ASSIGNMENT 6
17/09/24

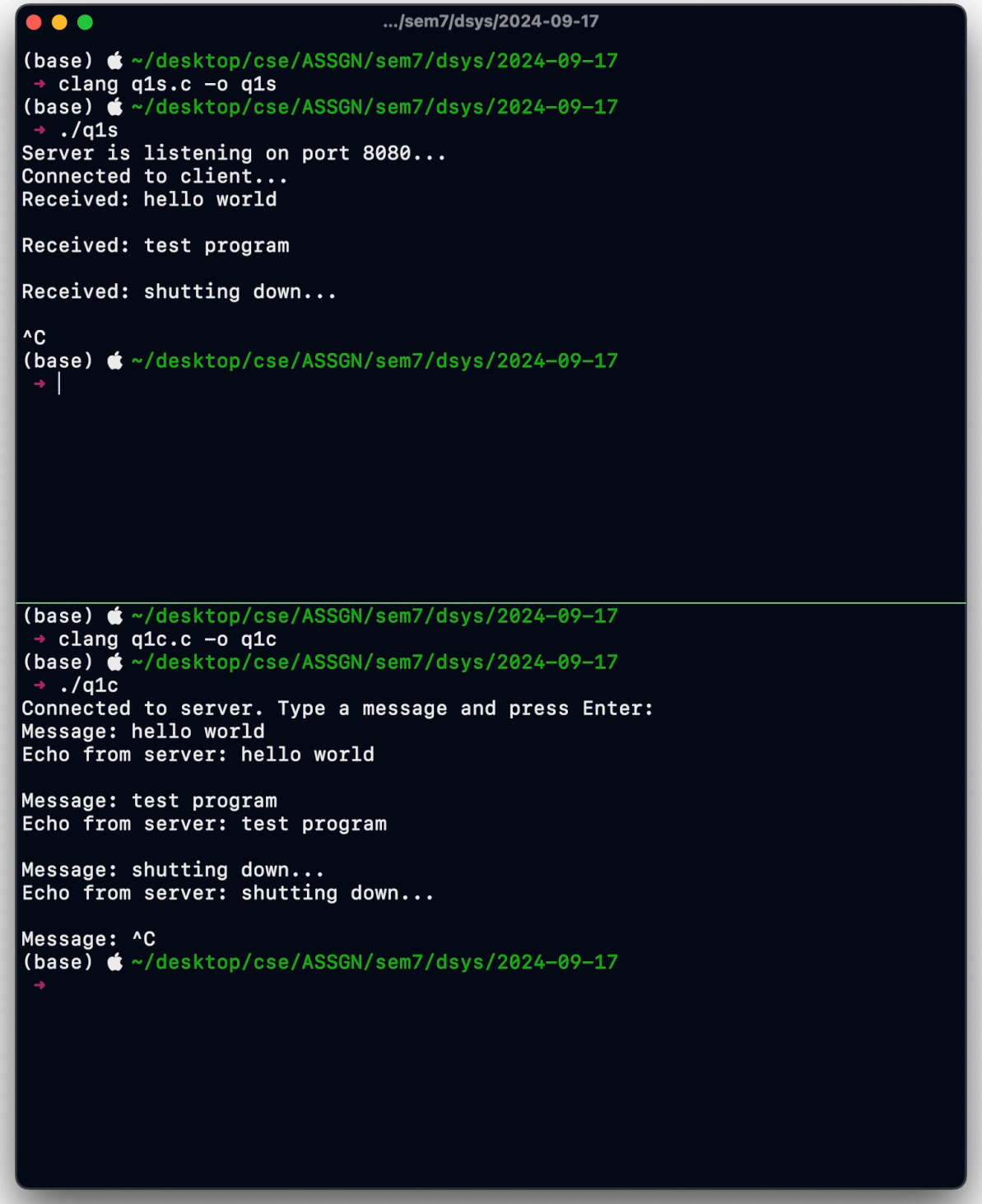
NAME : SHRESTH SONKAR
REGNO : 20214272
GROUP : CS7D
TOPIC : DISTRIBUTED SYSTEM
CODE : CS-17201

```c
// echo server

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

#define PORT 8080
#define BUFFER_SIZE 1024

void handle_client(int client_socket) {
    char buffer[BUFFER_SIZE];
    int bytes_received;

    while ((bytes_received = recv(client_socket,
buffer, BUFFER_SIZE, 0)) > 0) {
        buffer[bytes_received] = '\0';
        printf("Received: %s\n", buffer);
        send(client_socket, buffer, bytes_received, 0);
    }

    close(client_socket);
}

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

    if ((server_socket = socket(AF_INET, SOCK_STREAM,
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);
```

```c
    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, 10) < 0) {
        perror("Listen failed");
        close(server_socket);
        exit(EXIT_FAILURE);
    }

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

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

        printf("Connected to client...\n");

        if ((child_pid = fork()) == 0) {
            close(server_socket);
            handle_client(client_socket);
            exit(0);
        } else if (child_pid > 0) {
            close(client_socket);
        } else {
            perror("Fork failed");
            close(client_socket);
        }
    }
    return 0;
}

// Echo client

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

```c
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define PORT 8080
#define BUFFER_SIZE 1024

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

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

    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(EXIT_FAILURE);
    }

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

    printf("Connected to server. Type a message and
press Enter:\n");

    while (1) {
        printf("Message: ");
        fgets(buffer, BUFFER_SIZE, stdin);

        send(client_socket, buffer, strlen(buffer), 0);
```

```c
        if ((bytes_received = recv(client_socket,
buffer, BUFFER_SIZE, 0)) > 0) {
            buffer[bytes_received] = '\0';
            printf("Echo from server: %s\n", buffer);
        }
    }

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

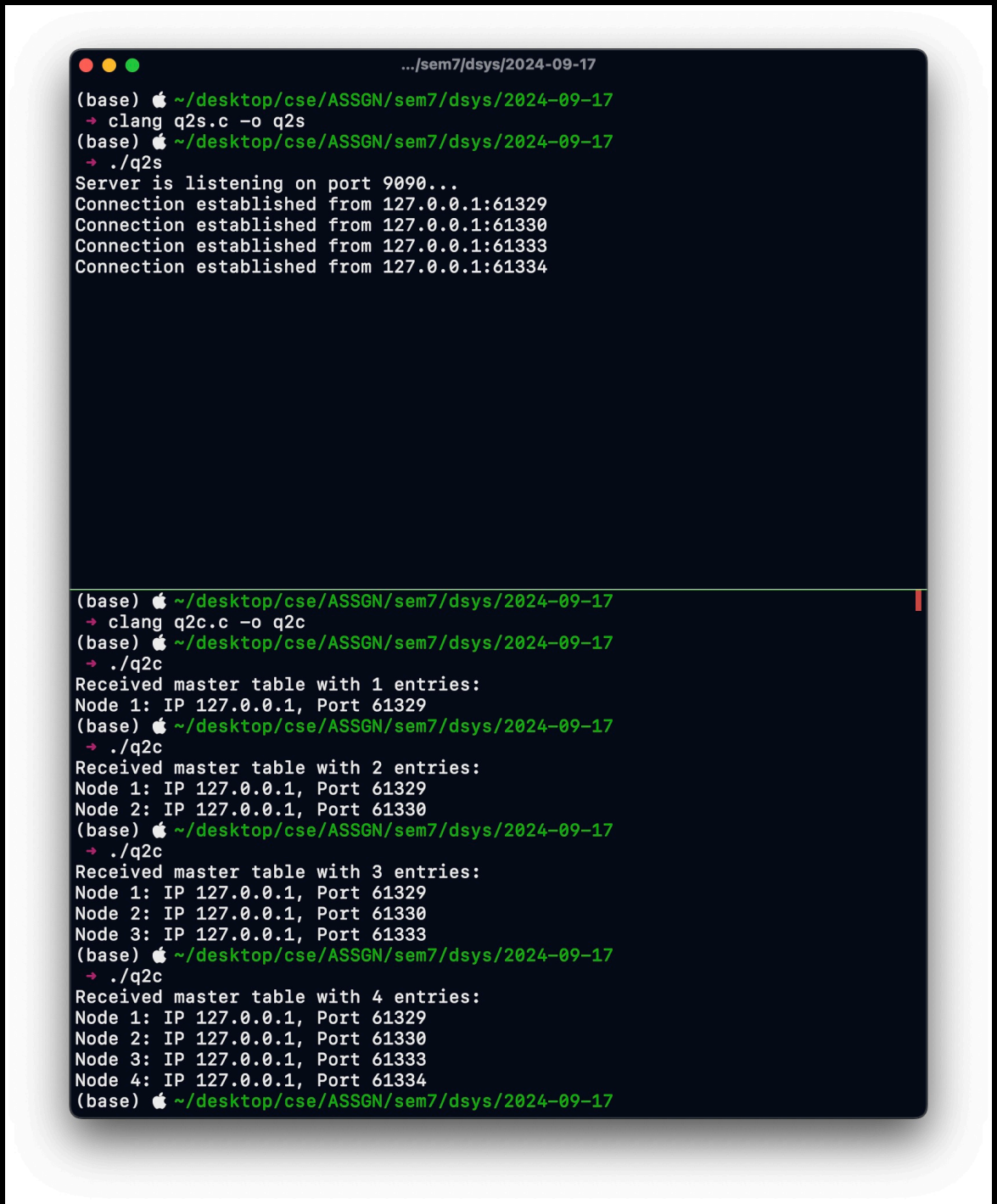


```c
// master routing table server

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

#define PORT 9090
#define MAX_CLIENTS 10

typedef struct {
    int node_no;
    char ip_address[INET_ADDRSTRLEN];
    int port_no;
} ClientInfo;

ClientInfo master_table[MAX_CLIENTS];
int num_clients = 0;

void add_client_to_master(int client_sock, struct
sockaddr_in client_addr) {
    inet_ntop(AF_INET, &client_addr.sin_addr,
master_table[num_clients].ip_address, INET_ADDRSTRLEN);
    master_table[num_clients].port_no =
ntohs(client_addr.sin_port);
    master_table[num_clients].node_no = num_clients +
1;
    num_clients++;
}

void send_master_table(int client_sock) {
    write(client_sock, &num_clients,
sizeof(num_clients));
    for (int i = 0; i < num_clients; i++) {
        write(client_sock, &master_table[i],
sizeof(master_table[i]));
    }
}

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

```
    int opt = 1;
    int addrlen = sizeof(address);

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

    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR,
&opt, sizeof(opt))) {
        perror("setsockopt");
        exit(EXIT_FAILURE);
    }
    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");
        exit(EXIT_FAILURE);
    }
    if (listen(server_fd, 3) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);
    }
    printf("Server is listening on port %d...\n",
PORT);

    while ((new_socket = accept(server_fd, (struct
sockaddr *) &address, (socklen_t * ) & addrlen))) {
        printf("Connection established from %s:%d\n",
inet_ntoa(address.sin_addr), ntohs(address.sin_port));
        add_client_to_master(new_socket, address);
        send_master_table(new_socket);
        close(new_socket);
    }

    return 0;
}
```

```c
// master routing table client

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

#define SERVER_PORT 9090
#define SERVER_IP "127.0.0.1"

typedef struct {
    int node_no;
    char ip_address[INET_ADDRSTRLEN];
    int port_no;
} ClientInfo;

int main() {
    int sock = 0;
    struct sockaddr_in serv_addr;
    ClientInfo master_table[10];
    int num_entries;

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\n Socket creation error \n");
        return -1;
    }

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

    if (inet_pton(AF_INET, SERVER_IP,
&serv_addr.sin_addr) <= 0) {
        printf("\nInvalid address/ Address not
supported \n");
        return -1;
    }

    if (connect(sock, (struct sockaddr *) &serv_addr,
sizeof(serv_addr)) < 0) {
        printf("\nConnection Failed \n");
        return -1;
    }
```

```c
    read(sock, &num_entries, sizeof(num_entries));
    printf("Received master table with %d entries:\n",
num_entries);
    for (int i = 0; i < num_entries; i++) {
        read(sock, &master_table[i],
sizeof(master_table[i]));
        printf("Node %d: IP %s, Port %d\n",
master_table[i].node_no, master_table[i].ip_address,
               master_table[i].port_no);
    }

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

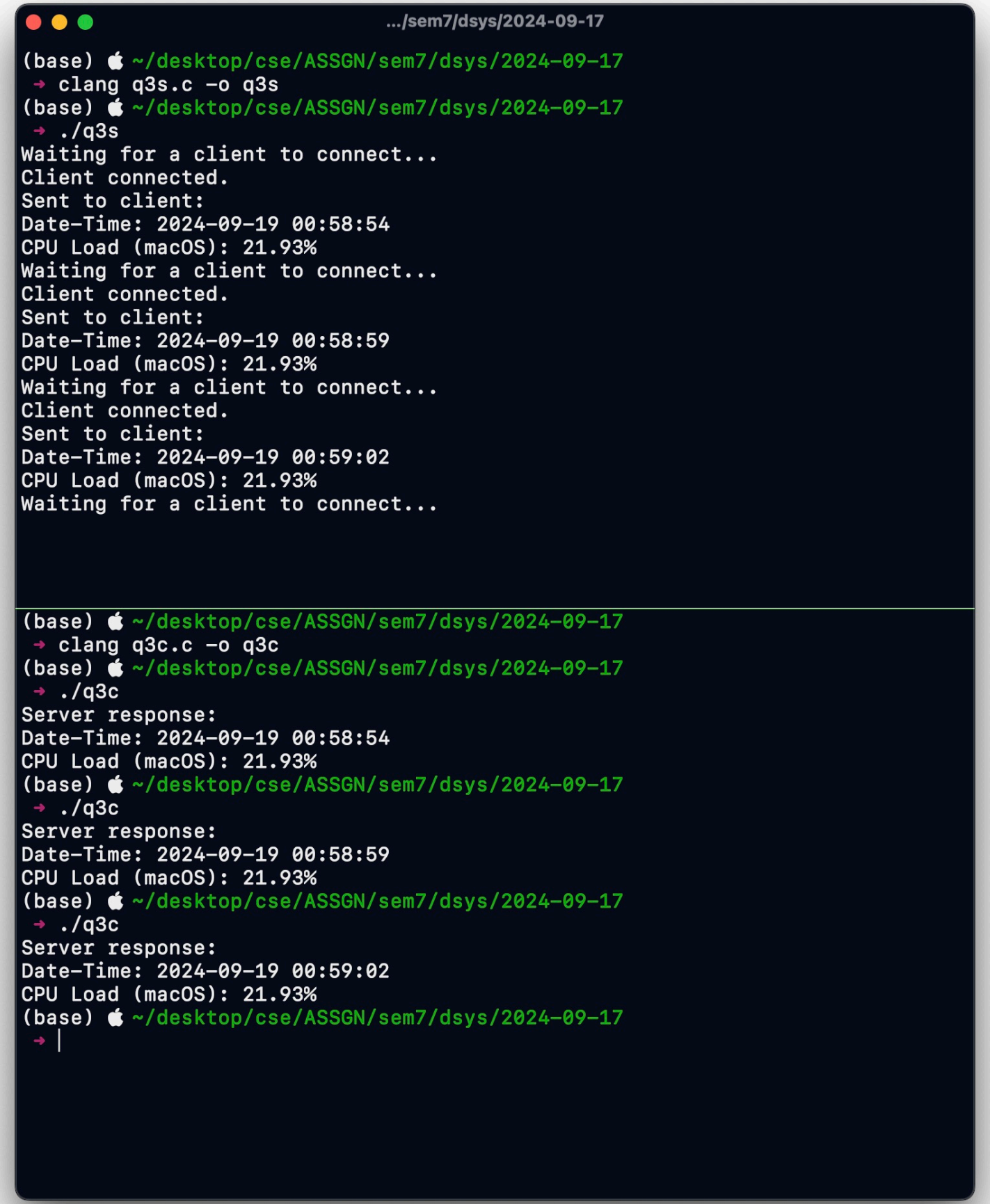


```c
// Date time CPU load server

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

#include <mach/mach.h>

#define PORT 8080
#define BUFFER_SIZE 1024

void get_datetime(char *buffer) {
    time_t t;
    struct tm *tmp;
    char time_str[64];

    t = time(NULL);
    tmp = localtime(&t);

    if (tmp == NULL) {
        perror("localtime");
        exit(EXIT_FAILURE);
    }

    if (strftime(time_str, sizeof(time_str), "%Y-%m-%d
%H:%M:%S", tmp) == 0) {
        fprintf(stderr, "strftime returned 0");
        exit(EXIT_FAILURE);
    }

    strcpy(buffer, time_str);
}

void get_cpu_load_macos(char *buffer) {
    natural_t cpu_count;
    processor_info_array_t cpu_info;
    mach_msg_type_number_t num_cpu_info;
    kern_return_t kr;
```

```
    kr = host_processor_info(mach_host_self(),
PROCESSOR_CPU_LOAD_INFO, &cpu_count, &cpu_info,
&num_cpu_info);
    if (kr != KERN_SUCCESS) {
        strcpy(buffer, "Failed to get CPU load info");
        return;
    }

    unsigned long long total_ticks = 0, idle_ticks = 0;
    for (unsigned int i = 0; i < cpu_count; i++) {
        natural_t *cpu_load = (natural_t *)&cpu_info[i
* CPU_STATE_MAX];
        idle_ticks += cpu_load[CPU_STATE_IDLE];
        for (int j = 0; j < CPU_STATE_MAX; j++) {
            total_ticks += cpu_load[j];
        }
    }

    vm_deallocate(mach_task_self(),
(vm_address_t)cpu_info, num_cpu_info *
sizeof(natural_t));

    double cpu_usage = (1.0 - ((double)idle_ticks /
total_ticks)) * 100.0;
    sprintf(buffer, "CPU Load (macOS): %.2f%%",
cpu_usage);
}

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);
    char buffer[BUFFER_SIZE] = {0};
    char datetime[64], cpu_load[64];

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

    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR,
&opt, sizeof(opt))) {
        perror("setsockopt SO_REUSEADDR failed");
```

```
        exit(EXIT_FAILURE);
    }

    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");
        exit(EXIT_FAILURE);
    }

    if (listen(server_fd, 3) < 0) {
        perror("listen failed");
        exit(EXIT_FAILURE);
    }

    while (1) {
        printf("Waiting for a client to connect...\n");
        if ((new_socket = accept(server_fd, (struct
sockaddr *) &address, (socklen_t * ) & addrlen)) < 0) {
            perror("accept failed");
            exit(EXIT_FAILURE);
        }

        printf("Client connected.\n");
        get_datetime(datetime);
        get_cpu_load_macos(cpu_load);

        snprintf(buffer, sizeof(buffer), "Date-Time:
%s\n%s\n", datetime, cpu_load);
        send(new_socket, buffer, strlen(buffer), 0);
        printf("Sent to client:\n%s", buffer);

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

```c
// Date time CPU load client

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

#define PORT 8080
#define BUFFER_SIZE 1024

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

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\n Socket creation error \n");
        return -1;
    }

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

    if (inet_pton(AF_INET, "127.0.0.1",
&serv_addr.sin_addr) <= 0) {
        printf("\nInvalid address/ Address not
supported \n");
        return -1;
    }

    if (connect(sock, (struct sockaddr *) &serv_addr,
sizeof(serv_addr)) < 0) {
        printf("\nConnection Failed \n");
        return -1;
    }

    read(sock, buffer, BUFFER_SIZE);
    printf("Server response:\n%s", buffer);

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


.../sem7/dsys/2024-09-17
(base) ~/desktop/cse/ASSGN/sem7/dsys/2024-09-17
→ clang q3s.c -o q3s
(base) ~/desktop/cse/ASSGN/sem7/dsys/2024-09-17
→ ./q3s
Waiting for a client to connect...
Client connected.
Sent to client:
Date-Time: 2024-09-19 00:58:54
CPU Load (macOS): 21.93%
Waiting for a client to connect...
Client connected.
Sent to client:
Date-Time: 2024-09-19 00:58:59
CPU Load (macOS): 21.93%
Waiting for a client to connect...
Client connected.
Sent to client:
Date-Time: 2024-09-19 00:59:02
CPU Load (macOS): 21.93%
Waiting for a client to connect...
(base) ~/desktop/cse/ASSGN/sem7/dsys/2024-09-17
→ clang q3c.c -o q3c
(base) ~/desktop/cse/ASSGN/sem7/dsys/2024-09-17
→ ./q3c
Server response:
Date-Time: 2024-09-19 00:58:54
CPU Load (macOS): 21.93%
(base) ~/desktop/cse/ASSGN/sem7/dsys/2024-09-17
→ ./q3c
Server response:
Date-Time: 2024-09-19 00:58:59
CPU Load (macOS): 21.93%
(base) ~/desktop/cse/ASSGN/sem7/dsys/2024-09-17
→ ./q3c
Server response:
Date-Time: 2024-09-19 00:59:02
CPU Load (macOS): 21.93%
(base) ~/desktop/cse/ASSGN/sem7/dsys/2024-09-17
→