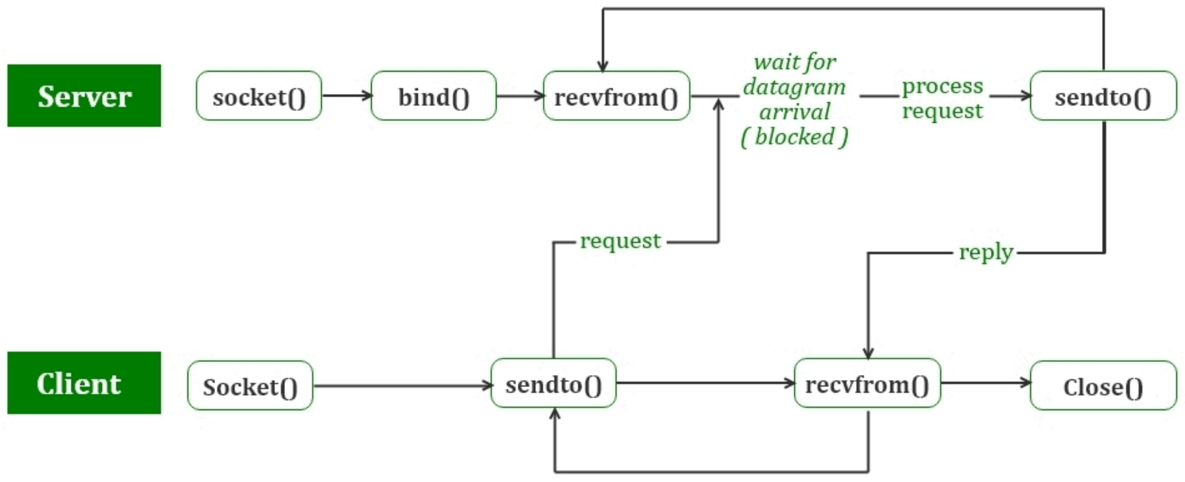
National University of Computer and Emerging Sciences (NUCES) Islamabad

Computer Networks Lab
Spring 2024
Week 09

Socket Programming (UDP Concurrent Servers)

There are two primary transport layer protocols to communicate between hosts: TCP and UDP. Creating TCP Server/Client was discussed in a previous post.

In UDP, the client does not form a connection with the server like in TCP and instead sends a datagram. Similarly, the server need not accept a connection and just waits for datagrams to arrive. Datagrams upon arrival contain the address of the sender which the server uses to send data to the correct client.



The entire process can be broken down into the following steps :

UDP Server :

1. Create a UDP socket.
2. Bind the socket to the server address.
3. Wait until the datagram packet arrives from the client.
4. Process the datagram packet and send a reply to the client.

5. Go back to Step 3.

UDP Client :

1. Create a UDP socket.
2. Send a message to the server.
3. Wait until a response from the server is received.
4. Process the reply and go back to step 2, if necessary.
5. Close socket descriptor and exit.

Sample Code:

Server Side

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


void handle_client(int server_socket);

int main() {
        int server_socket;
        struct sockaddr_in server_address, client_address;
        socklen_t client_address_len = sizeof(client_address);
        pid_t pid;

        // Create the server socket
        server_socket = socket(AF_INET, SOCK_DGRAM, 0);
        if (server_socket == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
        }

        // Configure server address
        server_address.sin_family = AF_INET;
        server_address.sin_addr.s_addr = INADDR_ANY;
```

```
	server_address.sin_port = htons(3001);

	// Bind the socket to the specified IP and port
	if (bind(server_socket, (struct sockaddr *) &server_address, sizeof(server_address)) ==
-1) {
	perror("Bind failed");
	exit(EXIT_FAILURE);
	}

	printf("Server started. Listening on port 3001 ...\n");

	handle_client(server_socket);

	// Close the server socket
	close(server_socket);

	return 0;
}

void handle_client(int server_socket) {
	char buf[200];
	int num1, num2, result;
	struct sockaddr_in client_address;
	socklen_t client_address_len = sizeof(client_address);

	while (1) {
	// Receive data from the client
	recvfrom(server_socket, buf, sizeof(buf), 0, (struct sockaddr *) &client_address,
&client_address_len);

	// Extract the numbers from the received data
	sscanf(buf, "%d %d", &num1, &num2);

	// Perform the operation
	result = num1 + num2;

	// Send the result back to the client
	sendto(server_socket, &result, sizeof(result), 0, (struct sockaddr *) &client_address,
client_address_len);
	}
}
```

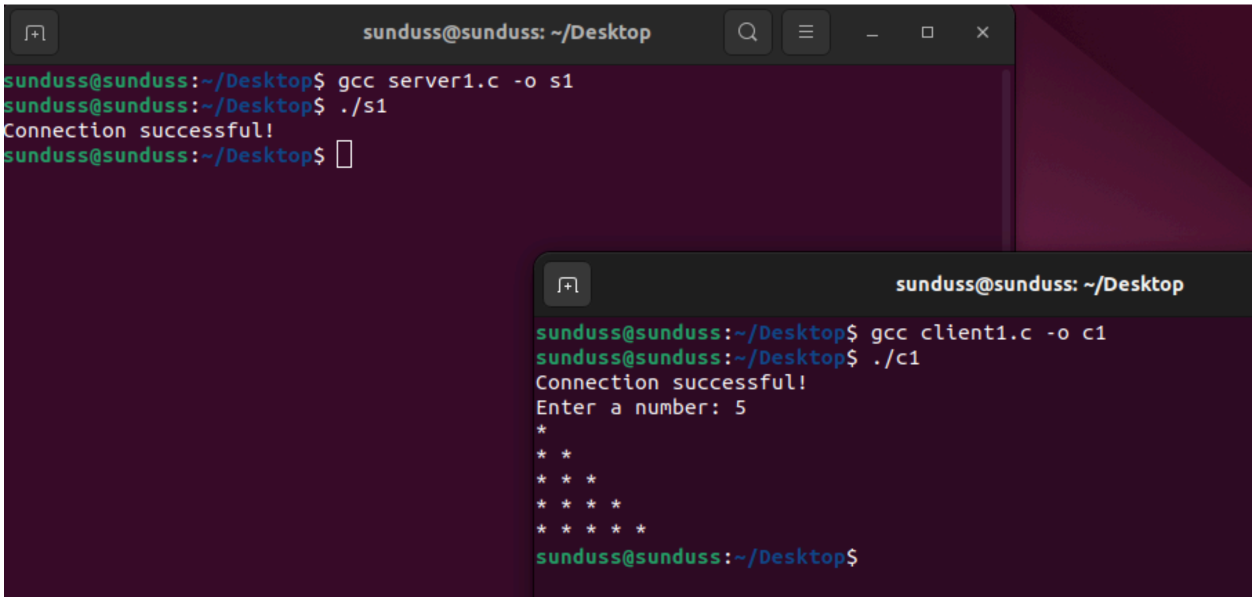
Client Side

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

#define SERVER_IP "127.0.0.1" // Change this to the IP address of the server

int main() {
        char request[256];
        int num1, num2, result;
        char buf[200];

        // create the socket
        int sock;
        sock = socket(AF_INET, SOCK_DGRAM, 0);

        // setup an address
        struct sockaddr_in server_address;
        server_address.sin_family = AF_INET;
        server_address.sin_addr.s_addr = inet_addr("127.0.0.1");
        server_address.sin_port = htons(3001);

        // Send the first message (numbers) to the server
        printf("Enter a number: ");
        scanf("%d", &num1);
        printf("Enter another number: ");
        scanf("%d", &num2);
        sprintf(request, "%d %d", num1, num2);
        sendto(sock, request, strlen(request), 0, (struct sockaddr *) &server_address,
sizeof(server_address));

        // Receive the result from the server
        recvfrom(sock, &result, sizeof(result), 0, NULL, NULL);
        printf("\nServer result: %d\n", result);

        // close the socket
        close(sock);

        return 0;
}
```

How to Run the code?

Open two separate terminals, each accessing the folder you have your code in. Compile both the files using the command : ***gcc filename.c -o exefilenameTobeassigned***

Once compiled without errors, access the server exe file, and then the client exe file. For reference see the image attached below.



sunduss@sunduss: ~/Desktop

```
sunduss@sunduss:~/Desktop$ gcc server1.c -o s1
sunduss@sunduss:~/Desktop$ ./s1
Connection successful!
sunduss@sunduss:~/Desktop$ 
```

sunduss@sunduss: ~/Desktop

```
sunduss@sunduss:~/Desktop$ gcc client1.c -o c1
sunduss@sunduss:~/Desktop$ ./c1
Connection successful!
Enter a number: 5
*
* *
* * *
* * * *
* * * * *
sunduss@sunduss:~/Desktop$
```

<u>Practice tasks</u>

<u>Task 1:</u>

Design a client-server model for one-way communication. The client should send messages and the server should be able to receive messages. The client should send a number and the server should respond whether the number is even or odd.

For example:

Client sent 2

Server should display 2 is an even number

Client sent 7

Server should display 7 is an odd number

Task 2:

Write a pair of UDP client/server programs showing the two-way communication between client and server. Each message should display the number of bytes of the message received. The communication should end when one of them enters the string “exit”.

Submission Guidelines:

- Rename the code as rollNumber_server1.c for task 1, and the code for the client module as rollNumber_client1.c for task 1.
- Submit the screenshot of the terminal with message transfer along with the source code for both client and server of both of the tasks.