Computer Networks

Name : M. Sai Saranya

Regno: 22BAI1471

Course Title : Computer Networks

Course code : BCSE308P

Slot : L45-46

Faculty : Dr Neelanarayanan V

S.No	Experiment Name	Date	Page No.	Marks
1.	Basic Network Configuration Commands	10-01-2024		
2.	Client-Server Application Echo	17-01-2024		
3.	IP Address Validation and Simple application of ATM using TCP	24-01-2024		
4.	CRC code generator using socket programming	07-02-2024		
5. a)	Echo programming using UDP	21-02-2024		
5. b)	IP address validation using UDP	21-02-2024		

S.No	Experiment Name	Date	Page No.	Marks
5. c)	ATM simulation using UDP	21-02-2024		
6.	Stop and wait ARQ	28-02-2024		
7.	Sliding window protocol	13-03-2024		

Experiment No. 7

Experiment Name: Sliding window protocol using either TCP or UDP programming

Date: 13-3-2024

Problem Statement

To simulate and understand the working of the sliding window protocol using TCP programming in Linux system.

Algorithm

Sliding window protocol is used. Two types of Sliding window protocol:

a. **Go-Back-N ARQ (Automatic Repeat Request)**:
   i. In this protocol, if any frame is **corrupted or lost**, all subsequent frames must be retransmitted.

ii. The sender's window size is denoted by **N** (e.g., Go-Back-8 means a sender window size of 8).

b. **Selective Repeat ARQ (Automatic Repeat Request)**:
   i. This protocol is used when there are **more errors** in the frame.
   ii. The sender window size is always equal to the receiver window size.
   iii. The sliding window size is greater than 1.
   iv. It sends only the specific frame that is lost or corrupted.

In this experiment Go back -N ARQ protocol using TCP socket programming is implemented

Client side code

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>


#define PORT 8080

#define WINDOW_SIZE 10

#define TIMEOUT_SEC 5


void error(const char *msg) {

    perror(msg);
```

```
    exit(1);
}

int main() {
    int sockfd;
    struct sockaddr_in servaddr;
    char buffer[1024];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        error("Socket creation failed");
    }

    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    servaddr.sin_port = htons(PORT);

    if (connect(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr)) != 0) {
        error("Connection failed");
    }
```

```
int base = 0;
int nextseqnum = 0;
int frame_count = 20;
int timeouts = 0;

while (base < frame_count) {
    // Sending the frames within window size
    while (nextseqnum < base + WINDOW_SIZE && nextseqnum < frame_count) {
        sprintf(buffer, "Frame %d", nextseqnum);
        send(sockfd, buffer, strlen(buffer), 0);
        printf("Sent: %s\n", buffer);
        nextseqnum++;
    }

    fd_set readfds;
    FD_ZERO(&readfds);
    FD_SET(sockfd, &readfds);
    struct timeval tv;
    tv.tv_sec = TIMEOUT_SEC;
    tv.tv_usec = 0;
```

```
// Wait for acknowledgment or timeout
int ready = select(sockfd + 1, &readfds, NULL, NULL, &tv);
if (ready < 0) {
    error("Select error");
} else if (ready == 0) {

    printf("Timeout occurred, resending frames from base %d\n", base);
    nextseqnum = base;
    timeouts++;
    continue;
}

// Receiving the acknowledgment
int ack;
recv(sockfd, &ack, sizeof(ack), 0);

if (ack >= base) {
    base = ack + 1;
    printf("Acknowledgment received for frame %d\n", ack);
    timeouts = 0;
```

```
        }
    }

    close(sockfd);
    return 0;
}
```

Server side code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PORT 8080
#define WINDOW_SIZE 10
#define RECEIVER_CAPACITY 5

void error(const char *msg) {
```

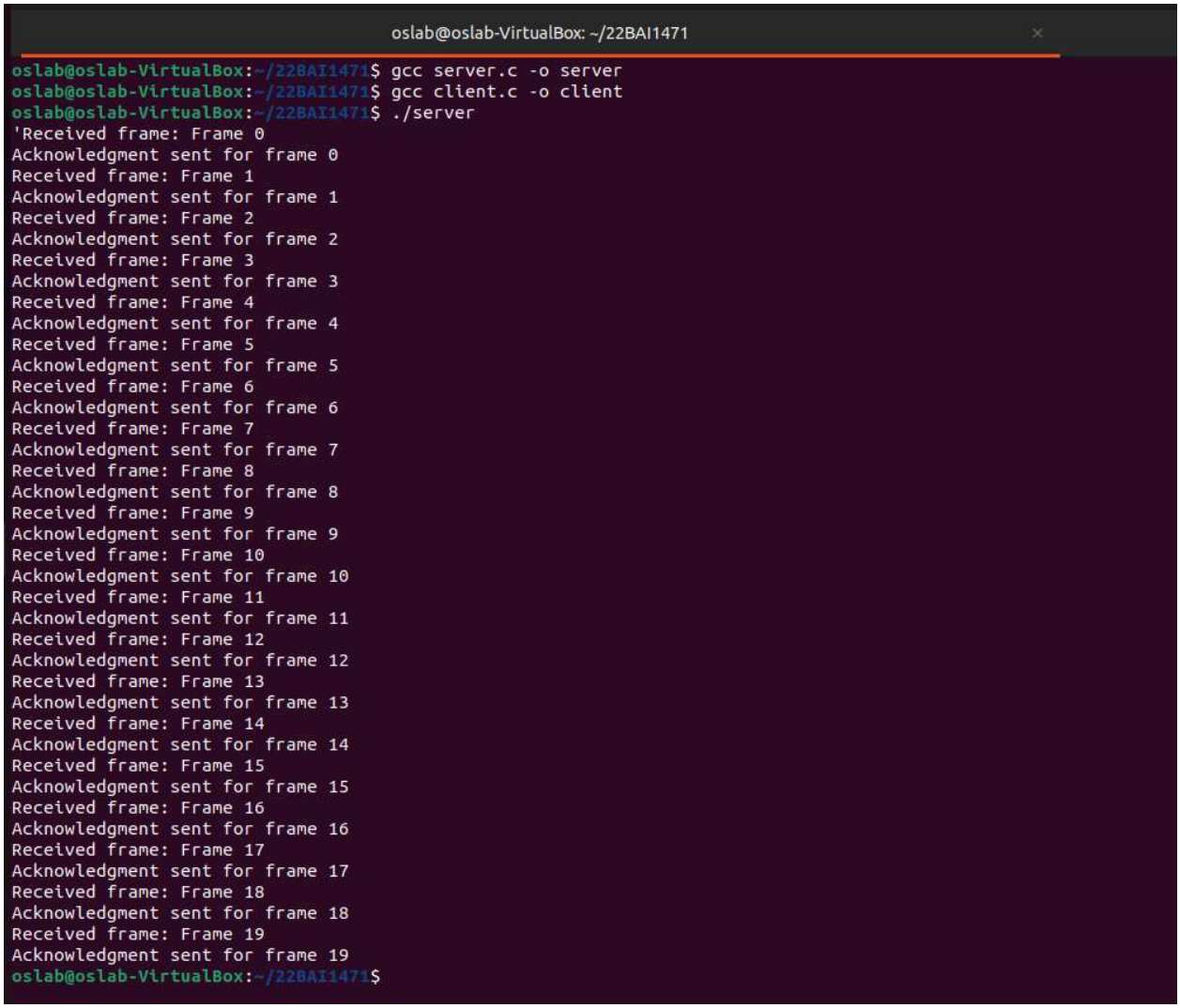
```
    perror(msg);
    exit(1);
}

int main() {
    int sockfd, connfd;
    struct sockaddr_in servaddr, cliaddr;
    char buffer[1024];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        error("Socket creation failed");
    }

    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    if ((bind(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr))) != 0) {
        error("Socket bind failed");
```

```
}

if ((listen(sockfd, 5)) != 0) {
    error("Listen failed");
}

socklen_t len = sizeof(cliaddr);
connfd = accept(sockfd, (struct sockaddr*)&cliaddr, &len);
if (connfd < 0) {
    error("Server accept failed");
}

int expectedseqnum = 0;
int last_frame_received = -1;

while (1) {
    recv(connfd, buffer, sizeof(buffer), 0);
    printf("Received frame: %s\n", buffer);

    if (expectedseqnum <= last_frame_received + RECEIVER_CAPACITY) {
        // Sending acknowledgment
```

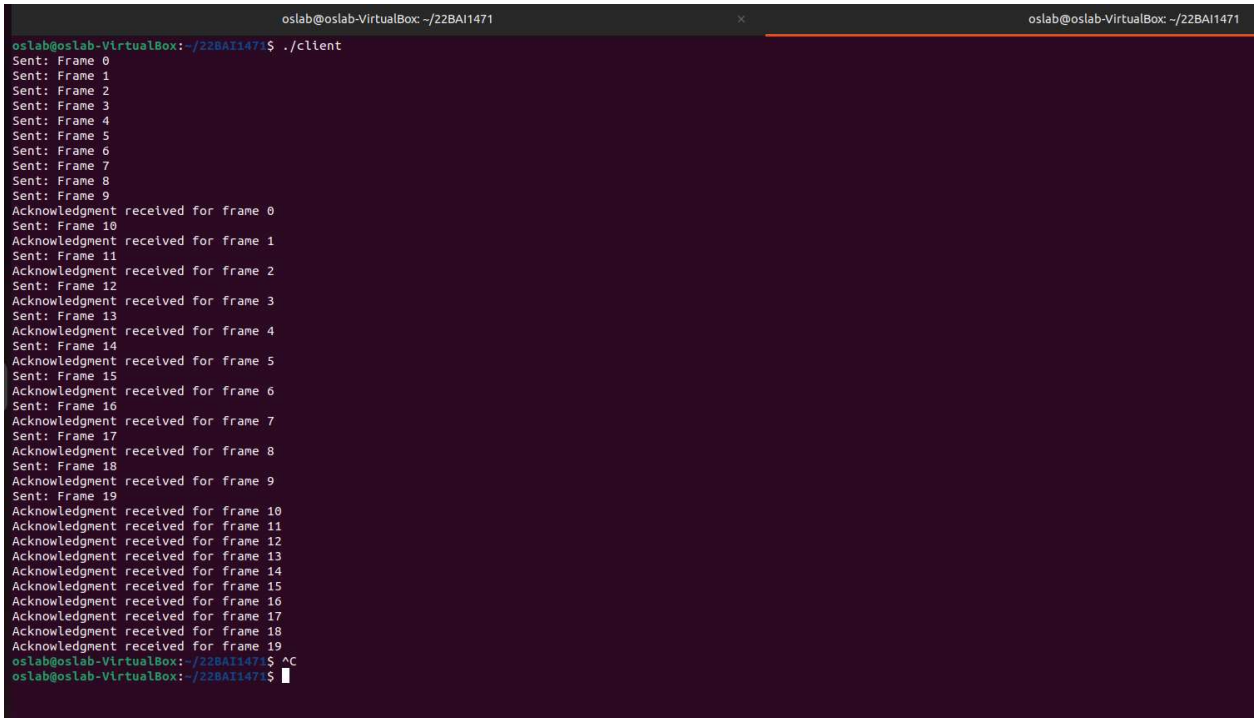
```
            int ack = expectedseqnum;
            send(connfd, &ack, sizeof(ack), 0);
            printf("Acknowledgment sent for frame %d\n", ack);
            expectedseqnum++;
            last_frame_received++;
        } else {
            printf("Receiver buffer full. Waiting for space...\n");
        }

        if (last_frame_received >= 19) {
            break;  // All frames received
        }
    }

    close(sockfd);
    return 0;
}
```

Output on the Command terminal

Server side terminal



client side terminal



Conclusion

The client or sender sends the first 10 frames numbering from 0 to 9 and for each frame it sent, it receives the acknowledgement.

Since the receiver capacity is set to 5 , after sending the first five frames(0-4) it starts receiving acknowledgements.Then after transmitting all 10 frames, it starts sending the next 10 frames.

Finally the client stops the execution of the program once all frames have been transmitted.