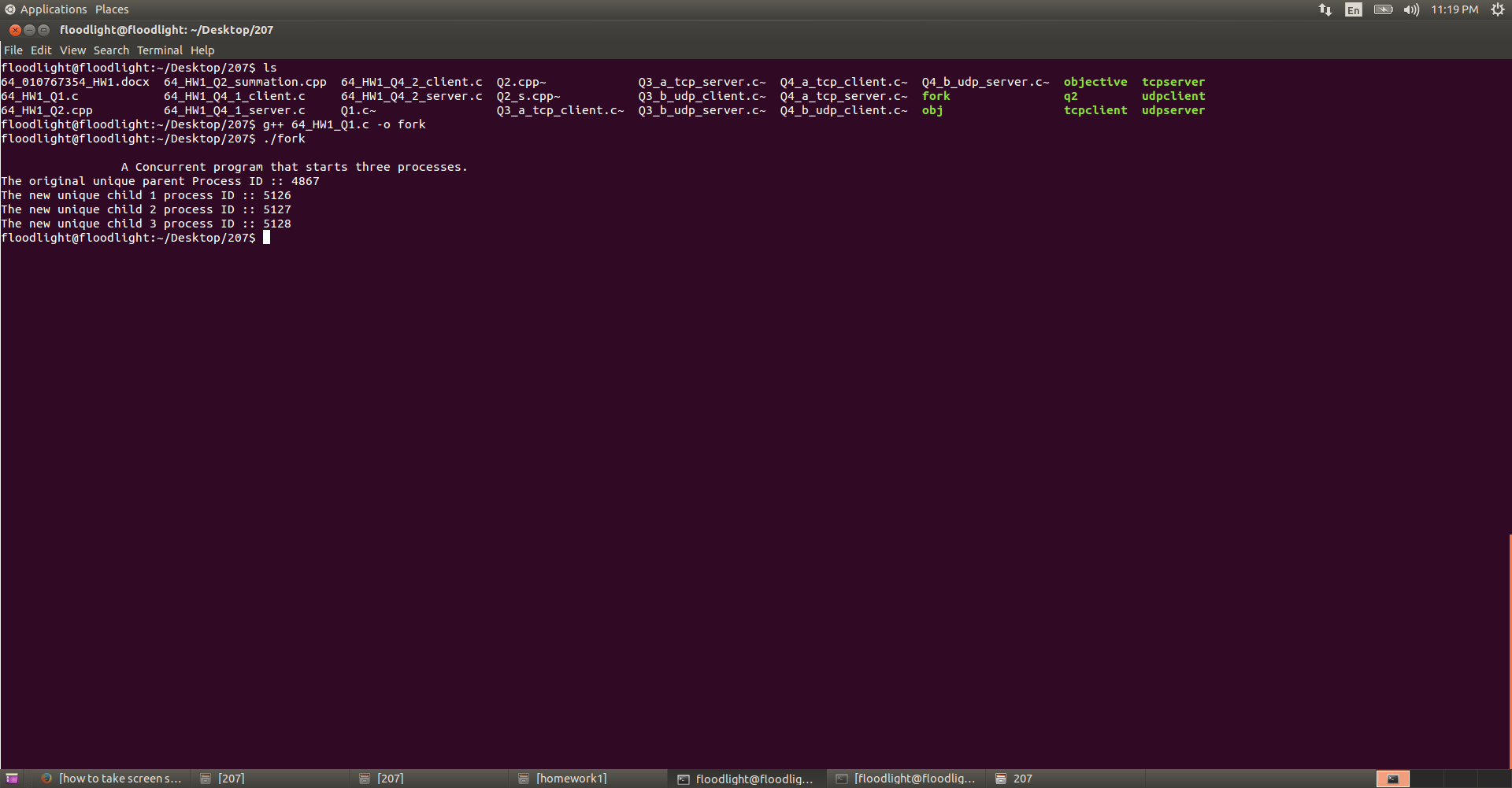
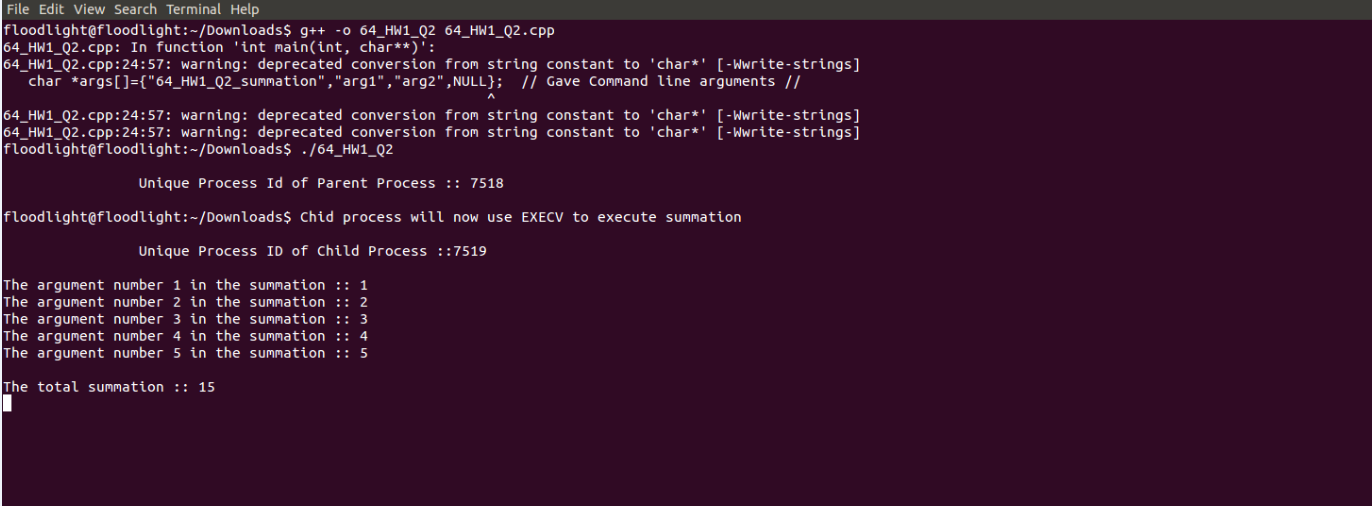
|  |
| --- |
| CMPE 207-01 HW#1 |
| Name: Aakash Chirag Shah (SJSU ID: 010767354) |
| Computer Engineering Department, |
| San Jose State University  Network Programming & Application |

**Q1. Write a concurrent program using fork() that starts three processes. Arrange for each process to print a few lines of output and then halt.**

****

**Q2. Write a program that uses “execve” to change the code a process executes. Write and compile the summation procedure as a separate program and uses execve() to call it in the child process.**

****

**Q3. Explain the following questions.**

1. **What is Network Byte Order?**

**Answer:**

* TCP/IP specifies a standard representation for binary integers used in protocol headers. The representation, known as **network byte order**, represents integers with the most significant byte first.
* Although the protocol software hides most values used in headers from application programs, a programmer must be aware of the standard because some socket routines require arguments to be stored in network byte order. For example, the protocol port field of a sockaddr\_in structure uses network byte order.
* The socket routines include several functions that convert between network byte order and the local host’s byte order.
* Programs should always call the conversion routines even if the local machine’s byte order is the same as the network byte order because doing so makes the source code portable too an arbitrary architecture.
* The conversion routines are divided into short and long sets to operate on 16-bit and 32-bit integers.
* Functions htons (Host to Network Short) and ntohs (Network to host short) convert a short integer from the host’s native byte order to the network byte order, and vice versa.
* Similarly, htonl and ntohl convert long integers from the host’s native byte order to network byte order and vice versa.

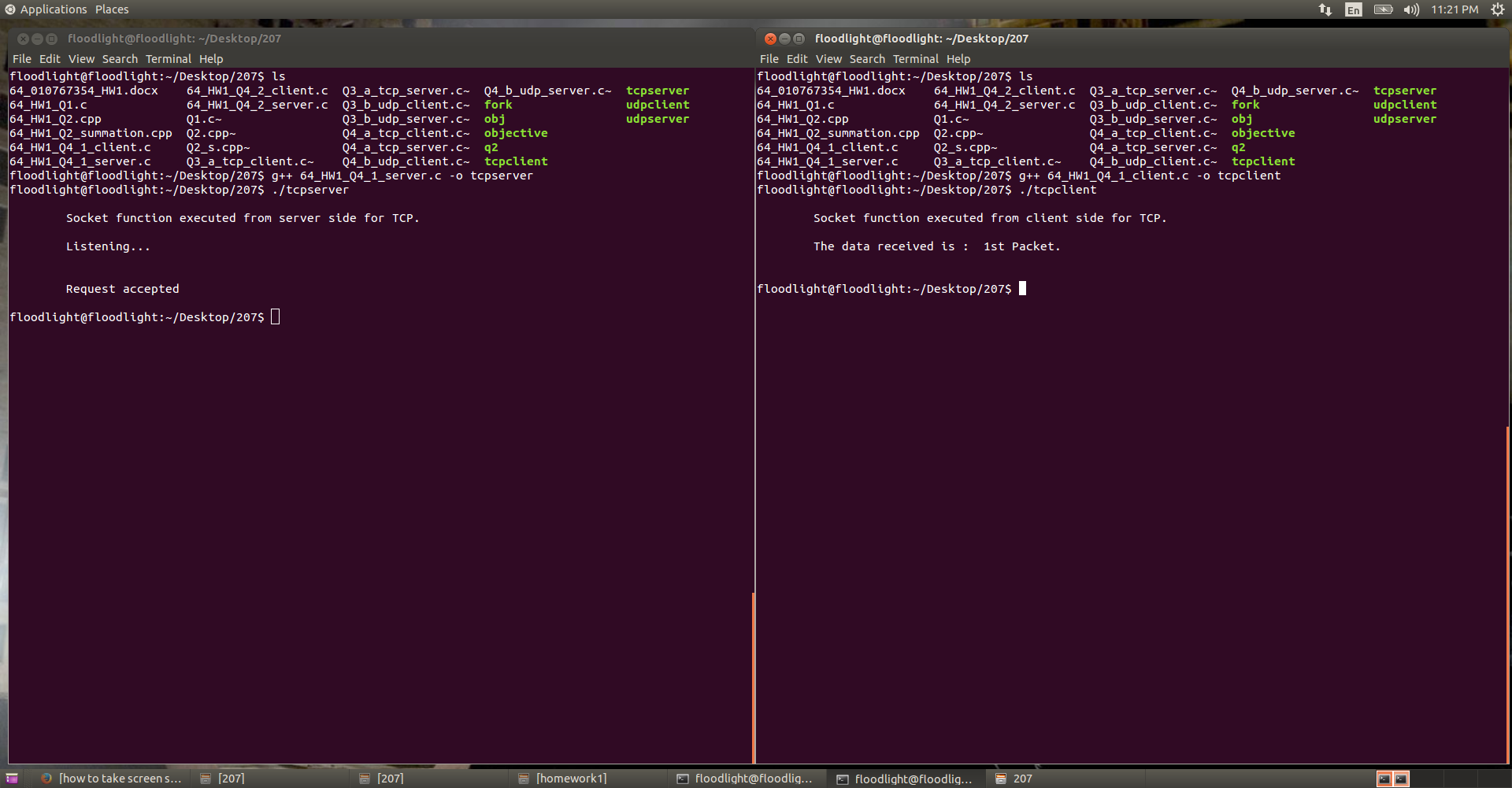
1. **What is the difference between processes and threads?**

**Answer:**

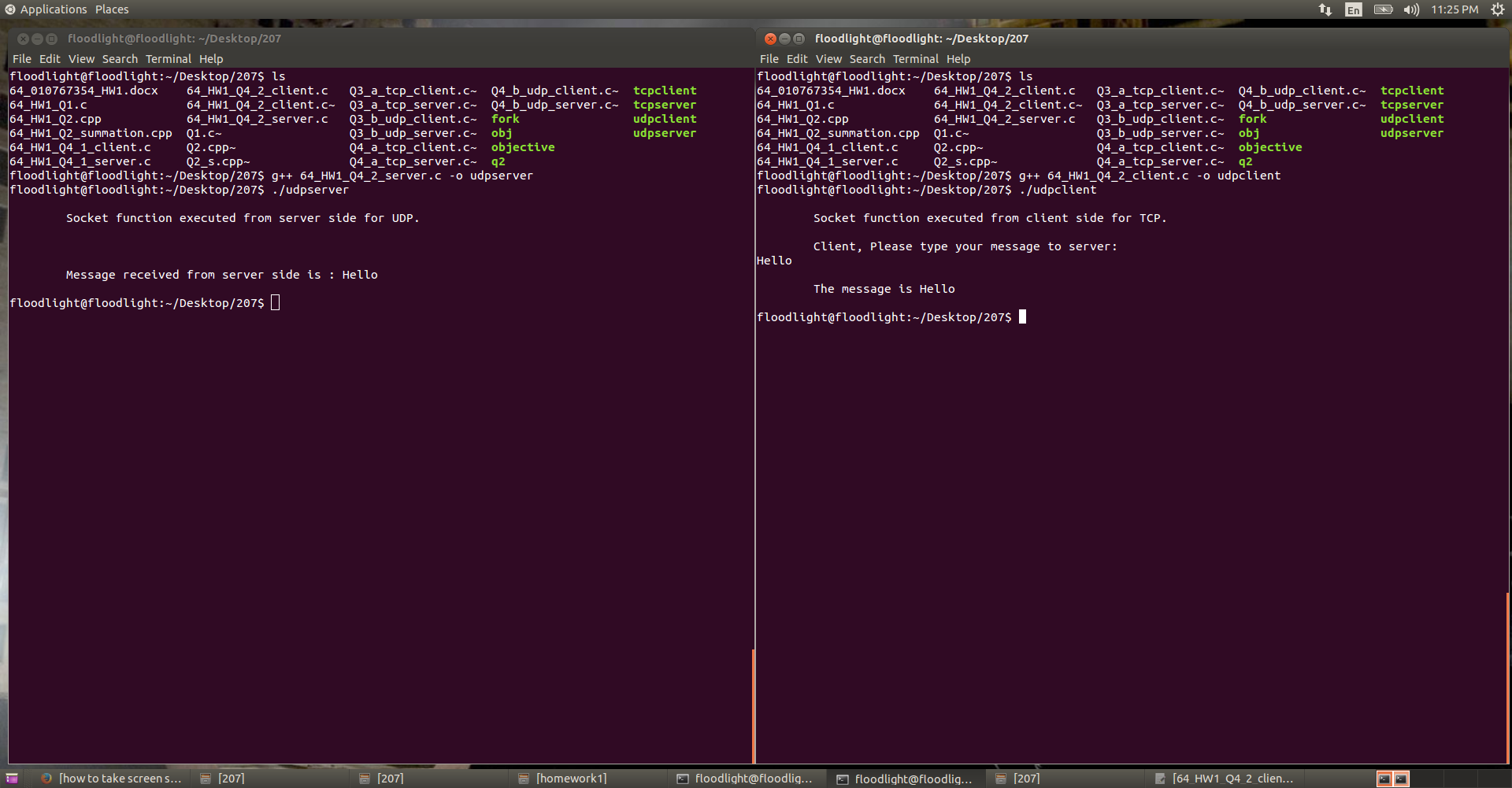
|  |  |
| --- | --- |
| **Processes** | **Threads** |
| * A process is an executing instance of an application or in other words, a process is a program in execution. | * A thread is a path of execution within a process. |
| * A process can consist of multiple threads. | * A thread cannot consist of any process. |
| * Processes are independent of each other. | * Threads, since they share the same address space are interdependent. So, caution must be taken so that different threads don’t step on each other. |
| * Process will by default not share memory space. | * Threads will by default share memory space. |
| * Processes don’t share most of the file descriptor. | * Threads will share file descriptors. |

**Question 4 (A) Write your server and client programs by using TCP as well as UDP . Just follow the socket programming flow like socket()🡪bind()🡪 and so on. No concurrency, non-blocking I/O, Simple message prints.**

1. **TCP based Server/Client**

****

1. **UDP based server/client**

****

**Question 4 (C) Explain main socket APIs in detail like Socket(), Bind() etc.**

**Answer:**

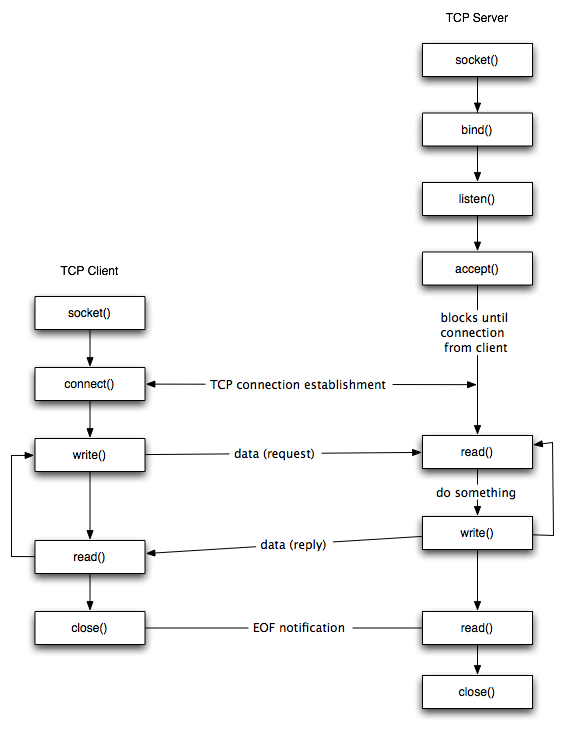
**For TCP:**

* **Server side sequence of API system calling would be:**

**Socket()🡪bind()🡪listen()🡪accept()🡪read()🡪write()🡪read()🡪close()**

* **Client Side sequence of API system calling would be:**

**Socket()🡪connect()🡪write()🡪read()🡪close()**



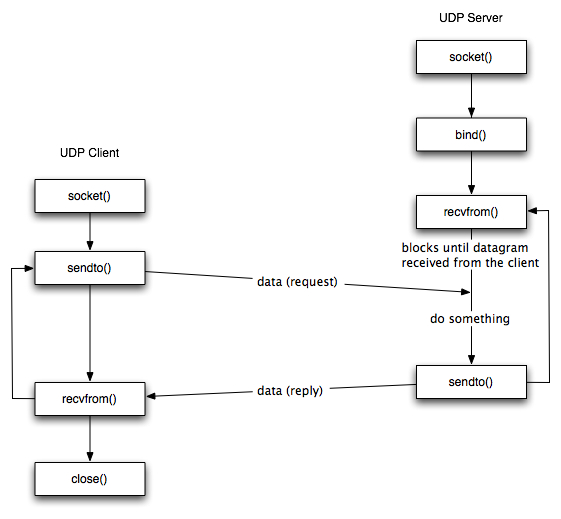
**For UDP:**

* **Server Side sequence of APIs would be:**

**Socket()🡪bind()🡪recvfrom()🡪sendto**

* **Client Side sequence of API system calling would be:**

**Socket()🡪sendto()🡪recvfrom()🡪close()**



**Now, I will explain all the details about the above APIs:-**

1. **Int socket (int domain, int type, int protocol) (CLIENT & SERVER SIDE)**

The function socket() creates an endpoint for communication and returns a socket descriptor for the socket. Socket() takes three arguments.

* Domain, which specifies the protocol family of the created socket. For example:
* AF\_INET for network protocol IPv4
* AF\_INET6 for IPv6
* AF\_UNIX for local socket(using a local file)
* Type, like:
* SOCK\_STREAM (Reliable connection oriented TCP network type)
* SOCK\_DGRAM (Fast unreliable connectionless UDP network type)
* SOCK\_RAW (Raw protocols atop the network layer)
* Protocol specific protocol
* UNSPEC (Unspecified)
* (PF\_INET and SOCK\_STREAM already implies TCP)

1. **Int bind (int sockfd, (struct sockaddr \*) &myaddr, socklen\_t addrlen) (SERVER SIDE)**

This function assigns a socket to an address. When a socket is created using socket(), it is only given a protocol family,but not assigned an address. Bind() takes three arguments.

* Sockfd, a socket descriptor representing the socket to perform the bind to.
* My-addr, a pointer to sockaddr structure representing the address to bind to.
* Addrlen, a socklen\_t field specifying the size of the sockaddr structure.

1. **Int listen (int sockfd, int backlog): (SERVER SIDE)**

After a socket has been associated with an address, listen() prepares it for incoming messages.

* Sockfd, a valid socket descriptor
* Backlog, an integer representing the number of pending connections that can be queued up at any one time. On success, 0 is returned. If an error occurs, -1 is returned.

1. **Int connect (int sockfd, (struct sockaddr\*) &servaddr, socklen\_t addrlen) (CLIENT SIDE)**

The connect() call establishes a direct communication link to a specific remote host identified by its address via a socket, supported by a socket descriptor.

* Sockfd, a socket descriptor representing the socket to perform the bind to.
* My-addr, a pointer to sockaddr structure representing the server address.
* Addrlen, a socklen\_t field specifying the size of the sockaddr structure.
* Connect() returns an integer representing the error code.
* 0 represents success.
* -1 represents error.

1. **Int accept (int sockfd, (struct sockaddr\*) &cliaddr, socklen\_t addrlen) (SERVER SIDE)**

* Sockfd, a socket descriptor representing the socket to perform the bind to.
* My-addr, a pointer to sockaddr structure representing the client address.
* Addrlen, a socklen\_t field specifying the size of the sockaddr structure.

1. **Int read/int write, int rec/send (int sockfd, void \*buff, size\_t mbytes) (BOTH SIDE)**

* Int recvfrom(int sockfd, void \*buff, size\_t mbytes, int flags, (struct sockaddr \*) from, socklen\_t addrlen)
* Int sendto((int sockfd, void \*buff, size\_t mbytes, int flags, (struct sockaddr \*) to, socklen\_t addrlen)

1. **Int close (int sockfd) (CLIENT SIDE)**

* Sockfd, only closes the socket descriptor for the process.
* Leaves the connection open if there is another process using the same socket.
* Shutdown(), breaks the connection for all the processes using the socket.

**Question 5 Install Mininet to explore Software Defined Networking(SDN) based on the manual on Canvas. Create a simple topology and print out the flow rules for your topology. Explain the flow rules that you capture.**

**Answer:**



* Cookie: this 64 bit unique identifier will identify the address of the host.
* Duration: As the name suggests, this is the time period starting from establishing a connection to sending packets to terminate the connection between two devices in the network.
* Table: Each and every node’s enrty, in this case that will be the host’s table with the connected node.
* n\_packets: The number of Packets
* Idle\_age: The time period for which the flow has not matched any packets
* Idle-timeout: The time a connection can stay open without making any request
* Priority: the priority given to the node in the network connected to that host
* In\_port: the port number
* Dl\_src: Source MAC address
* Dl\_dst: Destination MAC address
* Nw\_src:source IP Address
* Nw\_dst: Destination IP address
* Actions: Instructions as in what to do with the packet.