BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering

Name	**Shivam Santosh Kadam**
UID no.	**2023300099**
Experiment No.	**4**

AIM:	Network Socket Programming
OBJECTIVE:	The objective of this experiment is to make students acquainted with socket programming. And make them accustomed with applications executing on top of these sockets.
Part 1	
PROBLEM STATEMENT :	Implement the following rudimentary string processing application using connection-oriented client-server programming. Some guidelines for the implementation are as follows. The client will send a textual paragraph terminated by '\n' to the server (assume that in the paragraph, '.' appears only at the end of sentences and nowhere else). The server will compute the number of characters, number of words, and number of sentences in the paragraph, and send these numbers back to the client. The client will print these numbers on the screen.

THEORY:

Server-Side Execution Flow:

1. **Create Socket (`socket()`)**:
   - The server first creates a socket using the `socket()` function.
   - A socket is an instance that allows communication.
2. **Bind (`bind()`)**:
   - The socket is assigned to a specific IP address and port.
   - This ensures the server listens for incoming connections on the given port.
3. **Listen (`listen()`)**:
   - The server begins listening for client requests.
   - It enables the server to accept multiple incoming connections.
4. **Accept (`accept()`)**:
   - When a client connects, the server accepts the request.
   - A separate socket is created for communication with the client.
5. **Receive (`recv()`)**:

- The server waits for data from the client.
- Data transmission occurs after the connection is established.

6. **Send (`send()`)**:
   - The server sends data back to the client after processing the request.
7. **Receive (`recv()`)**:
   - If further communication is needed, the server continues receiving messages.
8. **Close (`close()`)**:
   - Finally, the connection is closed when communication is complete.

Client-Side Execution Flow:

1. **Create Socket (`socket()`)**:
   - The client creates its own socket instance.
2. **Connect (`connect()`)**:
   - The client sends a connection request to the server.
3. **Send (`send()`)**:
   - After the connection is established, the client sends data to the server.
4. **Receive (`recv()`)**:
   - The client waits for a response from the server.
5. **Close (`close()`)**:
   - The client sends a close message when the communication ends.

PROGRAM:	**Server1.c**

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define PORT 5000
#define BUFFER_SIZE 1024

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
```

```
    socklen_t addr_len = sizeof(address);
    char buffer[BUFFER_SIZE];

    // 1. Create socket
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
{
        perror("socket failed");
        exit(EXIT_FAILURE);
    }
    // 2. Bind to IP/port
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;  // Listen on any
network interface
    address.sin_port = htons(PORT);

    if (bind(server_fd, (struct sockaddr *)&address,
sizeof(address)) < 0) {
        perror("bind failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }
    // 3. Listen for incoming connections
    if (listen(server_fd, 1) < 0) {
        perror("listen failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }
    printf("Server (Part-1) listening on port %d...\n",
PORT);

    while (1) {
        // 4. Accept a client connection
        if ((new_socket = accept(server_fd, (struct
sockaddr *)&address, &addr_len)) < 0) {
            perror("accept failed");
            close(server_fd);
            exit(EXIT_FAILURE);
        }
```

```
        printf("Client connected.\n");

        // 5. Receive data (paragraph)
        memset(buffer, 0, BUFFER_SIZE);
        ssize_t valread = recv(new_socket, buffer,
BUFFER_SIZE - 1, 0);
        if (valread > 0) {
            buffer[valread] = '\0';  // Null-terminate

            // 6. Process: count chars, words, sentences
            int num_chars = 0, num_words = 0, num_sentences
= 0;
            int in_word = 0;

            for (int i = 0; buffer[i] != '\0'; i++) {
                char c = buffer[i];
                // Count characters (excluding '\n' if you
prefer)
                if (c != '\n' && c != '\r') {
                    num_chars++;
                }
                // Count words
                if ((c == ' ' || c == '\n' || c == '\t' ||
c == '\0') && in_word) {
                    num_words++;
                    in_word = 0;
                } else if (c != ' ' && c != '\n' && c !=
'\t' && c != '\0') {
                    in_word = 1;
                }
                // Count sentences (period-based)
                if (c == '.') {
                    num_sentences++;
                }
            }
            // If paragraph didn't end with space, finalize
last word
            if (in_word) num_words++;
```

```
            // 7. Send result back to client
            char result[50];
            snprintf(result, sizeof(result), "%d %d %d",
num_chars, num_words, num_sentences);
            send(new_socket, result, strlen(result), 0);
        }
        // 8. Close connection
        close(new_socket);
        printf("Client disconnected.\n");
    }
    // 9. Close the server socket (unreachable in this
infinite loop example)
    close(server_fd);
    return 0;
}
```

Client1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define PORT 5000
#define BUFFER_SIZE 1024

int main() {
    int sock = 0;
    struct sockaddr_in serv_addr;
    char buffer[BUFFER_SIZE];

    // 1. Create socket
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("Socket creation error");
        exit(EXIT_FAILURE);
```

```
    }

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

    // Convert IPv4/IPv6 addresses from text to binary form
    if (inet_pton(AF_INET, "10.10.60.250",
&serv_addr.sin_addr) <= 0) {
        perror("Invalid address/ Address not supported");
        close(sock);
        exit(EXIT_FAILURE);
    }
    // 2. Connect to server
    if (connect(sock, (struct sockaddr *)&serv_addr,
sizeof(serv_addr)) < 0) {
        perror("Connection Failed");
        close(sock);
        exit(EXIT_FAILURE);
    }
    printf("Connected to server (Part-1).\n");

    // 3. Input paragraph
    printf("Enter a paragraph (end with Enter):\n");
    fgets(buffer, BUFFER_SIZE, stdin);
    // 4. Send paragraph
    send(sock, buffer, strlen(buffer), 0);
    // 5. Receive result
    memset(buffer, 0, BUFFER_SIZE);
    ssize_t valread = recv(sock, buffer, BUFFER_SIZE - 1,
0);
    if (valread > 0) {
        buffer[valread] = '\0';
        int num_chars, num_words, num_sentences;
        sscanf(buffer, "%d %d %d", &num_chars, &num_words,
&num_sentences);
        printf("Number of characters: %d\n", num_chars);
        printf("Number of words: %d\n", num_words);
        printf("Number of sentences: %d\n", num_sentences);
```

```
    }
    close(sock);
    return 0;
}
```

RESULT:

Server

```
students@students-ThinkCentre-M720e:~/Downloads/ccn$ gcc server1.c -o s1
students@students-ThinkCentre-M720e:~/Downloads/ccn$ gcc client1.c -o c1
students@students-ThinkCentre-M720e:~/Downloads/ccn$ ./s1
Server (Part-1) listening on port 5000...
Client connected.
Client disconnected.
^C
```

```
students@students-ThinkCentre-M720e:~/Downloads/ccn$ ifconfig
enp1s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.10.60.250  netmask 255.255.254.0  broadcast 10.10.61.255
        inet6 fe80::3cd5:35e3:6611:792f  prefixlen 64  scopeid 0x20<link>
        ether a4:ae:11:1d:9a:76  txqueuelen 1000  (Ethernet)
        RX packets 109742  bytes 68025466 (68.0 MB)
        RX errors 0  dropped 2519  overruns 0  frame 0
        TX packets 29103  bytes 7006135 (7.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2727  bytes 407800 (407.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2727  bytes 407800 (407.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Client

```
students@spit:~/Documents$ gcc shivam.c -o shivam
students@spit:~/Documents$ ./shivam
Connected to server (Part-1).
Enter a paragraph (end with Enter):
My name is Anish. My best friend are shivam and ruchir.My hobby is playing chess.
Number of characters: 81
Number of words: 15
Number of sentences: 3
```

```
students@spit:~/Documents$ ifconfig
enp1s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.10.61.197  netmask 255.255.254.0  broadcast 10.10.61.255
        inet6 fe80::507a:8f67:2b80:a6e  prefixlen 64  scopeid 0x20<link>
        ether a4:ae:12:27:ec:53  txqueuelen 1000  (Ethernet)
        RX packets 399984  bytes 273320148 (273.3 MB)
        RX errors 0  dropped 10084  overruns 0  frame 0
        TX packets 80885  bytes 17665363 (17.6 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 6086  bytes 1210439 (1.2 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6086  bytes 1210439 (1.2 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

students@spit:~/Documents$
```

Part 2

PROBLEM STATEMENT :	Make it concurrent so that it can serve multiple clients at a time. (Multiple clients on multiple terminals and single server terminals)
PROGRAM:	**Server2.c**

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <pthread.h>

#define PORT 5001
#define BUFFER_SIZE 1024
#define MAX_CLIENTS 10  // Up to 10 concurrent clients

// Thread function to handle each client
void* handle_client(void* arg) {
   int client_socket = *(int*)arg;
   free(arg); // free the dynamically allocated socket
pointer

   char buffer[BUFFER_SIZE];
```

```
    memset(buffer, 0, BUFFER_SIZE);

    ssize_t valread = recv(client_socket, buffer,
BUFFER_SIZE - 1, 0);
    if (valread > 0) {
        buffer[valread] = '\0';
        // Count chars, words, sentences
        int num_chars = 0, num_words = 0, num_sentences =
0;
        int in_word = 0;
        for (int i = 0; buffer[i] != '\0'; i++) {
            char c = buffer[i];
            if (c != '\n' && c != '\r') {
                num_chars++;
            }
            if ((c == ' ' || c == '\n' || c == '\t' || c ==
'\0') && in_word) {
                num_words++;
                in_word = 0;
            } else if (c != ' ' && c != '\n' && c != '\t'
&& c != '\0') {
                in_word = 1;
            }
            if (c == '.') {
                num_sentences++;
            }
        }
        if (in_word) num_words++;

        char result[50];
        snprintf(result, sizeof(result), "%d %d %d",
num_chars, num_words, num_sentences);
        send(client_socket, result, strlen(result), 0);
    }

    close(client_socket);
    pthread_exit(NULL);
}
```

```c
int main() {
    int server_fd;
    struct sockaddr_in address;
    socklen_t addr_len = sizeof(address);

    // 1. Create socket
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
{
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    // 2. Bind
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    if (bind(server_fd, (struct sockaddr*)&address,
sizeof(address)) < 0) {
        perror("bind failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    // 3. Listen
    if (listen(server_fd, MAX_CLIENTS) < 0) {
        perror("listen failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }
    printf("Concurrent Server (Part-2) listening on port
%d...\n", PORT);

    while (1) {
        // 4. Accept
        int* new_socket = malloc(sizeof(int));
        if ((*new_socket = accept(server_fd, (struct
```

```
sockaddr*)&address, &addr_len)) < 0) {
            perror("accept failed");
            free(new_socket);
            continue; // keep server running
        }
        printf("New client connected.\n");

        // 5. Create a thread for the new client
        pthread_t tid;
        if (pthread_create(&tid, NULL, handle_client,
(void*)new_socket) != 0) {
            perror("pthread_create");
            close(*new_socket);
            free(new_socket);
        }
        // Detach thread so resources are released when it
ends
        pthread_detach(tid);
    }

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

Client2.c (same as Client1.c**)**

RESULT:

Server

```
students@students-ThinkCentre-M720e:~/Downloads/ccn$ gcc server2.c -o server2 -pthread
students@students-ThinkCentre-M720e:~/Downloads/ccn$ ./server2
Concurrent Server (Part-2) listening on port 5001...
New client connected.
New client connected.
```

Client1

```
students@spit:~/Downloads$ gcc client2.c
students@spit:~/Downloads$ ./a.out
Connected to concurrent server (Part-2).
Enter a paragraph:
palash here.client
Number of characters: 18
Number of words: 2
Number of sentences: 1
students@spit:~/Downloads$ ifconfig
enp4s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.10.61.125  netmask 255.255.254.0  broadcast 10.10.61.255
        inet6 fe80::88b5:79a7:9a57:e823  prefixlen 64  scopeid 0x20<link>
        ether f4:6b:8c:d0:a6:2f  txqueuelen 1000  (Ethernet)
        RX packets 187996  bytes 157024306 (157.0 MB)
        RX errors 0  dropped 2981  overruns 0  frame 0
        TX packets 73349  bytes 13652184 (13.6 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 3926  bytes 363279 (363.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3926  bytes 363279 (363.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp3s0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 9c:a2:f4:a5:b6:ab  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Client2

```
students@spit:~/Documents$ ./client2
Connected to concurrent server (Part-2).
Enter a paragraph:
I am anish My friends are ruchir and shivam.sO LET IT BE.pLEASES CONSIDER THIS AS
A SAMPLE PARAGRAPH.
Number of characters: 101
Number of words: 18
Number of sentences: 3
students@spit:~/Documents$
```

OBSERVATIONS:

- Socket programming enables communication between a client and a server over a network.
- The server processes textual data by counting characters, words, and sentences.
- Using threads allows the server to handle multiple clients concurrently.
- Proper handling of socket creation, binding, and data transmission is crucial for stability.
-

CONCLUSION:	In this experiment, I learned how to implement network socket programming using connection-oriented client-server communication. I developed a basic string processing application and extended it to support multiple concurrent clients. The experiment helped me understand socket operations, data transmission, and multithreading in networking.