COMPUTER NETWORKS

LAB ASSIGNMENT-07

REDDIPALLI SAI CHARISH

CS22B1095

PURE ALOHA:

SERVER CODE:

aloha_server.c

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

#define PORT 8080
#define MAX_USERS 5

int user_count = 0;
pthread_mutex_t medium_lock;

void *handle_user(void *socket_desc) {
    int new_socket = *(int*)socket_desc;
    char message[1024];
    int read_size;
    int packet_number;

    while ((read_size = recv(new_socket, message, 1024, 0)) > 0) {
        pthread_mutex_lock(&medium_lock);
```

```
            printf("Received packet: %s", message);
            sscanf(message, "Packet %d", &packet_number);
            int collision = rand() % 2;
            if (collision == 0) {
                printf("\nCollision detected on Packet %d! Notifying user to retry...\n", packet_number);
                char collision_message[50];
                sprintf(collision_message, "Collision occurred on Packet %d. Please resend.\n", packet_number);
                send(new_socket, collision_message, strlen(collision_message), 0);
            } else {
                printf("\nPacket %d received successfully.\n", packet_number);
                char success_message[50];
                sprintf(success_message, "Packet %d received successfully.\n", packet_number);
                send(new_socket, success_message, strlen(success_message), 0);
            }
            pthread_mutex_unlock(&medium_lock);
            memset(message, 0, 1024);
        }

        if (read_size == 0) {
            printf("User disconnected.\n");
            fflush(stdout);
        } else if (read_size == -1) {
            perror("recv failed");
        }
        free(socket_desc);
        pthread_mutex_lock(&medium_lock);
        user_count--;
        pthread_mutex_unlock(&medium_lock);

        return 0;
    }

    int main(int argc, char *argv[]) {
        int server_socket, client_socket, c, *new_sock;
        struct sockaddr_in server, client;

        srand(time(NULL));
        server_socket = socket(AF_INET, SOCK_STREAM, 0);
```

```
if (server_socket == -1) {
    printf("Could not create socket.\n");
}
puts("Socket created.");
server.sin_family = AF_INET;
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = htons(PORT);
if (bind(server_socket, (struct sockaddr *)&server, sizeof(server)) < 0) {
    perror("Bind failed.");
    return 1;
}
puts("Bind done.");
listen(server_socket, 3);
puts("Waiting for incoming connections...");
c = sizeof(struct sockaddr_in);
pthread_mutex_init(&medium_lock, NULL);

while ((client_socket = accept(server_socket, (struct sockaddr *)&client, (socklen_t*)&c))) {
    puts("Connection accepted.");
    pthread_mutex_lock(&medium_lock);
    if (user_count >= MAX_USERS) {
        printf("Max user limit reached, rejecting connection.\n");
        close(client_socket);
        pthread_mutex_unlock(&medium_lock);
        continue;
    }
    user_count++;
    pthread_mutex_unlock(&medium_lock);
    pthread_t user_thread;
    new_sock = malloc(1);
    *new_sock = client_socket;

    if (pthread_create(&user_thread, NULL, handle_user, (void*)new_sock) < 0) {
        perror("Could not create thread.");
        return 1;
    }
    pthread_detach(user_thread);
}
```

```
    if (client_socket < 0) {
        perror("Accept failed.");
        return 1;
    }

    pthread_mutex_destroy(&medium_lock);
    return 0;
}
```

CLIENT CODE:

aloha_client.c

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

#define PORT 8080
#define MAX_MESSAGE_LEN 900
#define COLLISION_PROBABILITY 0.3

void *send_data(void *socket_desc) {
    int sock = *(int*)socket_desc;
    char message[MAX_MESSAGE_LEN];
    char server_reply[1024];
    int packet_number = 0;

    while (1) {
        printf("Enter a message to send (or type 'exit' to quit): ");
        fgets(message, MAX_MESSAGE_LEN, stdin);
        if (strncmp(message, "exit", 4) == 0) {
            printf("Disconnecting...\n");
            break;
```

```
        }
        message[strcspn(message, "\n")] = '\0';
        packet_number++;
        char packet_message[1024];
        snprintf(packet_message, sizeof(packet_message), "Packet %d: %s", packet_number, message);
        double random_value = (double)rand() / RAND_MAX;
        if (random_value < COLLISION_PROBABILITY) {
            printf("Collision detected for Packet %d! Waiting to retry...\n", packet_number);
            int backoff_time = rand() % 5 + 1;
            printf("Retrying after %d seconds...\n", backoff_time);
            sleep(backoff_time);
            printf("Resending Packet %d: %s\n", packet_number, message);
        }
        if (send(sock, packet_message, strlen(packet_message), 0) < 0) {
            puts("Send failed.");
            return 0;
        }
        if (recv(sock, server_reply, 1024, 0) < 0) {
            puts("Recv failed.");
            return 0;
        }
        printf("Server reply: %s\n", server_reply);

        memset(server_reply, 0, 1024);
    }

    return 0;
}

int main(int argc, char *argv[]) {
    int sock;
    struct sockaddr_in server;
    pthread_t send_thread;

    srand(time(NULL));



    sock = socket(AF_INET, SOCK_STREAM, 0);
```

```
    if (sock == -1) {
        printf("Could not create socket.\n");
    }
    puts("Socket created.");

    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_family = AF_INET;
    server.sin_port = htons(PORT);
    if (connect(sock, (struct sockaddr *)&server, sizeof(server)) < 0) {
        perror("Connection failed.");
        return 1;
    }
    puts("Connected to server.");
    if (pthread_create(&send_thread, NULL, send_data, (void*)&sock) < 0) {
        perror("Could not create thread.");
        return 1;
    }

    pthread_join(send_thread, NULL);
    close(sock);

    return 0;
}
```

```
charish@LAPTOP-GFCS9LJ9:~/cn/LAB 07$ gcc aloha_server.c
charish@LAPTOP-GFCS9LJ9:~/cn/LAB 07$ ./a.out
Socket created.
Bind done.
Waiting for incoming connections...
Connection accepted.
Received packet: Packet 1: hi
Collision detected on Packet 1! Notifying user to retry...
Received packet: Packet 2: hello
Collision detected on Packet 2! Notifying user to retry...
Received packet: Packet 3: hi
Packet 3 received successfully.
Received packet: Packet 4: hello
Collision detected on Packet 4! Notifying user to retry...
User disconnected.
^C
charish@LAPTOP-GFCS9LJ9:~/cn/LAB 07$
```

```
Server: Data transmitted successfully: hello
Enter message to send (type 'exit' to quit): exit
Disconnecting...
charish@LAPTOP-GFCS9LJ9:~/cn/LAB 07$ gcc aloha_client.c
charish@LAPTOP-GFCS9LJ9:~/cn/LAB 07$ ./a.out
Socket created.
Connected to server.
Enter a message to send (or type 'exit' to quit): hi
Collision detected for Packet 1! Waiting to retry...
Retrying after 1 seconds...
Resending Packet 1: hi
Server reply: Collision occurred on Packet 1. Please resend.

Enter a message to send (or type 'exit' to quit): hello
Server reply: Collision occurred on Packet 2. Please resend.

Enter a message to send (or type 'exit' to quit): hi
Server reply: Packet 3 received successfully.

Enter a message to send (or type 'exit' to quit): hello
Server reply: Collision occurred on Packet 4. Please resend.

Enter a message to send (or type 'exit' to quit): exit
Disconnecting...
charish@LAPTOP-GFCS9LJ9:~/cn/LAB 07$
```

CSMA/CD:

Server code:

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

#define PORT 8080
#define MAX_CLIENTS 5
#define MEDIUM_BUSY 1
#define MEDIUM_FREE 0

int medium_status = MEDIUM_FREE;
pthread_mutex_t medium_mutex;
int connected_clients = 0;

void *handle_client(void *socket_desc) {
    int client_sock = *(int*)socket_desc;
    int collision_detected = 0;

    char client_message[1024];
    char response[1024];

    while (1) {
        memset(client_message, 0, 1024);
        if (recv(client_sock, client_message, sizeof(client_message), 0) > 0) {
            pthread_mutex_lock(&medium_mutex);
```

```
        if (medium_status == MEDIUM_FREE) {
            printf("Medium is free. Client can send data.\n");
            medium_status = MEDIUM_BUSY;
            printf("Client is transmitting: %s", client_message);
            sleep(1);
            if (collision_detected) {
                snprintf(response, sizeof(response), "Collision detected! Stop transmission.\n");
                collision_detected = 0;
            } else {

                snprintf(response, sizeof(response), "Data transmitted successfully: %.900s",
client_message);

            }
            medium_status = MEDIUM_FREE;
        } else {
            snprintf(response, sizeof(response), "Medium busy, wait for retry.\n");
            collision_detected = 1;
        }

        pthread_mutex_unlock(&medium_mutex);
        send(client_sock, response, strlen(response), 0);
      }
   }
   close(client_sock);
   free(socket_desc);
   connected_clients--;
   return 0;
}
```

```
int main() {
    int server_sock, client_sock, c;
    struct sockaddr_in server, client;
    pthread_t client_thread;
    pthread_mutex_init(&medium_mutex, NULL);
    server_sock = socket(AF_INET, SOCK_STREAM, 0);
    if (server_sock == -1) {
        printf("Could not create socket.\n");
        return 1;
    }
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(PORT);
    if (bind(server_sock, (struct sockaddr *)&server, sizeof(server)) < 0) {
        perror("Bind failed.");
        return 1;
    }
    listen(server_sock, MAX_CLIENTS);
    printf("Server is listening on port %d...\n", PORT);
    c = sizeof(struct sockaddr_in);
    while ((client_sock = accept(server_sock, (struct sockaddr *)&client, (socklen_t*)&c)) &&
connected_clients < MAX_CLIENTS) {
        printf("Client connected.\n");
        pthread_t client_thread;
        int *new_sock = malloc(1);
        *new_sock = client_sock;

        if (pthread_create(&client_thread, NULL, handle_client, (void*)new_sock) < 0) {
            perror("Could not create thread.");
            return 1;
        }
```

```
        connected_clients++;
    }

    if (client_sock < 0) {
        perror("Accept failed.");
        return 1;
    }

    pthread_mutex_destroy(&medium_mutex);
    close(server_sock);

    return 0;
}
```

Client Code:

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

#define PORT 8080
#define MAX_MESSAGE_LEN 1024
#define MEDIUM_CHECK_PROBABILITY 0.7

int main() {
    int sock;
    struct sockaddr_in server;
```

```
char message[MAX_MESSAGE_LEN], server_reply[MAX_MESSAGE_LEN];
srand(time(0));
sock = socket(AF_INET, SOCK_STREAM, 0);
if (sock == -1) {
    printf("Could not create socket.\n");
    return 1;
}
server.sin_addr.s_addr = inet_addr("127.0.0.1");
server.sin_family = AF_INET;
server.sin_port = htons(PORT);
if (connect(sock, (struct sockaddr *)&server, sizeof(server)) < 0) {
    perror("Connection failed.");
    return 1;
}
printf("Connected to server.\n");

while (1) {
    printf("Enter message to send (type 'exit' to quit): ");
    fgets(message, MAX_MESSAGE_LEN, stdin);

    if (strncmp(message, "exit", 4) == 0) {
        printf("Disconnecting...\n");
        break;
    }
    float random_value = (float)rand() / RAND_MAX;
    if (random_value < MEDIUM_CHECK_PROBABILITY) {
        printf("Medium perceived as free (probability check passed).\n");
        if (send(sock, message, strlen(message), 0) < 0) {
            puts("Send failed.");
            return 1;
        }
```

```
            if (recv(sock, server_reply, MAX_MESSAGE_LEN, 0) < 0) {
                puts("Receive failed.");
                break;
            }
            printf("Server: %s", server_reply);
            if (strstr(server_reply, "Collision detected")) {
                int backoff_time = rand() % 5 + 1;
                printf("Backing off for %d seconds...\n", backoff_time);
                sleep(backoff_time);
            }
        } else {
            printf("Medium perceived as busy (probability check failed). Retrying...\n");
            int retry_time = rand() % 3 + 1;
            sleep(retry_time);
        }
    }

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