Computer-Networks-UDP-Program-Tips

UDP-Client

Include Statements

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

UDP-Client: Basic Algorithm for remembering

1. Create the socket
2. Setup the server address struct
3. Loop
    1. Read input and Send message to server
    2. Receive message server and display message
    3. Logic to end the loop
4. Close the socket

UDP-Client: Steps in more detail

1. Create the socket

```
lfd = socket(AF_INET, SOCK_DGRAM, 0); // SOCK_DGRAM indicates that its UDP
```

- Here AF_INET is the address_family
- SOCK_DGRAM is the socket type (SOCK_DGRAM Indicates its UDP)
- Protocol is 0

2. Setup Server address struct

```
struct sockaddr_in server;
server.sin_family = AF_INET; // IPv4
server.sin_port = 2000; // Port number
server.sin_addr.s_addr = inet_addr("127.0.0.1"); // IP address
```

3. Loop

3.1 Read input and send message to server

```
n = sizeof server; // Get the size of the server address structure
gets(sBuf); // Get user input
sendto(lfd, sBuf, sizeof sBuf, 0, (struct sockaddr *)&server, n); // Send
data to the server
```

- sendto sends data to a server using the previously created UDP socket
- sendto has the following parameters
  - **`lfd`**: This is the file descriptor of the socke
  - Buffer
  - Length of Buffer
  - Flags
    - Set as 0
  - **`(struct sockaddr *)&server`**: This argument specifies the address of the server you want to send data to. It's a pointer to a `struct sockaddr` which is a generic socket address structure.
  - n
    - size of server struct

3.2 Receive message from server and display message

```
recvfrom(lfd, rBuf, sizeof rBuf, 0, (struct sockaddr*)&server, &n); //
Receive data from the server
printf("\nServer:%s", rBuf);
```

- Parameters used
  - First 5 parameters are similar

- `&n` : This is a pointer to an integer variable.
    - The `recvfrom` function will store the number of bytes actually received from the server in this memory location pointed to by `&n`. This allows you to know how much data you received successfully.

4. Close the socket

```
close(lfd);
```

UDP-Server: Basic Algorithm for remembering

1. Create the socket
2. Setup the server address struct
3. Bind the socket
4. Loop
    1. Receive data from client and print the message
    2. Read input and send message to client
    3. Logic to end the loop
5. Close server socket

UDP-Server: Steps in more detail

1. Create the socket

```
lfd = socket(AF_INET, SOCK_DGRAM, 0); // Create a socket (UDP)
```

2. Server address struct

```
server.sin_family = AF_INET; // IPv4
server.sin_port = 2001; // Port number
server.sin_addr.s_addr = inet_addr("127.0.0.1"); // IP address
```

3. Bind the socket

```
bind(lfd, (struct sockaddr *)&server, sizeof server); // Bind the socket to
the server address
```

Bind

- The `bind` function is used to associate a socket with a specific local IP address and port number.
- The parameters used are
  - sockfd
    - The file descriptor of the socket to be bound. In your example, this is `lfd`, which is the listening file descriptor.
  - **`addr` (const struct sockaddr *)**:
    - A pointer to a `struct sockaddr` that contains the address to bind to the socket.
  - **`addrlen` (socklen_t)**:
    - The size of the address structure. In your example, this is `sizeof(server)`, which is the size of the `struct sockaddr_in` structure.

4. Loop

4.1 Receive data from client and print the message

```
recvfrom(lfd, rBuf, sizeof rBuf, 0, (struct sockaddr *)&client, &n); //
Receive data from the client
printf("\nClient:%s", rBuf); // Print the received data
```

- The `recvfrom` function is a function for receiving data on a socket in network programming, specifically for connectionless protocols like UDP.
- The `recvfrom` function is used to receive data from a connected socket. It reads data from the socket into a buffer
- Parameters used
  - sockfd
    - The file descriptor of the connected socket from which to receive data. In this example, this is `confd`, which represents the new socket file descriptor created

    by the `accept` function for the specific client connection.
- Buffer
    - buffer where the received data will be stored.
- Size of buffer
- Flags
- **`(struct sockaddr *)&server`**: This argument specifies the address of the server you want to send data to. It's a pointer to a `struct sockaddr` which is a generic socket address structure.
- **`&n`**: This is a pointer to an integer variable.
    - The `recvfrom` function will store the number of bytes actually received from the server in this memory location pointed to by `&n`. This allows you to know how much data you received successfully.\

4.2 Read input and send message to client

```
gets(sBuf); // Get server input
sendto(lfd, sBuf, sizeof sBuf, 0, (struct sockaddr *)&client, n); // Send
data to the client
```

- sendto sends data to a server using the previously created UDP socket
- sendto has the following parameters
    - **`lfd`**: This is the file descriptor of the socke
    - Buffer
    - Length of Buffer
    - Flags
        - Set as 0
    - **`(struct sockaddr *)&server`**: This argument specifies the address of the server you want to send data to. It's a pointer to a `struct sockaddr` which is a generic socket address structure.
    - n
        - size of server struct

5. Close server socket

```c
close(lfd); // Close the socket
```