LABORATORIO DI RETI E SISTEMI DISTRIBUITI

HANDS-ON 1

Comunicazione UDP

Antonio Mastrolembo Ventura

543644

February 28, 2025

1 Esercizio1

Sono stati realizzati due file: *server_udp.c* e *client_udp.c*. Abbiamo un server che, in questo caso, non stabilisce connessioni persistenti ma resta in ascolto di pacchetti dati inviati dal client e risponde direttamente senza creare un canale dedicato per ogni comunicazione. A differenza del protocollo TCP, in cui ogni comunicazione avviene su un canale dedicato tra client e server, con UDP i messaggi possono essere inviati e ricevuti indipendentemente, senza il bisogno di una fase di handshake.

1.1 Codice del Server UDP

1.1.1 Funzione Principale

```
int main(){
    // Un solo file descriptor a differenza di TCP
    int server_fd;

    // Due struct per memorizzare gli indirizzi del server e del
   client
    struct sockaddr_in server_addr, client_addr;
    int addrlen = sizeof(client_addr);

    // Buffer per la ricezione e l'invio dei messaggi
    char buffer[1024] = {0};
    char *string = "[SERVER] Hello, message received!";
```

In prima battuta notiamo che è presente **un solo file descriptor**, questo perché, a differenza del procotollo TCP, in UDP non abbiamo un socket separato per ogni client. Successivamente abbiamo due struct *sockaddr_in*: una per memorizzare l'indirizzo del server, necessario per il binding del socket e una per memorizzare l'indirizzo del client, che è necessario ogni volta che il server riceve un pacchetto dal client.

1.1.2 Creazione del Socket

```
    // Creazione del socket
    if ((server_fd = socket(AF_INET, SOCK_DGRAM, 0)) == 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }
```

Il *type* è impostato come *SOCK_DGRAM* in quanto stiamo lavorando con UDP.

1.1.3 Configurazione dell'indirizzo del server e binding

```
    // Configurazione dell'indirizzo del server
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);

    // Binding del socket
    if (bind(server_fd, (struct sockaddr *)&server_addr, sizeof(
server_addr)) < 0) {
        perror("Bind failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }
```

La porta utilizzata è la 7600.

1.1.4 Ricezione del messaggio del client

```
    // Ricezione del messaggio dal client
        recvfrom(server_fd, buffer, 1024, 0, (struct sockaddr *)&
client_addr, (socklen_t*)&addrlen);
        printf("%s\n", buffer);
```

Legge il messaggio inviato dal client e lo stampa sulla console.

client_addr specifica l'indirizzo del mittente.

1.1.5 Invio della risposta al client

```
    // Invio di una risposta al client
        sendto(server_fd, string, strlen(string), 0, (struct sockaddr
    *)&client_addr, addrlen);
        printf("Message sent\n");
```

Invia un messaggio di risposta al client.

client_addr è l'indirizzo del destinatario.

1.1.6 Chiusura del Socket

```
    // Chiusura del socket
    close(server_fd);
    return 0;
}
```

Chiude il socket e il programma termina.

1.2 Codice del Client UDP

1.2.1 Funzione Principale

```
int main() {
    int sock = 0;
    struct sockaddr_in server_addr;
    char *string = "[CLIENT] Hello from client!";
    char buffer[1024] = {0};
```

1.2.2 Creazione del Socket

```
    // Creazione del socket
    if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        printf("\n[CLIENT] Socket creation error\n");
        return -1;
    }
```

1.2.3 Configurazione dell'indirizzo del server e conversione IP

```
    // Configurazione dell'indirizzo del server
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);

    // Conversione dell'indirizzo IP
    if (inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr) <= 0)
    {
        printf("\n[CLIENT] Invalid address/Address not supported\n");
        return -1;
    }
```

inet_pton converte l'indirizzo IP in binario.

1.2.4 Invio di un messaggio al server

```
    // Invio di un messaggio al server
    sendto(sock, string, strlen(string), 0, (struct sockaddr *)&
    server_addr, sizeof(server_addr));
    printf("[CLIENT] Message sent to server\n");
```

server_addr è l'indirizzo del destinatario.

1.2.5 Ricezione del messaggio dal server

```
    // Ricezione di un messaggio dal server
    recvfrom(sock, buffer, 1024, 0, NULL, NULL);
    printf("[CLIENT] Message received from server: %s\n", buffer);
```

1.2.6 Chiusura del Socket

```
    // Chiusura del socket
    close(sock);
    return 0;
}
```

Chiude il socket e il programma termina.

1.3 Output ottenuto

Figure 1: Output del client

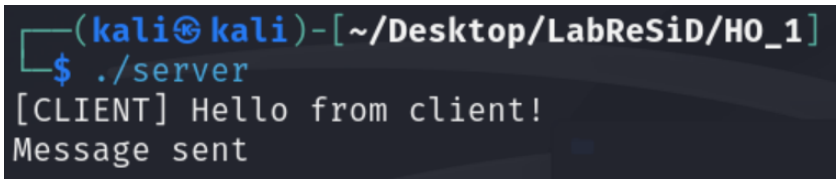

Figure 2: Output del server

2 Esercizio2

Per listare correttamente il file descriptor relativo all'esercizio precedente è possibile utilizzare il comando `"lsof"` accompagnato ad esempio dal flag `"-i :porta"`. In questo modo riesco a listare tutti i processi che stanno utilizzando una specifica porta di rete.

Figure 3: File descriptor del server socket

Osserviamo diversi campi:

- **COMMAND**: il nome del comando, in questo caso `server`;
- **PID**: L'ID del processo (*process id*) in questione;
- **USER**: L'utente che esegue il processo (colui che ha aperto il socket);

- **FD**: File descriptor associato al socket (u indica che è in modalità lettura e scrittura);
- **TYPE**: La connessione usa il protocollo IPv4;
- **NAME**: Indica l'indirizzo e la porta del socket. In questo caso un socket in ascolto sulla porta 7600.