JUNE 2018

S1. Write and run a TCP client and a TCP server program using 'C' language in Unix/Linux as per the following details : 20

• Client program will send three numbers to the server.

• Server will return the smallest number to the client:

Answer:

// server.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int addrlen = sizeof(address);

int numbers[3], smallest;

// Create socket

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("Socket failed");

exit(EXIT\_FAILURE);

}

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons(PORT);

// Bind the socket

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

perror("Bind failed");

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

if (listen(server\_fd, 3) < 0) {

perror("Listen failed");

exit(EXIT\_FAILURE);

}

printf("Server is listening on port %d...\n", PORT);

// Accept incoming connection from client

if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t\*)&addrlen)) < 0) {

perror("Accept failed");

exit(EXIT\_FAILURE);

}

// Read the three numbers from the client

read(new\_socket, numbers, sizeof(numbers));

// Find the smallest number

smallest = numbers[0];

for (int i = 1; i < 3; i++) {

if (numbers[i] < smallest) {

smallest = numbers[i];

}

}

// Send the smallest number back to the client

write(new\_socket, &smallest, sizeof(smallest));

printf("Smallest number is: %d\n", smallest);

// Close the socket

close(new\_socket);

close(server\_fd);

return 0;

}

Client

// client.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int sock = 0;

struct sockaddr\_in server\_address;

int numbers[3], smallest;

// Create socket

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

server\_address.sin\_family = AF\_INET;

server\_address.sin\_port = htons(PORT);

// Convert the IP address to binary format

if (inet\_pton(AF\_INET, "127.0.0.1", &server\_address.sin\_addr) <= 0) {

perror("Invalid address");

exit(EXIT\_FAILURE);

}

// Connect to the server

if (connect(sock, (struct sockaddr \*)&server\_address, sizeof(server\_address)) < 0) {

perror("Connection failed");

exit(EXIT\_FAILURE);

}

// Input the three numbers to be sent to the server

printf("Enter three numbers: ");

scanf("%d %d %d", &numbers[0], &numbers[1], &numbers[2]);

// Send the numbers to the server

send(sock, numbers, sizeof(numbers), 0);

// Receive the smallest number from the server

recv(sock, &smallest, sizeof(smallest), 0);

// Print the smallest number received

printf("Smallest number received from server: %d\n", smallest);

// Close the socket

close(sock);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

S2 Write and run a UDP client and a UDP server program using 'C' language in Unix/Linux as per the following details : 20

• UDP client will send a string to the server.

• Server will return the ASCII value of that string to the client.

Answer:

Server Side

// server.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define MAX\_BUFFER\_SIZE 1024

int main() {

int sockfd;

struct sockaddr\_in server\_addr, client\_addr;

char buffer[MAX\_BUFFER\_SIZE];

socklen\_t client\_addr\_len;

// Create UDP socket

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

server\_addr.sin\_port = htons(PORT);

// Bind the socket to the port

if (bind(sockfd, (const struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) {

perror("Bind failed");

exit(EXIT\_FAILURE);

}

printf("UDP Server listening on port %d...\n", PORT);

// Wait for the client to send a message

client\_addr\_len = sizeof(client\_addr);

int n = recvfrom(sockfd, (char \*)buffer, MAX\_BUFFER\_SIZE, MSG\_WAITALL,

(struct sockaddr \*)&client\_addr, &client\_addr\_len);

buffer[n] = '\0'; // Null-terminate the received string

printf("Received string from client: %s\n", buffer);

// Prepare the ASCII values of the string

char ascii\_values[MAX\_BUFFER\_SIZE];

for (int i = 0; i < n; i++) {

ascii\_values[i] = (char)buffer[i]; // ASCII values of each character

}

// Send the ASCII values back to the client

sendto(sockfd, (const char \*)ascii\_values, n, MSG\_CONFIRM,

(const struct sockaddr \*)&client\_addr, client\_addr\_len);

printf("Sent ASCII values to client.\n");

// Close the socket

close(sockfd);

return 0;

}

Client Side

// client.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define MAX\_BUFFER\_SIZE 1024

int main() {

int sockfd;

struct sockaddr\_in server\_addr;

char buffer[MAX\_BUFFER\_SIZE];

char ascii\_values[MAX\_BUFFER\_SIZE];

socklen\_t server\_addr\_len;

// Create UDP socket

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(PORT);

// Convert the IP address to binary format

if (inet\_pton(AF\_INET, "127.0.0.1", &server\_addr.sin\_addr) <= 0) {

perror("Invalid address");

exit(EXIT\_FAILURE);

}

// Input the string to be sent to the server

printf("Enter a string: ");

fgets(buffer, MAX\_BUFFER\_SIZE, stdin);

buffer[strcspn(buffer, "\n")] = '\0'; // Remove newline character

// Send the string to the server

sendto(sockfd, buffer, strlen(buffer), MSG\_CONFIRM,

(const struct sockaddr \*)&server\_addr, sizeof(server\_addr));

// Receive the ASCII values from the server

server\_addr\_len = sizeof(server\_addr);

int n = recvfrom(sockfd, ascii\_values, MAX\_BUFFER\_SIZE, MSG\_WAITALL,

(struct sockaddr \*)&server\_addr, &server\_addr\_len);

// Print the received ASCII values

printf("ASCII values received from server: ");

for (int i = 0; i < n; i++) {

printf("%d ", ascii\_values[i]);

}

printf("\n");

// Close the socket

close(sockfd);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

S3 Write and run a UDP client and a UDP server program using 'C' language in Unix/Linux as per the following details : 20

• Client program will send employee ID to the server.

• Server will fetch the corresponding name against this employee ID from the stored file and send back to the client.

Answer:

// server.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 12345

#define BUF\_SIZE 1024

#define FILE\_NAME "employee.txt"

// Function to read employee details from file and search by ID

char\* get\_employee\_name(int emp\_id) {

static char emp\_name[100];

FILE \*file = fopen(FILE\_NAME, "r");

if (file == NULL) {

perror("File opening failed");

return NULL;

}

int id;

char name[100];

while (fscanf(file, "%d %s", &id, name) != EOF) {

if (id == emp\_id) {

fclose(file);

strcpy(emp\_name, name);

return emp\_name;

}

}

fclose(file);

return NULL; // ID not found

}

int main() {

int sockfd;

struct sockaddr\_in server\_addr, client\_addr;

socklen\_t client\_addr\_len = sizeof(client\_addr);

char buffer[BUF\_SIZE];

// Create UDP socket

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

memset(&server\_addr, 0, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

server\_addr.sin\_port = htons(PORT);

// Bind the socket to the port

if (bind(sockfd, (const struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) {

perror("Bind failed");

close(sockfd);

exit(EXIT\_FAILURE);

}

printf("Server listening on port %d...\n", PORT);

while (1) {

int emp\_id;

ssize\_t len = recvfrom(sockfd, &emp\_id, sizeof(emp\_id), MSG\_WAITALL, (struct sockaddr \*)&client\_addr, &client\_addr\_len);

if (len < 0) {

perror("Receive failed");

continue;

}

printf("Received employee ID: %d\n", emp\_id);

// Look up employee name by ID

char \*emp\_name = get\_employee\_name(emp\_id);

if (emp\_name) {

sendto(sockfd, emp\_name, strlen(emp\_name) + 1, MSG\_CONFIRM, (const struct sockaddr \*)&client\_addr, client\_addr\_len);

} else {

char \*not\_found = "Employee not found";

sendto(sockfd, not\_found, strlen(not\_found) + 1, MSG\_CONFIRM, (const struct sockaddr \*)&client\_addr, client\_addr\_len);

}

}

close(sockfd);

return 0;

}

Client Side

// client.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 12345

#define BUF\_SIZE 1024

int main() {

int sockfd;

struct sockaddr\_in server\_addr;

char buffer[BUF\_SIZE];

int emp\_id;

// Create UDP socket

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

memset(&server\_addr, 0, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(PORT);

server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); // Local server address

// Input employee ID

printf("Enter Employee ID: ");

scanf("%d", &emp\_id);

// Send employee ID to server

sendto(sockfd, &emp\_id, sizeof(emp\_id), MSG\_CONFIRM, (const struct sockaddr \*)&server\_addr, sizeof(server\_addr));

// Receive the employee name from server

ssize\_t len = recvfrom(sockfd, buffer, BUF\_SIZE, MSG\_WAITALL, NULL, NULL);

if (len < 0) {

perror("Receive failed");

close(sockfd);

exit(EXIT\_FAILURE);

}

// Output the server response

printf("Employee Name: %s\n", buffer);

close(sockfd);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

S4 Write and run a TCP client and a TCP server program in 'C' language in Unix/Linux as per the following details : 20

• TCP 'client will establish a connection with the server and send two numbers to the server.

• TCP server will calculate the multiplication of these numbers and return to client.

Answer:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int server\_fd, client\_fd;

struct sockaddr\_in server\_addr, client\_addr;

socklen\_t addr\_len = sizeof(client\_addr);

int num1, num2, result;

// Create socket

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {

perror("Socket creation failed");

exit(1);

}

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

server\_addr.sin\_port = htons(PORT);

// Bind the socket

if (bind(server\_fd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

perror("Bind failed");

close(server\_fd);

exit(1);

}

// Listen for incoming connections

if (listen(server\_fd, 5) == -1) {

perror("Listen failed");

close(server\_fd);

exit(1);

}

printf("Server listening on port %d...\n", PORT);

// Accept client connection

if ((client\_fd = accept(server\_fd, (struct sockaddr \*)&client\_addr, &addr\_len)) == -1) {

perror("Accept failed");

close(server\_fd);

exit(1);

}

// Receive two numbers from client

if (recv(client\_fd, &num1, sizeof(int), 0) <= 0 || recv(client\_fd, &num2, sizeof(int), 0) <= 0) {

perror("Receive failed");

close(client\_fd);

close(server\_fd);

exit(1);

}

printf("Received numbers: %d and %d\n", num1, num2);

// Calculate multiplication

result = num1 \* num2;

// Send the result back to the client

if (send(client\_fd, &result, sizeof(int), 0) <= 0) {

perror("Send failed");

close(client\_fd);

close(server\_fd);

exit(1);

}

printf("Sent result: %d\n", result);

// Close connections

close(client\_fd);

close(server\_fd);

return 0;

}

Client Side

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <arpa/inet.h>

#define PORT 8080

#define SERVER\_IP "127.0.0.1" // Localhost

int main() {

int sock\_fd;

struct sockaddr\_in server\_addr;

int num1, num2, result;

// Create socket

if ((sock\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {

perror("Socket creation failed");

exit(1);

}

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(PORT);

// Convert IP address to binary form

if (inet\_pton(AF\_INET, SERVER\_IP, &server\_addr.sin\_addr) <= 0) {

perror("Invalid address or address not supported");

close(sock\_fd);

exit(1);

}

// Connect to the server

if (connect(sock\_fd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

perror("Connection failed");

close(sock\_fd);

exit(1);

}

// Get two numbers from user

printf("Enter first number: ");

scanf("%d", &num1);

printf("Enter second number: ");

scanf("%d", &num2);

// Send numbers to server

if (send(sock\_fd, &num1, sizeof(int), 0) <= 0 || send(sock\_fd, &num2, sizeof(int), 0) <= 0) {

perror("Send failed");

close(sock\_fd);

exit(1);

}

// Receive result from server

if (recv(sock\_fd, &result, sizeof(int), 0) <= 0) {

perror("Receive failed");

close(sock\_fd);

exit(1);

}

printf("Received result: %d\n", result);

// Close socket

close(sock\_fd);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

DECEMBER 2017

S2 Write and run a UDP client and a UDP server program using 'C' language in Unix/Linux operating system as per the following details : 20

• UDP client will send a text string to the client.

• UDP server will return its reverse order to the client.

Answer:

// udp\_server.c

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUF\_SIZE 1024

void reverse\_string(char\* str) {

int len = strlen(str);

int start = 0, end = len - 1;

while (start < end) {

// Swap characters at start and end

char temp = str[start];

str[start] = str[end];

str[end] = temp;

start++;

end--;

}

}

int main() {

int sockfd;

struct sockaddr\_in server\_addr, client\_addr;

char buffer[BUF\_SIZE];

socklen\_t client\_len = sizeof(client\_addr);

// Create UDP socket

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0) {

perror("Socket creation failed");

exit(1);

}

memset(&server\_addr, 0, sizeof(server\_addr));

// Initialize server address

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

server\_addr.sin\_port = htons(PORT);

// Bind the socket with the server address

if (bind(sockfd, (const struct sockaddr\*)&server\_addr, sizeof(server\_addr)) < 0) {

perror("Bind failed");

exit(1);

}

printf("UDP server is running on port %d...\n", PORT);

while (1) {

// Receive message from client

int n = recvfrom(sockfd, buffer, BUF\_SIZE, 0, (struct sockaddr\*)&client\_addr, &client\_len);

buffer[n] = '\0'; // Null-terminate the received string

printf("Client: %s\n", buffer);

// Reverse the string

reverse\_string(buffer);

// Send the reversed string back to the client

sendto(sockfd, buffer, strlen(buffer), 0, (const struct sockaddr\*)&client\_addr, client\_len);

}

close(sockfd);

return 0;

}

Client

// udp\_client.c

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUF\_SIZE 1024

int main() {

int sockfd;

struct sockaddr\_in server\_addr;

char buffer[BUF\_SIZE];

socklen\_t len;

// Create UDP socket

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0) {

perror("Socket creation failed");

exit(1);

}

memset(&server\_addr, 0, sizeof(server\_addr));

// Initialize server address

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(PORT);

server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); // Localhost (change if server is remote)

printf("Enter a string to send to the server: ");

fgets(buffer, BUF\_SIZE, stdin);

buffer[strcspn(buffer, "\n")] = '\0'; // Remove the newline character

// Send the string to the server

sendto(sockfd, buffer, strlen(buffer), 0, (const struct sockaddr\*)&server\_addr, sizeof(server\_addr));

// Receive the reversed string from the server

int n = recvfrom(sockfd, buffer, BUF\_SIZE, 0, (struct sockaddr\*)&server\_addr, &len);

buffer[n] = '\0'; // Null-terminate the received string

printf("Reversed string from server: %s\n", buffer);

close(sockfd);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

S3 Write and run a UDP client and a UDP server program using 'C' language in Unix/Linux as per the following details : 20

• UDP client will send an enrollment number of a BCA student.

• UDP server will fetch the corresponding name against this enrollment number from the stored records and send back to the client.

Answer:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define SERVER\_PORT 8080

#define MAX\_BUFFER\_SIZE 1024

int main() {

int sockfd;

struct sockaddr\_in server\_addr;

char buffer[MAX\_BUFFER\_SIZE];

socklen\_t addr\_len;

char \*enrollment\_number;

// Create a UDP socket

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Set server details

memset(&server\_addr, 0, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(SERVER\_PORT);

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

// Get enrollment number from user input

printf("Enter the enrollment number: ");

fgets(buffer, MAX\_BUFFER\_SIZE, stdin);

enrollment\_number = strtok(buffer, "\n"); // Remove the newline character

// Send enrollment number to the server

sendto(sockfd, enrollment\_number, strlen(enrollment\_number), 0, (const struct sockaddr \*)&server\_addr, sizeof(server\_addr));

printf("Enrollment number sent to the server: %s\n", enrollment\_number);

// Receive the corresponding student name from the server

addr\_len = sizeof(server\_addr);

int n = recvfrom(sockfd, buffer, MAX\_BUFFER\_SIZE, 0, (struct sockaddr \*)&server\_addr, &addr\_len);

buffer[n] = '\0'; // Null-terminate the received string

// Display the received student name

printf("Student name received from server: %s\n", buffer);

// Close the socket

close(sockfd);

return 0;

}

Client Side

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define SERVER\_PORT 8080

#define MAX\_BUFFER\_SIZE 1024

// Sample student records

struct student {

char enrollment\_number[20];

char name[50];

};

struct student records[] = {

{"BCA101", "John Doe"},

{"BCA102", "Jane Smith"},

{"BCA103", "Alice Johnson"},

{"BCA104", "Bob Brown"}

};

#define NUM\_RECORDS (sizeof(records) / sizeof(records[0]))

// Function to find the student name by enrollment number

char\* get\_student\_name\_by\_enrollment(char\* enrollment\_number) {

for (int i = 0; i < NUM\_RECORDS; i++) {

if (strcmp(records[i].enrollment\_number, enrollment\_number) == 0) {

return records[i].name;

}

}

return "Record not found"; // Return error message if record is not found

}

int main() {

int sockfd;

struct sockaddr\_in server\_addr, client\_addr;

char buffer[MAX\_BUFFER\_SIZE];

socklen\_t addr\_len;

char \*enrollment\_number, \*student\_name;

// Create a UDP socket

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Set server details

memset(&server\_addr, 0, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(SERVER\_PORT);

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

// Bind the socket to the server address

if (bind(sockfd, (const struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) {

perror("Bind failed");

close(sockfd);

exit(EXIT\_FAILURE);

}

printf("UDP server is waiting for requests...\n");

// Infinite loop to receive client messages and send responses

while (1) {

addr\_len = sizeof(client\_addr);

// Receive the enrollment number from client

int n = recvfrom(sockfd, buffer, MAX\_BUFFER\_SIZE, 0, (struct sockaddr \*)&client\_addr, &addr\_len);

buffer[n] = '\0'; // Null-terminate the received string

enrollment\_number = buffer;

printf("Received enrollment number: %s\n", enrollment\_number);

// Get the student name from the enrollment number

student\_name = get\_student\_name\_by\_enrollment(enrollment\_number);

// Send the student name back to the client

sendto(sockfd, student\_name, strlen(student\_name), 0, (const struct sockaddr \*)&client\_addr, addr\_len);

printf("Sent student name: %s\n", student\_name);

}

// Close the socket (not reached if server runs forever)

close(sockfd);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

S 4 Write and run a TCP client and a TCP server program in 'C' language in Unix/Linux as per the following details : 20

• TCP client will establish a connection with the server and send two text strings to the server.

• TCP server will concatenate these strings and send back to the client.

Answer:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define MAX\_BUFFER 1024

int main() {

int server\_fd, client\_fd;

struct sockaddr\_in server\_addr, client\_addr;

socklen\_t addr\_len;

char buffer[MAX\_BUFFER];

// Create socket

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Set server address

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

server\_addr.sin\_port = htons(PORT);

// Bind the socket

if (bind(server\_fd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

perror("Bind failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

if (listen(server\_fd, 5) == -1) {

perror("Listen failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

printf("Server listening on port %d...\n", PORT);

// Accept client connection

addr\_len = sizeof(client\_addr);

if ((client\_fd = accept(server\_fd, (struct sockaddr \*)&client\_addr, &addr\_len)) == -1) {

perror("Accept failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

// Read two text strings from the client

recv(client\_fd, buffer, MAX\_BUFFER, 0);

char str1[MAX\_BUFFER];

strncpy(str1, buffer, MAX\_BUFFER);

recv(client\_fd, buffer, MAX\_BUFFER, 0);

char str2[MAX\_BUFFER];

strncpy(str2, buffer, MAX\_BUFFER);

// Concatenate strings

strcat(str1, str2);

// Send concatenated string back to client

send(client\_fd, str1, strlen(str1), 0);

printf("Concatenated string sent to client: %s\n", str1);

// Close connections

close(client\_fd);

close(server\_fd);

return 0;

}

Client Side

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define MAX\_BUFFER 1024

int main() {

int client\_fd;

struct sockaddr\_in server\_addr;

char buffer[MAX\_BUFFER];

char str1[MAX\_BUFFER], str2[MAX\_BUFFER];

// Create socket

if ((client\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Set server address

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(PORT);

// Convert IP address from text to binary

if (inet\_pton(AF\_INET, "127.0.0.1", &server\_addr.sin\_addr) <= 0) {

perror("Invalid address");

close(client\_fd);

exit(EXIT\_FAILURE);

}

// Connect to server

if (connect(client\_fd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

perror("Connection failed");

close(client\_fd);

exit(EXIT\_FAILURE);

}

// Read two strings from user input

printf("Enter first string: ");

fgets(str1, MAX\_BUFFER, stdin);

str1[strcspn(str1, "\n")] = 0; // Remove trailing newline

printf("Enter second string: ");

fgets(str2, MAX\_BUFFER, stdin);

str2[strcspn(str2, "\n")] = 0; // Remove trailing newline

// Send strings to the server

send(client\_fd, str1, strlen(str1), 0);

send(client\_fd, str2, strlen(str2), 0);

// Receive concatenated string from server

recv(client\_fd, buffer, MAX\_BUFFER, 0);

printf("Concatenated string from server: %s\n", buffer);

// Close the connection

close(client\_fd);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

June 2017

S1 Write and run a TCP client and a TCP server program using 'C' language in Unix/Linux according to the following specifications : 20

• Client program will send a string of characters to the specific server.

• Server program will count the number of characters in the given string and return back to the respective client.

Answer:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int server\_fd, client\_fd;

struct sockaddr\_in server\_addr, client\_addr;

socklen\_t addr\_len;

char buffer[BUFFER\_SIZE];

ssize\_t bytes\_received, bytes\_sent;

// Create a socket

server\_fd = socket(AF\_INET, SOCK\_STREAM, 0);

if (server\_fd == -1) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Set up the server address structure

memset(&server\_addr, 0, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

server\_addr.sin\_port = htons(PORT);

// Bind the socket to the port

if (bind(server\_fd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

perror("Binding failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

// Start listening for incoming connections

if (listen(server\_fd, 5) == -1) {

perror("Listen failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

printf("Server listening on port %d...\n", PORT);

// Accept an incoming client connection

addr\_len = sizeof(client\_addr);

client\_fd = accept(server\_fd, (struct sockaddr \*)&client\_addr, &addr\_len);

if (client\_fd == -1) {

perror("Accept failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

printf("Client connected!\n");

// Receive the string from the client

bytes\_received = recv(client\_fd, buffer, sizeof(buffer), 0);

if (bytes\_received == -1) {

perror("Receive failed");

close(client\_fd);

close(server\_fd);

exit(EXIT\_FAILURE);

}

buffer[bytes\_received] = '\0'; // Null terminate the received string

printf("Received string: %s\n", buffer);

// Count the number of characters in the received string

int char\_count = strlen(buffer);

// Send the character count back to the client

bytes\_sent = send(client\_fd, &char\_count, sizeof(char\_count), 0);

if (bytes\_sent == -1) {

perror("Send failed");

close(client\_fd);

close(server\_fd);

exit(EXIT\_FAILURE);

}

printf("Character count sent to client: %d\n", char\_count);

// Close the connections

close(client\_fd);

close(server\_fd);

return 0;

}

Client Side

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int client\_fd;

struct sockaddr\_in server\_addr;

char buffer[BUFFER\_SIZE];

int char\_count;

// Create a socket

client\_fd = socket(AF\_INET, SOCK\_STREAM, 0);

if (client\_fd == -1) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Set up the server address structure

memset(&server\_addr, 0, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(PORT);

// Convert IP address from text to binary form

if (inet\_pton(AF\_INET, "127.0.0.1", &server\_addr.sin\_addr) <= 0) {

perror("Invalid address");

close(client\_fd);

exit(EXIT\_FAILURE);

}

// Connect to the server

if (connect(client\_fd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

perror("Connection failed");

close(client\_fd);

exit(EXIT\_FAILURE);

}

printf("Enter a string to send to the server: ");

fgets(buffer, sizeof(buffer), stdin);

buffer[strcspn(buffer, "\n")] = '\0'; // Remove newline character

// Send the string to the server

ssize\_t bytes\_sent = send(client\_fd, buffer, strlen(buffer), 0);

if (bytes\_sent == -1) {

perror("Send failed");

close(client\_fd);

exit(EXIT\_FAILURE);

}

printf("String sent to server: %s\n", buffer);

// Receive the character count from the server

ssize\_t bytes\_received = recv(client\_fd, &char\_count, sizeof(char\_count), 0);

if (bytes\_received == -1) {

perror("Receive failed");

close(client\_fd);

exit(EXIT\_FAILURE);

}

printf("Character count received from server: %d\n", char\_count);

// Close the connection

close(client\_fd);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

S2 Write a UDP client and a UDP server program in 'C' language in Unix/Linux as per the following specifications : 20

• A UDP client program and a UDP server should be available at different machines in a LAN. UDP client should send a series of numbers to the server.

• UDP server will find the largest number and return back to the respective client.

Answer:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int server\_fd;

struct sockaddr\_in server\_addr, client\_addr;

socklen\_t addr\_len;

char buffer[BUFFER\_SIZE];

int numbers[BUFFER\_SIZE / sizeof(int)];

int num\_count, largest\_num;

// Create a UDP socket

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

if (server\_fd == -1) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Set up the server address structure

memset(&server\_addr, 0, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

server\_addr.sin\_port = htons(PORT);

// Bind the socket to the port

if (bind(server\_fd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

perror("Binding failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

printf("Server listening on port %d...\n", PORT);

while (1) {

addr\_len = sizeof(client\_addr);

// Receive the series of numbers from the client

int bytes\_received = recvfrom(server\_fd, buffer, sizeof(buffer), 0, (struct sockaddr \*)&client\_addr, &addr\_len);

if (bytes\_received == -1) {

perror("Receive failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

// Determine the number of integers sent by the client

num\_count = bytes\_received / sizeof(int);

memcpy(numbers, buffer, bytes\_received);

// Find the largest number in the array

largest\_num = numbers[0];

for (int i = 1; i < num\_count; i++) {

if (numbers[i] > largest\_num) {

largest\_num = numbers[i];

}

}

printf("Received numbers: ");

for (int i = 0; i < num\_count; i++) {

printf("%d ", numbers[i]);

}

printf("\n");

printf("Largest number: %d\n", largest\_num);

// Send the largest number back to the client

int bytes\_sent = sendto(server\_fd, &largest\_num, sizeof(largest\_num), 0, (struct sockaddr \*)&client\_addr, addr\_len);

if (bytes\_sent == -1) {

perror("Send failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

}

// Close the socket (never reached in this infinite loop)

close(server\_fd);

return 0;

}

Client Side

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int client\_fd;

struct sockaddr\_in server\_addr;

char buffer[BUFFER\_SIZE];

int numbers[] = {5, 9, 3, 7, 2, 8}; // Example numbers to send

int num\_count = sizeof(numbers) / sizeof(numbers[0]);

int largest\_num;

// Create a UDP socket

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

if (client\_fd == -1) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Set up the server address structure

memset(&server\_addr, 0, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(PORT);

// Convert IP address from text to binary form

if (inet\_pton(AF\_INET, "127.0.0.1", &server\_addr.sin\_addr) <= 0) { // Replace "127.0.0.1" with the server's IP address

perror("Invalid address");

close(client\_fd);

exit(EXIT\_FAILURE);

}

// Send the series of numbers to the server

memcpy(buffer, numbers, num\_count \* sizeof(int));

ssize\_t bytes\_sent = sendto(client\_fd, buffer, num\_count \* sizeof(int), 0, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));

if (bytes\_sent == -1) {

perror("Send failed");

close(client\_fd);

exit(EXIT\_FAILURE);

}

printf("Sent numbers to server: ");

for (int i = 0; i < num\_count; i++) {

printf("%d ", numbers[i]);

}

printf("\n");

// Receive the largest number from the server

ssize\_t bytes\_received = recvfrom(client\_fd, &largest\_num, sizeof(largest\_num), 0, NULL, NULL);

if (bytes\_received == -1) {

perror("Receive failed");

close(client\_fd);

exit(EXIT\_FAILURE);

}

printf("Largest number received from server: %d\n", largest\_num);

// Close the socket

close(client\_fd);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

DECEMBER 2016

S1 Write and run a TCP client and a TCP server program using 'C' language in Unix/Linux as per the following details : 20

• A TCP client program should send 3 numbers to the server. Server can have maximum 3 clients at a time.

• The server should send back the largest among the 3 numbers to the respective client.

Answer:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <pthread.h>

#define PORT 8080

#define MAX\_CLIENTS 3

// Function to handle client connections

void \*client\_handler(void \*client\_sock) {

int sock = \*(int \*)client\_sock;

int numbers[3], largest\_num;

// Receive the 3 numbers from the client

ssize\_t bytes\_received = recv(sock, numbers, sizeof(numbers), 0);

if (bytes\_received == -1) {

perror("Receive failed");

close(sock);

pthread\_exit(NULL);

}

// Find the largest number

largest\_num = numbers[0];

for (int i = 1; i < 3; i++) {

if (numbers[i] > largest\_num) {

largest\_num = numbers[i];

}

}

// Send the largest number back to the client

ssize\_t bytes\_sent = send(sock, &largest\_num, sizeof(largest\_num), 0);

if (bytes\_sent == -1) {

perror("Send failed");

}

// Close the socket

close(sock);

pthread\_exit(NULL);

}

int main() {

int server\_fd, client\_fd;

struct sockaddr\_in server\_addr, client\_addr;

socklen\_t addr\_len;

pthread\_t threads[MAX\_CLIENTS];

int client\_count = 0;

// Create the server socket

server\_fd = socket(AF\_INET, SOCK\_STREAM, 0);

if (server\_fd == -1) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Set up the server address

memset(&server\_addr, 0, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

server\_addr.sin\_port = htons(PORT);

// Bind the socket to the port

if (bind(server\_fd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

perror("Bind failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

if (listen(server\_fd, MAX\_CLIENTS) == -1) {

perror("Listen failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

printf("Server is listening on port %d...\n", PORT);

// Accept incoming client connections

while (client\_count < MAX\_CLIENTS) {

addr\_len = sizeof(client\_addr);

client\_fd = accept(server\_fd, (struct sockaddr \*)&client\_addr, &addr\_len);

if (client\_fd == -1) {

perror("Accept failed");

continue;

}

printf("Client connected! Handling client...\n");

// Create a thread to handle the client

if (pthread\_create(&threads[client\_count], NULL, client\_handler, (void \*)&client\_fd) != 0) {

perror("Thread creation failed");

close(client\_fd);

continue;

}

client\_count++;

}

// Wait for all threads to finish

for (int i = 0; i < client\_count; i++) {

pthread\_join(threads[i], NULL);

}

// Close the server socket

close(server\_fd);

return 0;

}

Client Side

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int client\_fd;

struct sockaddr\_in server\_addr;

int numbers[3];

int largest\_num;

// Create the client socket

client\_fd = socket(AF\_INET, SOCK\_STREAM, 0);

if (client\_fd == -1) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Set up the server address structure

memset(&server\_addr, 0, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(PORT);

// Convert IP address from text to binary form

if (inet\_pton(AF\_INET, "127.0.0.1", &server\_addr.sin\_addr) <= 0) { // Use server's IP if different

perror("Invalid address");

close(client\_fd);

exit(EXIT\_FAILURE);

}

// Connect to the server

if (connect(client\_fd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

perror("Connection failed");

close(client\_fd);

exit(EXIT\_FAILURE);

}

// Get 3 numbers from the user

printf("Enter 3 numbers: ");

for (int i = 0; i < 3; i++) {

scanf("%d", &numbers[i]);

}

// Send the numbers to the server

ssize\_t bytes\_sent = send(client\_fd, numbers, sizeof(numbers), 0);

if (bytes\_sent == -1) {

perror("Send failed");

close(client\_fd);

exit(EXIT\_FAILURE);

}

// Receive the largest number from the server

ssize\_t bytes\_received = recv(client\_fd, &largest\_num, sizeof(largest\_num), 0);

if (bytes\_received == -1) {

perror("Receive failed");

close(client\_fd);

exit(EXIT\_FAILURE);

}

// Print the largest number received from the server

printf("The largest number is: %d\n", largest\_num);

// Close the socket

close(client\_fd);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

S2 Write and run a client and a server program using 'C' language in Unix/Linux with the following specifications : 20

• A UDP client will send a string to the server.

• The server will check the string, whether it is a palindrome or not. If yes, the server will send reply as "palindrome found" else send the string "no palindrome found" to the respective client.

Answer:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

// Function to check if a string is a palindrome

int is\_palindrome(const char \*str) {

int length = strlen(str);

for (int i = 0; i < length / 2; i++) {

if (str[i] != str[length - 1 - i]) {

return 0; // Not a palindrome

}

}

return 1; // Is a palindrome

}

int main() {

int server\_fd;

struct sockaddr\_in server\_addr, client\_addr;

socklen\_t addr\_len;

char buffer[BUFFER\_SIZE];

char response[BUFFER\_SIZE];

// Create a UDP socket

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

if (server\_fd == -1) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Set up the server address

memset(&server\_addr, 0, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

server\_addr.sin\_port = htons(PORT);

// Bind the socket to the port

if (bind(server\_fd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

perror("Binding failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

printf("Server is listening on port %d...\n", PORT);

while (1) {

addr\_len = sizeof(client\_addr);

// Receive the string from the client

int bytes\_received = recvfrom(server\_fd, buffer, sizeof(buffer), 0, (struct sockaddr \*)&client\_addr, &addr\_len);

if (bytes\_received == -1) {

perror("Receive failed");

continue;

}

buffer[bytes\_received] = '\0'; // Null terminate the received string

// Check if the string is a palindrome

if (is\_palindrome(buffer)) {

strcpy(response, "Palindrome found");

} else {

strcpy(response, "No palindrome found");

}

// Send the response back to the client

int bytes\_sent = sendto(server\_fd, response, strlen(response), 0, (struct sockaddr \*)&client\_addr, addr\_len);

if (bytes\_sent == -1) {

perror("Send failed");

}

}

// Close the socket (never reached in this infinite loop)

close(server\_fd);

return 0;

}

Client Side

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int client\_fd;

struct sockaddr\_in server\_addr;

char buffer[BUFFER\_SIZE];

char response[BUFFER\_SIZE];

// Create a UDP socket

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

if (client\_fd == -1) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Set up the server address structure

memset(&server\_addr, 0, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(PORT);

// Convert IP address from text to binary form

if (inet\_pton(AF\_INET, "127.0.0.1", &server\_addr.sin\_addr) <= 0) { // Use server's IP if different

perror("Invalid address");

close(client\_fd);

exit(EXIT\_FAILURE);

}

// Get a string input from the user

printf("Enter a string to check if it's a palindrome: ");

fgets(buffer, sizeof(buffer), stdin);

buffer[strcspn(buffer, "\n")] = '\0'; // Remove the newline character

// Send the string to the server

ssize\_t bytes\_sent = sendto(client\_fd, buffer, strlen(buffer), 0, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));

if (bytes\_sent == -1) {

perror("Send failed");

close(client\_fd);

exit(EXIT\_FAILURE);

}

// Receive the response from the server

ssize\_t bytes\_received = recvfrom(client\_fd, response, sizeof(response), 0, NULL, NULL);

if (bytes\_received == -1) {

perror("Receive failed");

close(client\_fd);

exit(EXIT\_FAILURE);

}

// Print the response from the server

printf("Server response: %s\n", response);

// Close the socket

close(client\_fd);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

JUNE 2016

S2 Write and run a TCP Client and a TCP Server program using 'C' language in Unix/Linux as per the following details : 20

• Client program will send a string including characters and numbers, to the server.

• Server will extract these numbers from this string and return back to the client.

Answer:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <ctype.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

// Function to extract numbers from a string

void extract\_numbers(char \*input, char \*output) {

while (\*input) {

if (isdigit(\*input)) {

\*output = \*input; // Copy the digit to the output

output++;

}

input++;

}

\*output = '\0'; // Null terminate the extracted numbers string

}

int main() {

int server\_fd, client\_fd;

struct sockaddr\_in server\_addr, client\_addr;

socklen\_t addr\_len;

char buffer[BUFFER\_SIZE];

char response[BUFFER\_SIZE];

// Create the server socket

server\_fd = socket(AF\_INET, SOCK\_STREAM, 0);

if (server\_fd == -1) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Set up the server address structure

memset(&server\_addr, 0, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

server\_addr.sin\_port = htons(PORT);

// Bind the socket to the port

if (bind(server\_fd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

perror("Binding failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

if (listen(server\_fd, 5) == -1) {

perror("Listen failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

printf("Server is listening on port %d...\n", PORT);

// Accept incoming client connection

addr\_len = sizeof(client\_addr);

client\_fd = accept(server\_fd, (struct sockaddr \*)&client\_addr, &addr\_len);

if (client\_fd == -1) {

perror("Accept failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

printf("Client connected!\n");

// Receive the string from the client

int bytes\_received = recv(client\_fd, buffer, sizeof(buffer), 0);

if (bytes\_received == -1) {

perror("Receive failed");

close(client\_fd);

close(server\_fd);

exit(EXIT\_FAILURE);

}

buffer[bytes\_received] = '\0'; // Null terminate the received string

printf("Received from client: %s\n", buffer);

// Extract numbers from the received string

extract\_numbers(buffer, response);

// Send the extracted numbers back to the client

int bytes\_sent = send(client\_fd, response, strlen(response), 0);

if (bytes\_sent == -1) {

perror("Send failed");

}

printf("Sent to client: %s\n", response);

// Close the connection

close(client\_fd);

close(server\_fd);

return 0;

}

Client Side

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int client\_fd;

struct sockaddr\_in server\_addr;

char buffer[BUFFER\_SIZE];

char response[BUFFER\_SIZE];

// Create the client socket

client\_fd = socket(AF\_INET, SOCK\_STREAM, 0);

if (client\_fd == -1) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Set up the server address structure

memset(&server\_addr, 0, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(PORT);

// Convert IP address from text to binary form

if (inet\_pton(AF\_INET, "127.0.0.1", &server\_addr.sin\_addr) <= 0) { // Use server's IP if different

perror("Invalid address");

close(client\_fd);

exit(EXIT\_FAILURE);

}

// Connect to the server

if (connect(client\_fd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

perror("Connection failed");

close(client\_fd);

exit(EXIT\_FAILURE);

}

// Get a string from the user

printf("Enter a string (with characters and numbers): ");

fgets(buffer, sizeof(buffer), stdin);

buffer[strcspn(buffer, "\n")] = '\0'; // Remove the newline character

// Send the string to the server

ssize\_t bytes\_sent = send(client\_fd, buffer, strlen(buffer), 0);

if (bytes\_sent == -1) {

perror("Send failed");

close(client\_fd);

exit(EXIT\_FAILURE);

}

// Receive the response from the server (extracted numbers)

ssize\_t bytes\_received = recv(client\_fd, response, sizeof(response), 0);

if (bytes\_received == -1) {

perror("Receive failed");

close(client\_fd);

exit(EXIT\_FAILURE);

}

// Null-terminate the response and print it

response[bytes\_received] = '\0';

printf("Numbers extracted from the string: %s\n", response);

// Close the socket

close(client\_fd);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

DECEMBER 2015

S4 Write and run a UDP Client and a UDP Server program using 'C' language in Unix/Linux operating system as per the following specifications : 20

• Client will sent "hello" message to the Server.

• UDP Server will return back with a "welcome" message to that Client.

Answer:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define SERVER\_PORT 12345

#define MAX\_BUF\_SIZE 1024

int main() {

int sockfd;

struct sockaddr\_in server\_addr, client\_addr;

char buffer[MAX\_BUF\_SIZE];

socklen\_t client\_len = sizeof(client\_addr);

// Create UDP socket

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0) {

perror("Socket creation failed");

exit(1);

}

// Server address setup

memset(&server\_addr, 0, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

server\_addr.sin\_port = htons(SERVER\_PORT);

// Bind the socket to the address and port

if (bind(sockfd, (const struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) {

perror("Bind failed");

close(sockfd);

exit(1);

}

printf("UDP Server is waiting for messages...\n");

// Listen for messages

while (1) {

int n = recvfrom(sockfd, (char \*)buffer, MAX\_BUF\_SIZE, 0, (struct sockaddr \*)&client\_addr, &client\_len);

buffer[n] = '\0';

printf("Server received: %s\n", buffer);

// Respond with "welcome" message

const char \*response = "welcome";

sendto(sockfd, (const char \*)response, strlen(response), 0, (const struct sockaddr \*)&client\_addr, client\_len);

printf("Server sent: %s\n", response);

}

close(sockfd);

return 0;

}

Client Side

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define SERVER\_IP "127.0.0.1"

#define SERVER\_PORT 12345

#define MAX\_BUF\_SIZE 1024

int main() {

int sockfd;

struct sockaddr\_in server\_addr;

char buffer[MAX\_BUF\_SIZE];

// Create UDP socket

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0) {

perror("Socket creation failed");

exit(1);

}

memset(&server\_addr, 0, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(SERVER\_PORT);

server\_addr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP);

// Send "hello" message to the server

const char \*message = "hello";

sendto(sockfd, (const char \*)message, strlen(message), 0, (const struct sockaddr \*)&server\_addr, sizeof(server\_addr));

printf("Client sent: %s\n", message);

// Receive response from server

int n = recvfrom(sockfd, (char \*)buffer, MAX\_BUF\_SIZE, 0, NULL, NULL);

buffer[n] = '\0';

printf("Client received: %s\n", buffer);

close(sockfd);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

DECEMBER 2013

S4 Write and run a client and a server program in C language using TCP in UNIX. The 20 client should send two strings (String 1 and String 2) to the server. The server must append to string 2 to sting 1 and send the find the client. Finally client will print the result.

Answer:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int addrlen = sizeof(address);

char buffer1[BUFFER\_SIZE], buffer2[BUFFER\_SIZE];

char result[BUFFER\_SIZE \* 2];

// Create socket

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("Socket failed");

exit(1);

}

// Set address information

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons(PORT);

// Bind the socket to the address

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

perror("Bind failed");

close(server\_fd);

exit(1);

}

// Listen for incoming connections

if (listen(server\_fd, 3) < 0) {

perror("Listen failed");

close(server\_fd);

exit(1);

}

printf("Waiting for client to connect...\n");

// Accept incoming connections from clients

if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t\*)&addrlen)) < 0) {

perror("Accept failed");

close(server\_fd);

exit(1);

}

// Receive first string (String 1)

read(new\_socket, buffer1, BUFFER\_SIZE);

printf("Received String 1: %s\n", buffer1);

// Receive second string (String 2)

read(new\_socket, buffer2, BUFFER\_SIZE);

printf("Received String 2: %s\n", buffer2);

// Concatenate string 2 to string 1

snprintf(result, sizeof(result), "%s%s", buffer1, buffer2);

// Send the result (concatenated string) back to the client

send(new\_socket, result, strlen(result), 0);

printf("Sent concatenated result to client: %s\n", result);

// Close the connection

close(new\_socket);

close(server\_fd);

return 0;

}

Client Side

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int sock = 0;

struct sockaddr\_in server\_addr;

char buffer1[BUFFER\_SIZE], buffer2[BUFFER\_SIZE], result[BUFFER\_SIZE \* 2];

// Create socket

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

perror("Socket creation failed");

exit(1);

}

// Set server address information

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(PORT);

if (inet\_pton(AF\_INET, "127.0.0.1", &server\_addr.sin\_addr) <= 0) {

perror("Invalid address or address not supported");

exit(1);

}

// Connect to the server

if (connect(sock, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) {

perror("Connection failed");

exit(1);

}

// Input two strings from user

printf("Enter String 1: ");

fgets(buffer1, sizeof(buffer1), stdin);

buffer1[strcspn(buffer1, "\n")] = '\0'; // Remove newline character

printf("Enter String 2: ");

fgets(buffer2, sizeof(buffer2), stdin);

buffer2[strcspn(buffer2, "\n")] = '\0'; // Remove newline character

// Send String 1 and String 2 to the server

send(sock, buffer1, strlen(buffer1), 0);

send(sock, buffer2, strlen(buffer2), 0);

// Receive the concatenated result from the server

int valread = read(sock, result, sizeof(result));

result[valread] = '\0'; // Null-terminate the received string

// Print the result

printf("Received concatenated result: %s\n", result);

// Close the socket

close(sock);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_