BIDDING ITEMS

Server Code:

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

#define PORT 12345
#define BUFFER_SIZE 1024

typedef struct {
    char item[20];
    int price;
    int bidder_fd;
} Item;

Item items[] = {
    {"item1", 100, -1},
    {"item2", 200, -1},
};

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

void* handle_client(void* arg) {
    int client_fd = *((int*)arg);
    free(arg);

    while (1) {
        char buffer[BUFFER_SIZE];
        int bytes_received = recv(client_fd, buffer, BUFFER_SIZE, 0);
        if (bytes_received <= 0) {
            break;
        }

        buffer[bytes_received] = '\0';
        printf("Received message: %s\n", buffer);

        char command[10];
        sscanf(buffer, "%9s", command);

        if (strcmp(command, "bid") == 0) {
            char item[20];
            int price;
            sscanf(buffer, "bid %19s %d", item, &price);

            pthread_mutex_lock(&lock);
            for (int i = 0; i < sizeof(items) / sizeof(Item); i++) {
                if (strcmp(items[i].item, item) == 0) {
                    if (price > items[i].price) {
                        items[i].price = price;
```

```
                        items[i].bidder_fd = client_fd;
                        send(client_fd, "Bid accepted", 12, 0);
                    } else {
                        send(client_fd, "Bid rejected", 12, 0);
                    }
                    break;
                }
            }
            pthread_mutex_unlock(&lock);
        }


else if (strcmp(command, "buy") == 0) {
            char item[20];
            sscanf(buffer, "buy %19s", item);

            pthread_mutex_lock(&lock);
            for (int i = 0; i < sizeof(items) / sizeof(Item); i++) {
                if (strcmp(items[i].item, item) == 0) {
                    if (items[i].bidder_fd == client_fd) {
                        send(client_fd, "Item bought", 11, 0);
                        items[i].price = 0;
                        items[i].bidder_fd = -1;
                    } else {
                        send(client_fd, "You are not the highest bidder", 24, 0);
                    }
                    break;
                }
            }
            pthread_mutex_unlock(&lock);
        } else if (strcmp(command, "list") == 0) {
            pthread_mutex_lock(&lock);
            char items_list[100];
            strcpy(items_list, "");
            for (int i = 0; i < sizeof(items) / sizeof(Item); i++) {
                char item_str[30];
                sprintf(item_str, "%s: %d\n", items[i].item, items[i].price);
                strcat(items_list, item_str);
            }
            send(client_fd, items_list, strlen(items_list), 0);
            pthread_mutex_unlock(&lock);
        } else if (strcmp(command, "quit") == 0) {
            break;
        }
    }

    close(client_fd);
    return NULL;
}

int main() {
    int server_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (server_fd < 0) {
        perror("socket failed");
        return 1;
```

```
    }

    struct sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr);

    if (bind(server_fd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
{
        perror("bind failed");
        return 1;
    }

    if (listen(server_fd, 5) < 0) {
        perror("listen failed");
        return 1;
    }

    printf("Auctioneer started. Waiting for bidders...\n");



    while (1) {
        struct sockaddr_in client_addr;
        socklen_t client_len = sizeof(client_addr);
        int client_fd = accept(server_fd, (struct sockaddr*)&client_addr,
&client_len);
        if (client_fd < 0) {
            perror("accept failed");
            continue;
        }

        printf("Connected to client\n");

        int* client_fd_ptr = malloc(sizeof(int));
        *client_fd_ptr = client_fd;
        pthread_t thread;
        pthread_create(&thread, NULL, handle_client, client_fd_ptr);
    }

    return 0;
}
```

Client Code:

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

#define MAX_MESSAGE 1024

int main() {
    int client_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (client_socket < 0) {
        perror("socket");
        return 1;
    }

    struct sockaddr_in server_address;
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(12345);
    inet_pton(AF_INET, "127.0.0.1", &server_address.sin_addr);

    if (connect(client_socket, (struct sockaddr*)&server_address,
sizeof(server_address)) < 0) {
        perror("connect");
        return 1;
    }
    printf("Connected to auctioneer\n");
    char message[MAX_MESSAGE];
    while (1) {
        printf("Enter command (bid <item> <price>, buy <item>, list, quit): ");
        fgets(message, MAX_MESSAGE, stdin);
        message[strlen(message) - 1] = '\0'; // Remove newline character

        if (send(client_socket, message, strlen(message), 0) < 0) {
            perror("send");
            return 1;
        }
        char response[MAX_MESSAGE];
        int bytes_received = recv(client_socket, response, MAX_MESSAGE, 0);
        if (bytes_received < 0) {
            perror("recv");
            return 1;
        }
        response[bytes_received] = '\0';
        printf("%s\n", response);
        if (strcmp(message, "quit") == 0) {
            break;
        }
    }

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

OUTPUT :

Server Terminal :

```
Auctioneer started. Waiting for bidders...
Connected to client
Received message: list
Received message: bid item1 120
Received message: list
Received message: buy item1
Received message: list
Received message: bid item2 250
Received message: bid item2 120
Received message: list
Received message: buy item2
Received message: quit
```

Client Terminal :

```
Connected to auctioneer
Enter command (bid <item> <price>, buy <item>, list, quit): list
item1: 100
item2: 200

Enter command (bid <item> <price>, buy <item>, list, quit): bid item1 120
Bid accepted
Enter command (bid <item> <price>, buy <item>, list, quit): list
item1: 120
item2: 200

Enter command (bid <item> <price>, buy <item>, list, quit): buy item1
Item bought
Enter command (bid <item> <price>, buy <item>, list, quit): list
item1: 0
item2: 200

Enter command (bid <item> <price>, buy <item>, list, quit): bid item2 250
Bid accepted
Enter command (bid <item> <price>, buy <item>, list, quit): bid item2 120
Bid rejected
Enter command (bid <item> <price>, buy <item>, list, quit): list
item1: 0
item2: 250

Enter command (bid <item> <price>, buy <item>, list, quit): buy item2
Item bought
Enter command (bid <item> <price>, buy <item>, list, quit): quit
```

ROUND TRIP TIME

Server Code :

```
#include <bits/stdc++.h>
#include <chrono>
#include <cstring>
#include <arpa/inet.h>
#include <unistd.h>
using namespace std;
#define PORT 8081
#define BUFFER_SIZE 1024

int main() {

    struct sockaddr_in serverAddress;
    int len = sizeof(serverAddress);
    char buffer[BUFFER_SIZE] = {0};

    // SOCKET
       int serverFD = socket(AF_INET, SOCK_STREAM, 0);
       if(serverFD<0){
              cout<<"Server Socket Creation failed"<<endl;
              exit(EXIT_FAILURE);
       }

    serverAddress.sin_family = AF_INET;
    serverAddress.sin_addr.s_addr = INADDR_ANY;
    serverAddress.sin_port = htons(PORT);

    // BIND
       int bindX = bind(serverFD,(struct sockaddr*)&serverAddress,
sizeof(serverAddress));
       if(bindX<0){
              cout<<"Bind Failed"<<endl;
              exit(EXIT_FAILURE);
       }
    // LISTEN
       int listenX = listen(serverFD, 3);
       if(listenX<0){
              cout<<"Listen Failed"<<endl;
              exit(EXIT_FAILURE);
       }

    // ACCEPT
       int newClientSocket = accept(serverFD, (struct sockaddr*)&serverAddress,
(socklen_t*)&len);
       if(newClientSocket<0){
              cout<<"Accept Failed"<<endl;
              exit(EXIT_FAILURE);
       }

    while (true) {
        memset(buffer, 0, BUFFER_SIZE);
        int valread = read(newClientSocket, buffer, BUFFER_SIZE);

            if(valread>0){
                   cout << "Received: " << buffer << endl;
```

```
            send(newClientSocket, buffer, strlen(buffer), 0);
            cout << "Echoed: " << buffer << endl;
             }
    }

    close(newClientSocket);
    close(serverFD);
    return 0;
}
```

Client Code :

```
#include <bits/stdc++.h>
#include <chrono>
#include <cstring>
#include <arpa/inet.h>
#include <unistd.h>
#include <chrono>
using namespace std;
using namespace std::chrono;

#define PORT 8081

int main() {
    struct sockaddr_in serverAddress;
    char buffer[1024] = {0};

    //SOCKET
    int clientFD = socket(AF_INET, SOCK_STREAM, 0);
    if(clientFD<0){
        cout<<"Socket failed"<<endl;
        exit(EXIT_FAILURE);
    }

    //FAMILY, PORT, ADDRESS of server
    serverAddress.sin_family=AF_INET;
    serverAddress.sin_port = htons(PORT);
    int convertX = inet_pton(AF_INET, "127.0.0.1", &serverAddress.sin_addr);
    if(convertX<0){
        cout<<"Conversion of address failed"<<endl;
        exit(EXIT_FAILURE);
    }

    //CONNECT
    int connectX = connect(clientFD,(struct sockaddr*)&serverAddress,
sizeof(serverAddress));
    if(connectX<0){
        cout<<"Connecting clientFD to server failed"<<endl;
        close(clientFD);
        exit(EXIT_FAILURE);
    }

    // Record start time
    auto start = high_resolution_clock::now();

    // Print current time before sending message
    auto now = system_clock::now();
```

```
    auto now_c = system_clock::to_time_t(now);
    cout << "Current Time Before Sending Message: " << ctime(&now_c);

    // Send message
       const char* message = "Hello from client";
    send(clientFD, message, strlen(message), 0);
    cout << "Message sent: " << message << endl;

    // Wait for echo
    read(clientFD, buffer, 1024);

    // Print received message after getting response from server
       cout << "Echo received at: ";
    now = chrono::system_clock::now();
    now_c = chrono::system_clock::to_time_t(now);
    cout << ctime(&now_c);
       cout << "Message: " << buffer << endl;

    // Record end time
    auto end = high_resolution_clock::now();
    duration<double> elapsed = end - start;

    cout << "Round-Trip Time: " << elapsed.count() << " seconds" << endl;

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

OUTPUT :

Server Terminal :

```
Received: Hello from client
Echoed: Hello from client
```

Client Terminal :

```
Current Time Before Sending Message: Sun Aug 18 15:26:54 2024
Message sent: Hello from client
Echo received at: Sun Aug 18 15:26:54 2024
Message: Hello from client
Round-Trip Time: 0.000768521 seconds
```

DNS PROTOCOL

Server Code:

```
#include <bits/stdc++.h>
#include <cstring>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#include <netdb.h>
#define PORT 4553
using namespace std;


string findIP(string& domain) {
    struct addrinfo hints = {}, *res;
    hints.ai_family = AF_INET; // IPv4
    if (getaddrinfo(domain.c_str(), nullptr, &hints, &res) != 0) {
        return "";
    }

    char ipString[INET_ADDRSTRLEN];
    inet_ntop(AF_INET, &((struct sockaddr_in*)res->ai_addr)->sin_addr, ipString,
sizeof(ipString));

    return ipString;
}

int main() {
    struct sockaddr_in serverAddress;
    struct sockaddr_in clientAddress;
    socklen_t len = sizeof(clientAddress);
    char buffer[1024];

    // Create socket
    int sockFD = socket(AF_INET, SOCK_DGRAM, 0);
    if(sockFD<0) {
        cout<<"Socket Creation Failed"<<endl; exit(EXIT_FAILURE);
    }

    // Set server Details
    memset(&serverAddress, 0, sizeof(serverAddress));
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_addr.s_addr = INADDR_ANY;
    serverAddress.sin_port = htons(PORT);
```

```
    int bindX = bind(sockFD, (struct sockaddr*)&serverAddress,
sizeof(serverAddress));
    if(bindX<0){
        cout<<"Bind Failed"<<endl; close(sockFD); exit(EXIT_FAILURE);
    }
    while (true) {
        memset(buffer, 0, sizeof(buffer));

        int n = recvfrom(sockFD, buffer, sizeof(buffer), 0, (struct
sockaddr*)&clientAddress, &len);
        buffer[n] = '\0';

        string domainName(buffer);
        string ipAddress = findIP(domainName);

        sendto(sockFD, ipAddress.c_str(), ipAddress.length(), 0, (struct
sockaddr*)&clientAddress, len);
       cout<<”IP address sent Successfully…”;
    }

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

Client Code:

```
#include <bits/stdc++.h>
#include <cstring>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <unistd.h>
#define PORT 4553
using namespace std;

int main() {

    struct sockaddr_in serverAddress;
    socklen_t len = sizeof(serverAddress);
    char buffer[1024];

    // Create socket
    int clientFD = socket(AF_INET, SOCK_DGRAM, 0);
    if(clientFD<0) {
        cout<<"Socket Creation Failed"<<endl; exit(EXIT_FAILURE);
    }

    // Set server details

    memset(&serverAddress, 0, sizeof(serverAddress));
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_port = htons(PORT);
    serverAddress.sin_addr.s_addr = INADDR_ANY;
```

```
    string domainName;
    cout << "Enter domain name: ";
    cin >> domainName;

    // Send domain name to server
    sendto(clientFD, domainName.c_str(), domainName.length(), 0, (const struct
sockaddr*)&serverAddress, sizeof(serverAddress));

    // Receive IP address from server

    int n = recvfrom(clientFD, buffer, sizeof(buffer), 0, (struct
sockaddr*)&serverAddress, &len);

    cout << "IP Address: " << buffer << std::endl;

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

OUTPUT :

Server Terminal:

```
IP address sent Successfully...
```

Client Terminal:

```
Enter domain name: google.com
IP Address: 142.250.194.110
```