CSE-7B
Distributed Systems (Lab)
Assignment 6

1. Implement concurrent echo client-server application.

Server.c

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

#define PORT 12347
#define BUFFER_SIZE 1024

int main() {
   int server_socket, client_socket;
   struct sockaddr_in server_addr, client_addr;
   socklen_t client_addr_len = sizeof(client_addr);
   char buffer[BUFFER_SIZE];
   ssize_t bytes_received;

   server_socket = socket(AF_INET, SOCK_STREAM, 0);
   if (server_socket < 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);

   if (bind(server_socket, (struct sockaddr*)&server_addr,
sizeof(server_addr)) < 0) {
       perror("Bind failed");
       close(server_socket);
       exit(EXIT_FAILURE);
```

```
    }

    if (listen(server_socket, 1) < 0) {
        perror("Listen failed");
        close(server_socket);
        exit(EXIT_FAILURE);
    }

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

    while (1) {
        client_socket = accept(server_socket, (struct
sockaddr*)&client_addr, &client_addr_len);
        if (client_socket < 0) {
            perror("Accept failed");
            continue;
        }

        printf("Accepted connection from %s\n",
inet_ntoa(client_addr.sin_addr));

        while ((bytes_received = recv(client_socket, buffer, sizeof(buffer)
- 1, 0)) > 0) {
            buffer[bytes_received] = '\0';  // Null-terminate the received
data
            fgets(buffer, sizeof(buffer), stdout);
            send(client_socket, buffer, bytes_received, 0);  // Echo the
data back
        }

        close(client_socket);
    }

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

Client.c

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

#define PORT 12347
#define BUFFER_SIZE 1024

int main() {
    int client_socket;
    struct sockaddr_in server_addr;
    char buffer[BUFFER_SIZE];
    ssize_t bytes_sent, bytes_received;

    client_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (client_socket < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    server_addr.sin_port = htons(PORT);

    if (connect(client_socket, (struct sockaddr*)&server_addr,
sizeof(server_addr)) < 0) {
        perror("Connection failed");
        close(client_socket);
        exit(EXIT_FAILURE);
    }

    while (1) {
        printf("Send message: ");
        fgets(buffer, sizeof(buffer), stdin);
        buffer[strcspn(buffer, "\n")] = '\0';  // Remove newline character

        bytes_sent = send(client_socket, buffer, strlen(buffer), 0);
        if (bytes_sent < 0) {
```

```
            perror("Send failed");
            break;
        }

        bytes_received = recv(client_socket, buffer, sizeof(buffer) - 1,
0);
        if (bytes_received < 0) {
            perror("Receive failed");
            break;
        }

        buffer[bytes_received] = '\0';  // Null-terminate the received data
        printf("Received: %s\n", buffer);
    }

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

Output:

```
bash-4.2$ gcc server1.c
bash-4.2$ ./a.out
Server listening on port 12347
Accepted connection from 127.0.0.1
_
```

```
bash-4.2$ gcc client1.c
bash-4.2$ ./a.out
Send message: hii
Received: hii
Send message: hello
Received: hello
Send message: _
```

2.Implement a client-server program in which the server accepts a connection from a client and updates its own Master table by adding the client information and send the updated table to client, so client can update their own table. Refer the following table format.

Node No.	IP address	Port No.
1	172.31.100.36	2345
2	172.31.100.40	3128
3	172.31.100.52	2323

Server.c

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

#define PORT 12349
#define MAX_CLIENTS 100
#define BUFFER_SIZE 1024

typedef struct {
    char node[20];
    char ip[INET_ADDRSTRLEN];
    int port;
} TableEntry;

TableEntry server_table[MAX_CLIENTS];
int table_size = 0;

void update_table(const char* ip, int port) {
    // Check if the IP and port are already in the table
    int i;
    for (i = 0; i < table_size; i++) {
```

```
        if (strcmp(server_table[i].ip, ip) == 0 && server_table[i].port ==
port) {
            return;
        }
    }

    // Add new entry to the table
    snprintf(server_table[table_size].node,
sizeof(server_table[table_size].node), "Client-%d", table_size + 1);
    strncpy(server_table[table_size].ip, ip,
sizeof(server_table[table_size].ip));
    server_table[table_size].port = port;
    table_size++;
}

void print_table() {
    printf("Master Table:\n");
    int i;
    for (i = 0; i < table_size; i++) {
        printf("Node: %s, IP: %s, Port: %d\n", server_table[i].node,
server_table[i].ip, server_table[i].port);
    }
    printf("\n");
}

void send_table(int client_socket) {
    write(client_socket, &table_size, sizeof(int));  // Send table size
    write(client_socket, server_table, sizeof(TableEntry) * table_size);
// Send table entries
}

int main() {
    int server_socket, client_socket;
    struct sockaddr_in server_addr, client_addr;
    socklen_t client_addr_len = sizeof(client_addr);

    server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket < 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);

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

    if (listen(server_socket, 5) < 0) {
        perror("Listen failed");
        close(server_socket);
        exit(EXIT_FAILURE);
    }

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

    while (1) {
        client_socket = accept(server_socket, (struct
sockaddr*)&client_addr, &client_addr_len);
        if (client_socket < 0) {
            perror("Accept failed");
            continue;
        }

        char client_ip[INET_ADDRSTRLEN];
        inet_ntop(AF_INET, &client_addr.sin_addr, client_ip,
sizeof(client_ip));
        int client_port = ntohs(client_addr.sin_port);

        update_table(client_ip, client_port);

        // Print the updated master table
        print_table();

        send_table(client_socket);
```

```
        close(client_socket);
    }

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

Client.c

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

#define PORT 12349
#define BUFFER_SIZE 1024

typedef struct {
    char node[20];
    char ip[INET_ADDRSTRLEN];
    int port;
} TableEntry;

void receive_table(int server_socket) {
    int table_size;
    read(server_socket, &table_size, sizeof(int));  // Read table size

    TableEntry* table = (TableEntry*)malloc(sizeof(TableEntry) *
table_size);
    if (table == NULL) {
        perror("Memory allocation failed");
        exit(EXIT_FAILURE);
    }
```

```
    read(server_socket, table, sizeof(TableEntry) * table_size);  // Read
table entries

    printf("Received table:\n");
    int i;
    for (i = 0; i < table_size; i++) {
        printf("Node: %s, IP: %s, Port: %d\n", table[i].node, table[i].ip,
table[i].port);
    }

    free(table);
}

int main() {
    int client_socket;
    struct sockaddr_in server_addr;

    client_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (client_socket < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    if (connect(client_socket, (struct sockaddr*)&server_addr,
sizeof(server_addr)) < 0) {
        perror("Connect failed");
        close(client_socket);
        exit(EXIT_FAILURE);
    }

    receive_table(client_socket);

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

Output:

```
bash-4.2$ gcc server2.c
bash-4.2$ ./a.out
Server is listening on port 12349
Master Table:
Node: Client-1, IP: 127.0.0.1, Port: 41290

Master Table:
Node: Client-1, IP: 127.0.0.1, Port: 41290
Node: Client-2, IP: 127.0.0.1, Port: 41292
```

```
bash-4.2$ gcc server2.c
bash-4.2$ ./a.out
Server is listening on port 12349
Master Table:
Node: Client-1, IP: 127.0.0.1, Port: 41290

Master Table:
Node: Client-1, IP: 127.0.0.1, Port: 41290
Node: Client-2, IP: 127.0.0.1, Port: 41292
```

3.Develop a client-server program to implement a date-time server and client. Upon connection establishment, the server should send its current date, time and CPU load information to its clients.

Server.c

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

#define PORT 12350
#define BUFFER_SIZE 256

void get_server_info(char *info_buffer, size_t buffer_size) {
   // Get current time
   time_t now = time(NULL);
   struct tm *tm_info = localtime(&now);

   char time_str[64];
   strftime(time_str, sizeof(time_str), "%Y-%m-%d %H:%M:%S", tm_info);

   // Get CPU load
   struct sysinfo sys_info;
   if (sysinfo(&sys_info) != 0) {
       perror("sysinfo failed");
       snprintf(info_buffer, buffer_size, "Failed to retrieve system
info.");
       return;
   }

   float loadavg = (float)sys_info.loads[0] / 65536.0; // Convert to load
average in format (0.00)
   snprintf(info_buffer, buffer_size, "Date/Time: %s\nCPU Load: %.2f\n",
time_str, loadavg);
```

```
}

void handle_client(int client_socket) {
    char info_buffer[BUFFER_SIZE];
    get_server_info(info_buffer, sizeof(info_buffer));
    send(client_socket, info_buffer, strlen(info_buffer), 0);
    close(client_socket);
}

int main() {
    int server_socket, client_socket;
    struct sockaddr_in server_addr, client_addr;
    socklen_t client_addr_len = sizeof(client_addr);

    server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket < 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);

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

    if (listen(server_socket, 5) < 0) {
        perror("Listen failed");
        close(server_socket);
        exit(EXIT_FAILURE);
    }

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

    while (1) {
```

```
        client_socket = accept(server_socket, (struct
sockaddr*)&client_addr, &client_addr_len);
        if (client_socket < 0) {
            perror("Accept failed");
            continue;
        }

        handle_client(client_socket);
    }

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

Client.c

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

#define PORT 12350
#define BUFFER_SIZE 256

int main() {
    int client_socket;
    struct sockaddr_in server_addr;
    char buffer[BUFFER_SIZE];

    client_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (client_socket < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    server_addr.sin_family = AF_INET;
```

```
    server_addr.sin_port = htons(PORT);
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    if (connect(client_socket, (struct sockaddr*)&server_addr,
sizeof(server_addr)) < 0) {
        perror("Connect failed");
        close(client_socket);
        exit(EXIT_FAILURE);
    }

    int bytes_received = recv(client_socket, buffer, sizeof(buffer) - 1,
0);
    if (bytes_received < 0) {
        perror("Receive failed");
        close(client_socket);
        exit(EXIT_FAILURE);
    }

    buffer[bytes_received] = '\0'; // Null-terminate the received data
    printf("Received from server:\n%s", buffer);

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

Output:

```
bash-4.2$ gcc server3.c
bash-4.2$ ./a.out
Server is listening on port 12350
```

```
bash-4.2$ gcc client3.c
bash-4.2$ ./a.out
Received from server:
Date/Time: 2024-09-17 09:59:21
CPU Load: 0.62
bash-4.2$ _
```