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;

}