sendto(sock_fd, matrix_1, sizeof(matrix_1),0, (struct sockaddr*)&servaddr,
sizeof(servaddr));
if( n < 0)
{
perror("error in matrix 1 sending");
exit(1);
}
// SENDING MATRIX 2
n = sendto(sock_fd, matrix_2, sizeof(matrix_2),0, (struct sockaddr*)&servaddr,
sizeof(servaddr));
if( n < 0)
{
perror("error in matrix 2 sending");
exit(1);
}
if((n=recvfrom(sock_fd, matrix_product, sizeof(matrix_product),0, NULL, NULL)) == -1)
{
perror("read error from server:");
exit(1);
}
printf("\n\nTHE PRODUCT OF MATRICES IS \n\n\n");
for( i=0; i < num_rows_1; i++)
{
for( j=0; j<num_cols_2; j++)
{
printf("%d ",matrix_product[i][j]);
}
printf("\n");
}
close(sock_fd);
}
```

Server Program

```
#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 n;
    int sock_fd;
    int i,j,k;
    int row_1, row_2, col_1, col_2;
    struct sockaddr_in servaddr, cliaddr;
    int len = sizeof(cliaddr);
    int matrix_1[10][10], matrix_2[10][10], matrix_product[10][10];
    int size[2][2];
    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);
    }
    // MATRICES RECEIVE
```

```
if((n = recvfrom(sock_fd, size, sizeof(size), 0, (struct sockaddr *)&cliaddr, &len)) == -1)
{
    perror("size not received:");
    exit(1);
}
// RECEIVE MATRIX 1
if((n = recvfrom(sock_fd, matrix_1, sizeof(matrix_1), 0, (struct sockaddr *)&cliaddr, &len)) ==
-1)
{
    perror("matrix 1 not received:");
    exit(1);
}
// RECEIVE MATRIX 2
if((n = recvfrom(sock_fd, matrix_2, sizeof(matrix_2), 0, (struct sockaddr *)&cliaddr, &len)) ==
-1)
{
    perror("matrix 2 not received:");
    exit(1);
}
row_1 = size[0][0];
col_1 = size[0][1];
row_2 = size[1][0];
col_2 = size[1][1];
for (i=0; i < row_1 ; i++)
for (j=0; j < col_2; j++)
{
    matrix_product[i][j] = 0;
}
for(i=0; i< row_1 ; i++)
for(j=0; j< col_2 ; j++)
for (k=0; k < col_1; k++)
{
    matrix_product[i][j] += matrix_1[i][k]*matrix_2[k][j];
}
n = sendto(sock_fd, matrix_product, sizeof(matrix_product),0, (struct sockaddr*)&cliaddr,
sizeof(cliaddr));
if( n < 0)
{
    perror("error in matrix product sending");
    exit(1);
}
```

```
}  
close(sock_fd);  
}
```

Output

Server

```
anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network_lab/expt2_udp  
File Edit View Search Terminal Help  
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt2_udp$ ./server 5300  
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt2_udp$
```

Client

```
anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network_lab/expt2_udp  
File Edit View Search Terminal Help  
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt2_udp$ ./client 127.0.0.1  
5300  
Enter the number of rows of first matrix  
3  
Enter the number of columns of first matrix  
4  
Enter the values row by row one on each line  
1 2 3 4  
5 6 7 8  
1 2 3 4  
Enter the number of rows of second matrix  
4  
Enter the number of columns of second matrix  
3  
Enter the values row by row one on each line  
1 2 3  
4 5 6  
7 8 9  
1 2 3  
  
THE PRODUCT OF MATRICES IS  
34 44 54  
86 112 138  
34 44 54  
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt2_udp$
```

Experiment 5

Simulate sliding window flow control protocols. (Stop and Wait, Go back N, Selective Repeat ARQ protocols)

sliding window flow control protocols

Flow control deals with problem that sender transmits frames faster than receiver can accept, and solution is to limit sender into sending no faster than receiver can handle Consider the simplex case: data is transmitted in one direction (Note although data frames are transmitted in one direction, frames are going in both directions, i.e. link is duplex) Stop and wait: sender sends one data frame, waits for acknowledgement (ACK) from receiver before proceeding to transmit next frame This simple flow control will break down if ACK gets lost or errors occur → sender may wait for ACK that never arrives

Go-back-n ARQ

The basic idea of go-back-n error control is: If frame i is damaged, receiver requests retransmission

of all frames starting from frame i

Notice that all possible cases of damaged frame and ACK / NAK must be taken into account

In selective-reject ARQ error control, the only frames retransmitted are those receive a NAK or which time out

1. Stop and Wait

Server.c

```
#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);
}
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;
}
```

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]);
```