

**Bansilal Ramnath Agarwal Charitable Trust’s**

Vishwakarma Institute of Technology, Pune-37

*(Anautonomous Institute of Savitribai Phule Pune University)*

**Department of Computer Engineering**

|  |  |
| --- | --- |
| **Division** | **A** |
| **Batch** | **B2** |
| **GR-no** | **12110353** |
| **Roll no** | **10** |
| **Name** | **Aneesh Hemant Dighe** |

***TCP Berkeley Socket Primitives Peer-to-Peer mode***

Berkeley socket primitives are a set of system calls and functions in the Berkeley sockets API used for network communication in Unix-like operating systems. These primitives provide an interface for creating, configuring, and managing TCP (Transmission Control Protocol) connections. Key functions include socket() for creating sockets, bind() for specifying a local address, listen() for setting a socket to listen for incoming connections, accept() for accepting incoming connections, connect() to establish a connection to a remote host, and send() and recv() for data transmission. These primitives are fundamental for building networked applications, providing a low-level, reliable communication mechanism over TCP/IP networks.

***CODE:***

**selectiverepeat\_receiver.c**

#include <stdio.h>

#include <winsock2.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define MAX\_FRAME\_SIZE 1024

#define MAX\_WINDOW\_SIZE 10

#pragma comment(lib, "ws2\_32.lib")

void die(char \*message) {

    perror(message);

    exit(1);

}

int main() {

    WSADATA wsaData;

    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {

        die("WSAStartup failed");

    }

    SOCKET receiverSocket;

    receiverSocket = socket(AF\_INET, SOCK\_DGRAM, 0);

    if (receiverSocket == INVALID\_SOCKET) {

        die("Socket creation failed");

    }

    struct sockaddr\_in receiverAddr;

    memset(&receiverAddr, 0, sizeof(receiverAddr));

    receiverAddr.sin\_family = AF\_INET;

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

    receiverAddr.sin\_port = htons(12345);

    if (bind(receiverSocket, (struct sockaddr \*)&receiverAddr, sizeof(receiverAddr)) == SOCKET\_ERROR) {

        die("Binding failed");

    }

    int expectedSeqNum = 0;

    char ack[MAX\_FRAME\_SIZE];

    struct sockaddr\_in senderAddr;

    int senderAddrSize = sizeof(senderAddr);

    int done = 1;

    while (1) {

        char frame[MAX\_FRAME\_SIZE];

        int recvSize = recvfrom(receiverSocket, frame, sizeof(frame), 0, (struct sockaddr \*)&senderAddr, &senderAddrSize);

        if (recvSize == SOCKET\_ERROR) {

            die("Receive error");

        }

        frame[recvSize] = '\0';

        int frameNum = atoi(frame + 6); // Assuming "Frame " prefix in the frame data

        // if (frameNum == expectedSeqNum) {

            printf("Received Frame %d\n", frameNum);

            if (frameNum != 6 || done == 0) {

                // Simulate ACK transmission for all frames except frame number 6

                sprintf(ack, "%d", frameNum);

                sendto(receiverSocket, ack, strlen(ack), 0, (struct sockaddr \*)&senderAddr, sizeof(senderAddr));

                expectedSeqNum++;

            } else {

                done = 0;

            }

        // }

    }

    closesocket(receiverSocket);

    WSACleanup();

    return 0;

}

**selectiverepeat\_sender.c**

#include <stdio.h>

#include <winsock2.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define MAX\_FRAME\_SIZE 1024

#define MAX\_WINDOW\_SIZE 10

#define TIMEOUT 5000 // 5 seconds

#pragma comment(lib, "ws2\_32.lib")

void die(char \*message) {

    perror(message);

    exit(1);

}

int main() {

    WSADATA wsaData;

    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {

        die("WSAStartup failed");

    }

    SOCKET senderSocket;

    senderSocket = socket(AF\_INET, SOCK\_DGRAM, 0);

    if (senderSocket == INVALID\_SOCKET) {

        die("Socket creation failed");

    }

    struct sockaddr\_in receiverAddr;

    memset(&receiverAddr, 0, sizeof(receiverAddr));

    receiverAddr.sin\_family = AF\_INET;

    receiverAddr.sin\_addr.s\_addr = htonl(INADDR\_LOOPBACK);

    receiverAddr.sin\_port = htons(12345);

    int n, k;

    printf("Enter the number of frames to send (n): ");

    scanf("%d", &n);

    printf("Enter the window size (k): ");

    scanf("%d", &k);

    char frame[MAX\_FRAME\_SIZE];

    srand(time(NULL));

    int base = 0;

    int nextSeqNum = 0;

    while (base < n) {

        for (int i = base; i < base + k && i < n; i++) {

            // Simulate sending the frame

            sprintf(frame, "Frame %d", i);

            sendto(senderSocket, frame, strlen(frame), 0, (struct sockaddr \*)&receiverAddr, sizeof(receiverAddr));

            printf("Sent Frame %d\n", i);

        }

        int acked = base;

        int maxAck = 0;

        while (acked < base + k && acked < n) {

            struct timeval timeout;

            timeout.tv\_sec = TIMEOUT / 1000;

            timeout.tv\_usec = (TIMEOUT % 1000) \* 1000;

            fd\_set read\_fds;

            FD\_ZERO(&read\_fds);

            FD\_SET(senderSocket, &read\_fds);

            int ready = select(senderSocket + 1, &read\_fds, NULL, NULL, &timeout);

            if (ready == SOCKET\_ERROR) {

                die("Select error");

            } else if (ready == 0) {

                // Timeout occurred, resend frames from base

                printf("Timeout occurred. Resending frame %d\n", base);

                sprintf(frame, "Frame %d", base);

                sendto(senderSocket, frame, strlen(frame), 0, (struct sockaddr \*)&receiverAddr, sizeof(receiverAddr));

                printf("Sent Frame %d\n", base);

                nextSeqNum = base;

            } else {

                char ack[MAX\_FRAME\_SIZE];

                recvfrom(senderSocket, ack, sizeof(ack), 0, NULL, NULL);

                int ackNum = atoi(ack);

                printf("Received acknowledgment for Frame %d\n", ackNum);

                if(ackNum >= maxAck) {

                    maxAck = ackNum;

                } else {

                    acked = maxAck;

                    nextSeqNum = maxAck + 1;

                    break;

                }

                if (ackNum == nextSeqNum) {

                    nextSeqNum++;

                    acked++;

                }

            }

        }

        base = nextSeqNum;

    }

    closesocket(senderSocket);

    WSACleanup();

    return 0;

}

***OUTPUT:***

***Sender side:***

PS D:\5th Sem\CN(Computer Networks)\Assignments> ./selectiverepeat\_sender.exe

Enter the number of frames to send (n): 3

Enter the window size (k): 2

Sent Frame 0

Sent Frame 1

Received acknowledgment for Frame 0

Received acknowledgment for Frame 1

Sent Frame 2

Received acknowledgment for Frame 2

***Receiver side:***

PS D:\5th Sem\CN(Computer Networks)\Assignments> ./selectiverepeat\_receiver.exe

Received Frame 0

Received Frame 1

Received Frame 2

***Contents of File:***

Hello world,this is a file to test the connection of the TCP