Sliding Window Protocol

19Z510 – COMPUTER NETWORKS LABORATORY

Anandkumar NS (22Z209)

BACHELOR OF ENGINEERING

Date: 31/08/2024

DEPARTMENT OF COMPUTER SCIENCE

ENGINEERING PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

Client:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080
#define PACKET_SIZE 64

int main() {
    int sock = 0;
    struct sockaddr_in serv_addr;
    char buffer[PACKET_SIZE * 4] = {0};  // Buffer size modified to accommodate larger packets

    // Creating socket file descriptor
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\nSocket creation error\n");
        return -1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IPv4 and IPv6 addresses from text to binary form
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        printf("\nInvalid address/Address not supported\n");
        return -1;
    }

    // Connect to the server
    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        printf("\nConnection Failed\n");
        return -1;
    }

    printf("Connected to the server.\n");

    int expected_seq_num = 0;  // Initialize the expected sequence number

    while (1) {
        // Clear the buffer and read the incoming packets
        memset(buffer, 0, sizeof(buffer));
        int valread = read(sock, buffer, sizeof(buffer));

        if (valread <= 0) {
```

```
            printf("No more data from server or connection closed.\n");
            break;
        }

        // Process each packet in the buffer
        char *packet = strtok(buffer, "\n");
        while (packet != NULL) {
            if (strcmp(packet, "DONE") == 0) {
                printf("End of transmission received. Exiting.\n");
                close(sock);
                return 0;
            }

            int received_packet_num = atoi(packet + 7);  // Extract packet number after "Packet "

            if (received_packet_num == expected_seq_num) {
                printf("Received: %s\n", packet);

                // Prompt user for acknowledgment
                int ack_num;
                do {
                    printf("Enter ACK for Packet %d: ", expected_seq_num);
                    scanf("%d", &ack_num);

                    // If the entered ACK is correct, send it and move to the next packet
                    if (ack_num == expected_seq_num) {
                        // Send acknowledgment for the received packet
                        snprintf(buffer, PACKET_SIZE, "%d\n", ack_num);
                        send(sock, buffer, strlen(buffer), 0);
                        printf("Sent ACK for Packet %d\n", ack_num);
                        expected_seq_num++;
                    } else {
                        printf("Incorrect ACK. Please enter the correct ACK for Packet %d.\n",
expected_seq_num);
                    }
                } while (ack_num != expected_seq_num);
            } else {
                printf("Unexpected packet received. Expected %d but got %d\n",
expected_seq_num, received_packet_num);
            }

            packet = strtok(NULL, "\n");  // Move to the next packet
        }
    }

    close(sock);
    return 0;
```

```
}
```

Server:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/select.h>
#include <time.h>

#define PORT 8080
#define TOTAL_PACKETS 15
#define PACKET_SIZE 64
#define TIMEOUT 5  // 5 seconds timeout for ACK

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);
    char buffer[PACKET_SIZE] = {0};
    fd_set readfds;
    struct timeval timeout;

    // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket failed");
        exit(EXIT_FAILURE);
    }

    // Forcefully attaching socket to the port
    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt))) {
        perror("setsockopt");
        close(server_fd);
        exit(EXIT_FAILURE);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    // Binding the socket to the network address and port
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("Bind failed");
```

```
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    // Start listening for incoming connections
    if (listen(server_fd, 3) < 0) {
        perror("Listen failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    printf("Server is waiting for a connection...\n");

    // Accept the incoming connection
    if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen)) <
0) {
        perror("Accept failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    printf("Connection established with client.\n");

    int seq_num = 0;  // Sequence number to send

    while (seq_num < TOTAL_PACKETS) {
        int ack_received = 0;

        while (!ack_received) {
            // Send the packet
            snprintf(buffer, PACKET_SIZE, "Packet %d\n", seq_num);
            send(new_socket, buffer, strlen(buffer), 0);
            printf("Sent: %s", buffer);

            // Initialize the file descriptor set
            FD_ZERO(&readfds);
            FD_SET(new_socket, &readfds);

            // Set the timeout value
            timeout.tv_sec = TIMEOUT;
            timeout.tv_usec = 0;

            // Wait for an acknowledgment with a timeout
            int activity = select(new_socket + 1, &readfds, NULL, NULL, &timeout);

            if (activity > 0) {
                // Receive acknowledgment from the client
```

```
            memset(buffer, 0, PACKET_SIZE);
            int valread = read(new_socket, buffer, PACKET_SIZE);
            if (valread <= 0) {
               printf("No ACK received or connection closed. Exiting.\n");
               close(new_socket);
               close(server_fd);
               return 0;
            }

            buffer[valread] = '\0';
            int ack_num = atoi(buffer);  // Convert ACK to integer
            printf("Received ACK for Packet %d\n", ack_num);

            // Check if the acknowledgment is for the current packet
            if (ack_num == seq_num) {
               ack_received = 1;  // ACK received, move to the next packet
            } else {
               printf("Unexpected ACK received. Expected %d but got %d\n", seq_num,
ack_num);
            }
         } else if (activity == 0) {
            // Timeout occurred, resend the packet
            printf("Timeout occurred. Resending Packet %d\n", seq_num);
         }
      }

      seq_num++;  // Move to the next packet
   }

   // Send an end-of-transmission message
   snprintf(buffer, PACKET_SIZE, "DONE\n");
   send(new_socket, buffer, strlen(buffer), 0);
   printf("Sent end-of-transmission message.\n");

   close(new_socket);
   close(server_fd);
   return 0;
}
```