**Department of Computer Science and Engineering**

**Subject: Network Programming Code: 18CS72 Marks**: 10

**OBA-1 Submission Date: 13/10/2023**

Q1. Implement Echo Client Server using UDP and demonstrate their working by executing the client server programs with sample input string messages. 5 marks

[L3, PO3, CO3]

**EchoClient.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define SERVER\_ADDRESS "127.0.0.1"

#define BUFFER\_SIZE 1024

int main() {

int sockfd;

char buffer[BUFFER\_SIZE];

struct sockaddr\_in server\_addr;

// Creating socket file descriptor

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) == -1) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

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

// Filling server information

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

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

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

perror("Invalid address/ Address not supported");

exit(EXIT\_FAILURE);

}

while (1) {

printf("Enter message to send: ");

fgets(buffer, BUFFER\_SIZE, stdin);

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

int n = recvfrom(sockfd, (char \*)buffer, BUFFER\_SIZE, MSG\_WAITALL, NULL, NULL);

buffer[n] = '\0';

printf("Echoed message from server: %s\n", buffer);

}

close(sockfd);

return 0;

}

**EchoServer.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 sockfd;

char buffer[BUFFER\_SIZE];

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

socklen\_t len;

// Creating socket file descriptor

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) == -1) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

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

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

// Filling server information

server\_addr.sin\_family = AF\_INET; // IPv4

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)) == -1) {

perror("Bind failed");

exit(EXIT\_FAILURE);

}

printf("UDP Echo Server started on port %d\n", PORT);

while (1) {

len = sizeof(client\_addr);

int n = recvfrom(sockfd, (char \*)buffer, BUFFER\_SIZE, MSG\_WAITALL, (struct sockaddr \*)&client\_addr, &len);

buffer[n] = '\0';

printf("Received message from %s:%d: %s\n", inet\_ntoa(client\_addr.sin\_addr), ntohs(client\_addr.sin\_port), buffer);

sendto(sockfd, (const char \*)buffer, strlen(buffer), MSG\_CONFIRM, (const struct sockaddr \*)&client\_addr, len);

printf("Echoed message sent.\n");

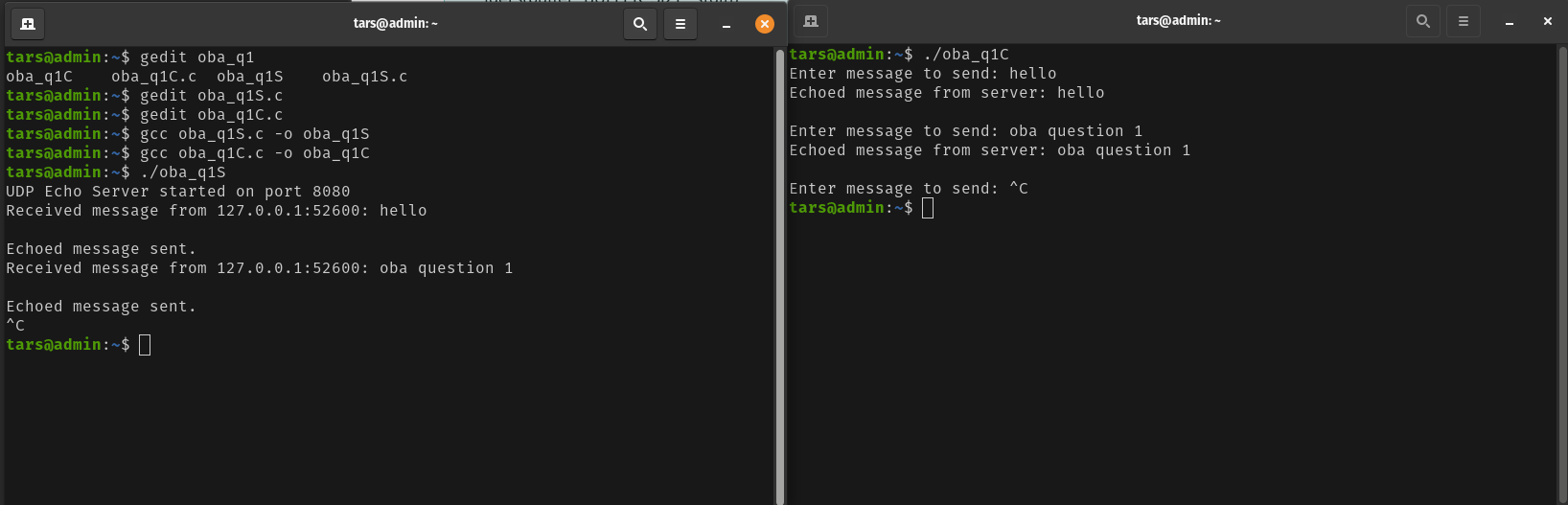
}

close(sockfd);

return 0;

}

**Output:**



Q2. Implement htons, htonl and pton functions to convert data as required in Socket programming.

[L3, PO1, CO1] 5 marks

#include <stdio.h>

#include <stdlib.h>

#include <arpa/inet.h>

unsigned short my\_htons(unsigned short value) {

return (value << 8) | (value >> 8);

}

unsigned int my\_htonl(unsigned int value) {

return ((value & 0xFF) << 24) |

((value & 0xFF00) << 8) |

((value >> 8) & 0xFF00) |

((value >> 24) & 0xFF);

}

int my\_pton(const char \*ip, struct in\_addr \*addr) {

return inet\_pton(AF\_INET, ip, addr);

}

int main() {

unsigned short port = 8080;

unsigned int ip = inet\_addr("192.168.1.1");

printf("Original port: %u\n", port);

printf("Host to Network Short (htons): %u\n", htons(port));

printf("Custom htons: %u\n", my\_htons(port));

printf("\nOriginal IP: %u\n", ip);

printf("Host to Network Long (htonl): %u\n", htonl(ip));

printf("Custom htonl: %u\n", my\_htonl(ip));

struct in\_addr addr;

const char \*ip\_address = "192.168.1.1";

printf("\nOriginal IP address: %s\n", ip\_address);

int pton\_result = my\_pton(ip\_address, &addr);

if (pton\_result == 1) {

printf("inet\_pton result: %s\n", inet\_ntoa(addr));

} else if (pton\_result == 0) {

printf("Invalid address format.\n");

} else {

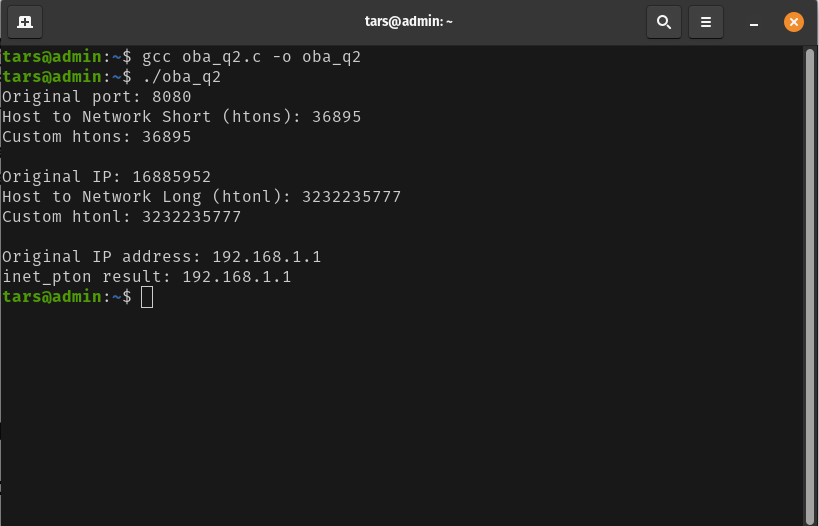
perror("inet\_pton");

}

return 0;

}

Output:



**Department of Computer Science and Engineering**

**Subject: Network Programming Code: 18S72 Marks**: 10

**OBA-2 Submission Date: 30/10/2023**

Q1. Write an IPV4 echo client and IPV6 echo server. Run IPV6 echo server on a host that has dual

protocol stack and test whether, IPV4 echo client works correctly.

. [L3,CO4, PO1] 5 marks

**Client.c**

#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 9000

int main() {

int sockfd;

struct sockaddr\_in server\_addr;

char message[] = "Hello, IPv4 Echo Server!";

char buffer[1024];

// Create a UDP socket

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

if (sockfd < 0) {

perror("socket");

exit(EXIT\_FAILURE);

}

// Set up the server address

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

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

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

inet\_pton(AF\_INET, SERVER\_IP, &server\_addr.sin\_addr);

// Send a message to the server

if (sendto(sockfd, message, strlen(message), 0, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)) < 0) {

perror("sendto");

exit(EXIT\_FAILURE);

}

// Receive the echoed message from the server

memset(buffer, 0, sizeof(buffer));

if (recvfrom(sockfd, buffer, sizeof(buffer), 0, NULL, NULL) < 0) {

perror("recvfrom");

exit(EXIT\_FAILURE);

}

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

close(sockfd);

return 0;

}

**server.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define SERVER\_PORT 9000

int main() {

int sockfd;

struct sockaddr\_in6 server\_addr, client\_addr;

char buffer[1024];

// Create a UDP socket

sockfd = socket(AF\_INET6, SOCK\_DGRAM, 0);

if (sockfd < 0) {

perror("socket");

exit(EXIT\_FAILURE);

}

// Set up the server address

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

server\_addr.sin6\_family = AF\_INET6;

server\_addr.sin6\_port = htons(SERVER\_PORT);

server\_addr.sin6\_addr = in6addr\_any;

// Bind the socket to the server address

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

perror("bind");

exit(EXIT\_FAILURE);

}

printf("IPv6 Echo Server listening on port %d\n", SERVER\_PORT);

// Receive a message from the client and echo it back

while (1) {

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

memset(buffer, 0, sizeof(buffer));

if (recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr\*)&client\_addr, &client\_addr\_len) < 0) {

perror("recvfrom");

exit(EXIT\_FAILURE);

}

// Echo the message back to the client

if (sendto(sockfd, buffer, strlen(buffer), 0, (struct sockaddr\*)&client\_addr, client\_addr\_len) < 0) {

perror("sendto");

exit(EXIT\_FAILURE);

}

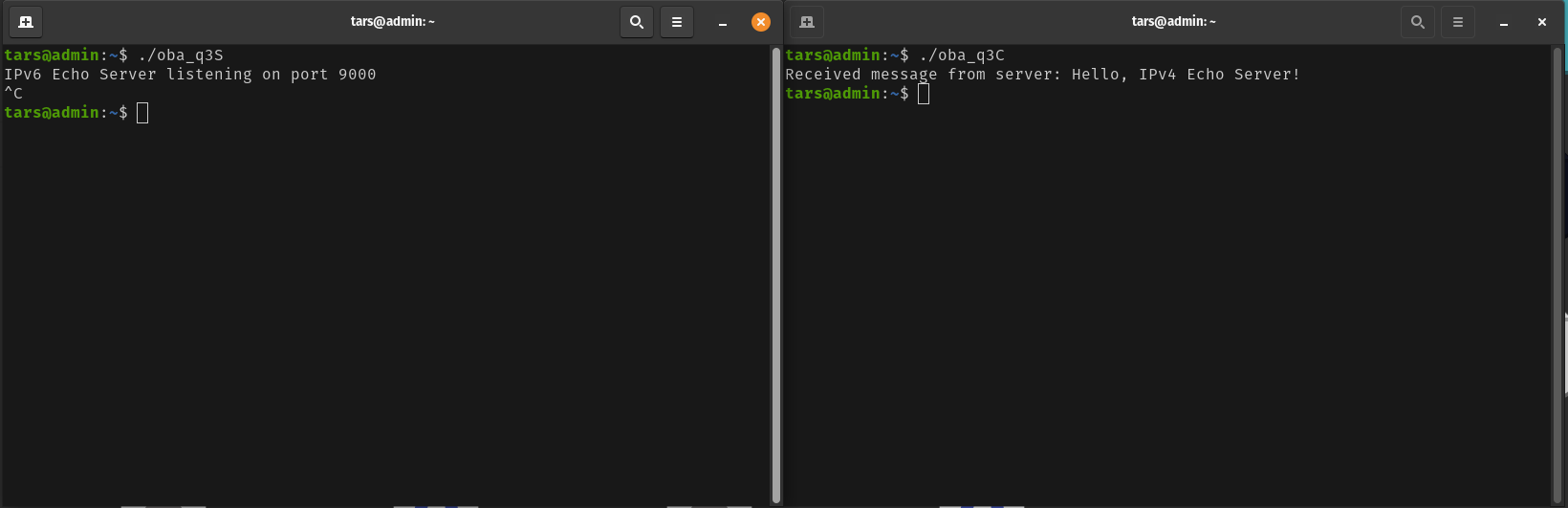
}

close(sockfd);

return 0;

}

Output:



Q2. Write a program to implement Multi-cast sender and receiver and test the programs for their

working. [L3, CO5,PO3] 5 marks

**MultiCast-Sender.c**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <sys/types.h>**

**#include <sys/socket.h>**

**#include <netinet/in.h>**

**#include <arpa/inet.h>**

**int main(int argc, char \*argv[]) {**

**int sockfd;**

**struct sockaddr\_in servaddr, cliaddr;**

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

**if (sockfd < 0) {**

**perror("socket creation failed");**

**exit(EXIT\_FAILURE);**

**}**

**memset(&servaddr, 0, sizeof(servaddr));**

**memset(&cliaddr, 0, sizeof(cliaddr));**

**servaddr.sin\_family = AF\_INET;**

**servaddr.sin\_addr.s\_addr = inet\_addr("224.0.0.1");**

**servaddr.sin\_port = htons(12345);**

**cliaddr.sin\_family = AF\_INET;**

**cliaddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");**

**cliaddr.sin\_port = htons(0);**

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

**perror("bind failed");**

**exit(EXIT\_FAILURE);**

**}**

**char message[1000] = "Hello, this is a multicast message.";**

**if (sendto(sockfd, (const char \*)message, strlen(message), 0, (const struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0) {**

**perror("sendto failed");**

**exit(EXIT\_FAILURE);**

**}**

**printf("Message sent.\n");**

**close(sockfd);**

**return 0;**

**}**

**MultiCast-Reciever.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

int main(int argc, char \*argv[]) {

int sockfd;

struct sockaddr\_in servaddr, cliaddr;

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

if (sockfd < 0) {

perror("socket creation failed");

exit(EXIT\_FAILURE);

}

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

servaddr.sin\_family = AF\_INET;

servaddr.sin\_addr.s\_addr = inet\_addr("224.0.0.1");

servaddr.sin\_port = htons(12345);

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

perror("bind failed");

exit(EXIT\_FAILURE);

}

char buffer[1000];

socklen\_t len = sizeof(cliaddr);

if (recvfrom(sockfd, (char \*)buffer, 1000, 0, (struct sockaddr \*)&cliaddr, &len) < 0) {

perror("recvfrom failed");

exit(EXIT\_FAILURE);

}

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

close(sockfd);

return 0;

}

Output:

