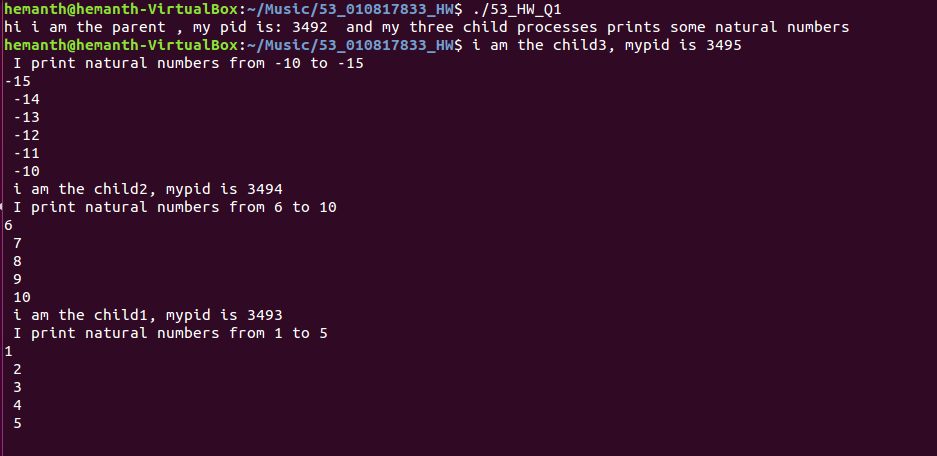
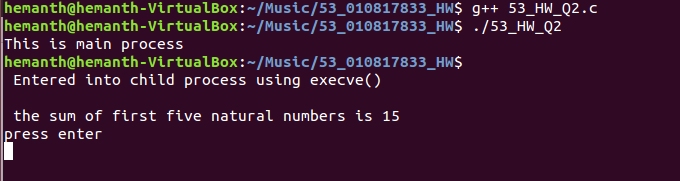
Q1. Write a concurrent program (using fork()) that starts three processes. Arrange for each process to print a few lines of output and then halt.

Solution:



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



Q3. Explain the following questions.

(1) What is Network Byte Order?

(2) What is the difference between processes and threads?

Solution:

(1)Network Byte Order:

To define the network byte order, the concept of Endianness is to be discussed

**Endianness:** Endianness in the networking world defines the order in which the bytes are transmitted over a digital link. Words may be represented in two formats:

1. Big-endian and
2. Little-endian

* In **Big-endian** format, the most significant byte of the word is stored at a particular memory location and the subsequent bytes are stored in the following higher memory locations, thus storing the LSB bit at the top address.
* To illustrate this more efficiently, consider a two-byte hex-number a56d, you will store it in two sequential bytes in the memory locations i.e., a5 followed by 6d. Thus storing the LSB(6d) in the higher memory location than the MSB(a5). In short, number stored with bi end first is called Big-endian.
* Big-endian is the most common format in data networking, in most protocols like TCP, UDP, IPV4, IPV6 data is transmitted in Big-endian order. Hence, Big-endian byte order is also referred as **Network Byte order**
* **Little-endian** format is reverse of Big-endian order i.e., LSB is stored at lower memory address while MSB being on top.

(2) Differences between Thread and Process:

Thread:

* Thread is an executing instance of an application and are used for executing a part of an application
* Usually threads are used for small tasks.
* In other words, thread could be considered as a ‘light weight’ process.
* A thread can do anything a process can do.
* Different threads executing within a process share the same address space.

Process:

* A process can be defined collectively as execution of multiple threads.
* Processes are used for more ‘heavy weight’ tasks like execution of applications.
* Processes are used for execution of large programs.
* On the contrary to threads, different process does not share the same address space.

Q4. 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, blocking I/O, Simple message prints.

(1) TCP-based server/client

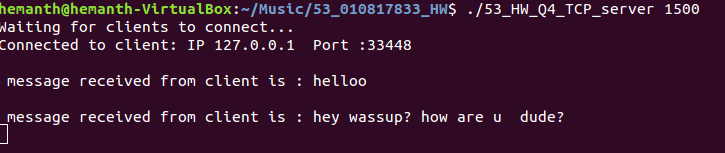
(2) UDP-based server/client

(3) Explain main socket APIs in detail like socket()->bind()->.

1. TCP-based server/client:

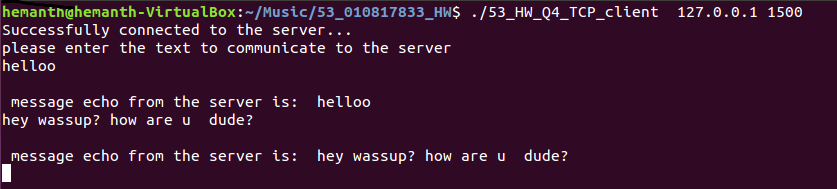
Server

Arguments: #port\_number



Client

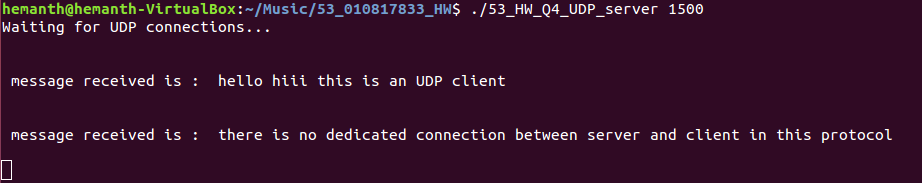
Arguments: #ip\_address #port\_number



(2) UDP-based server/client

Server:

Arguments : #port\_number



Client:

Arguments: #ip\_address #port\_number



1. Main socket APIs:

Socket(): A socket is an end point of a connection across a computer network. The function call socket() returns a *socket descriptor* which points to a linked list that consists of the server details like the domain, protocol being used and communication semantics.

Usage: socket(int domain, int type, int protocol);

* It provides handle that a local program can pass to the API’s to use the connection

**bind() :**

Bind is used to associate the socket to a port number on our local machine. Bind assigns the address specified by addr to the socket referred to by the file descriptor.

* Usage: int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);
* Sockfd -> socket file descriptor that is created using socket()
* Sockaddr -> It is the address specified by addr to the socket referred to by the file descriptor
* Addrlen -> is the length of the address in bytes.

**Listen() :**

* The listen() function marks a connection-mode socket (e.g, SOCK\_STREAM), specified by the socket argument like accepting connections.
* It also restricts the number of outstanding connections in the socket's listen queue to the value specified by the backlog argument.
* The socket is put into passive mode where incoming connection requests are acknowledged and queued pending acceptance by the process.
* listen() also attempts to continue to function rationally when there are no available descriptors. It accepts connections until the queue is emptied and resume listening for incoming connections.

Usage : listen(int socket\_descriptor, int backlog);

**send():**

The *send*() function will initiate transmission of a message from the specified socket to its peer and sends a message only when the socket is connected.

**send()** returns the number of bytes actually sent out.

The function takes the following arguments:

*Socket :* Specifies the socket file descriptor.

*Buffer :* Points to the buffer containing the message to send.

*Length :* Specifies the length of the message in bytes.

*Flags :* Specifies the type of message transmission

Successful completion of a call to *send*() does not guarantee delivery of the message. An error will be returned if its locally detected.

Usage: send(int socket\_descriptor,const void \*buffer, size\_t len, int flags);

**recv():**

* The *recv*() function will receive a message from a connection-mode or connectionless-mode socket.
* It is normally used with connected sockets because it does not permit the application to retrieve the source address of received data.

The *recv*() function takes the following arguments: socket, buffer, length.

*Socket* Specifies the socket file descriptor.

*Buffer* Points to a buffer where the message should be stored.

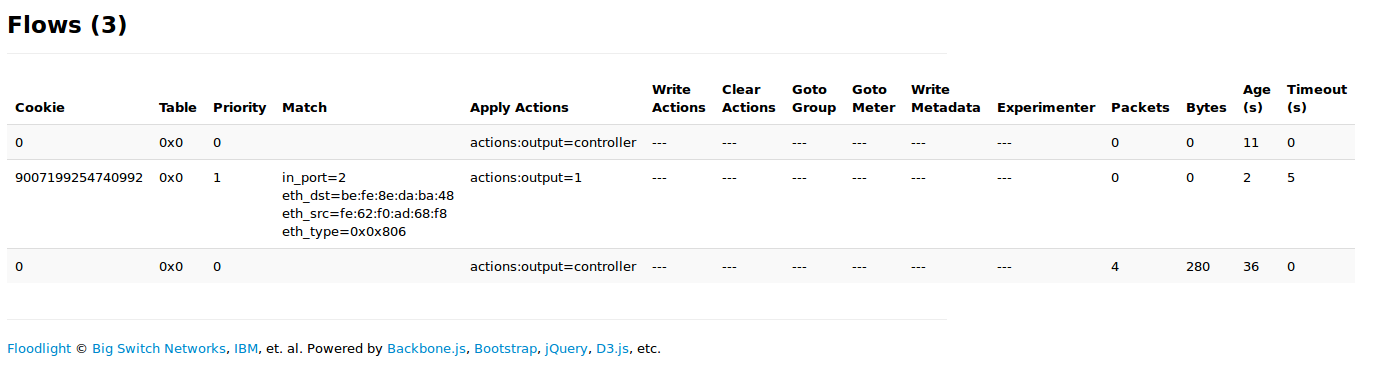
*Length* Specifies the length in bytes of the buffer pointed to by the *buffer* argument.

*Flags* Specifies the type of message reception.

Usage: recv(int socket\_descriptor, void \*buffer, size\_t len, int flags);

Q5. Create a simple topology and print out the flow rules for your topology. Explain the flow rules that you capture.

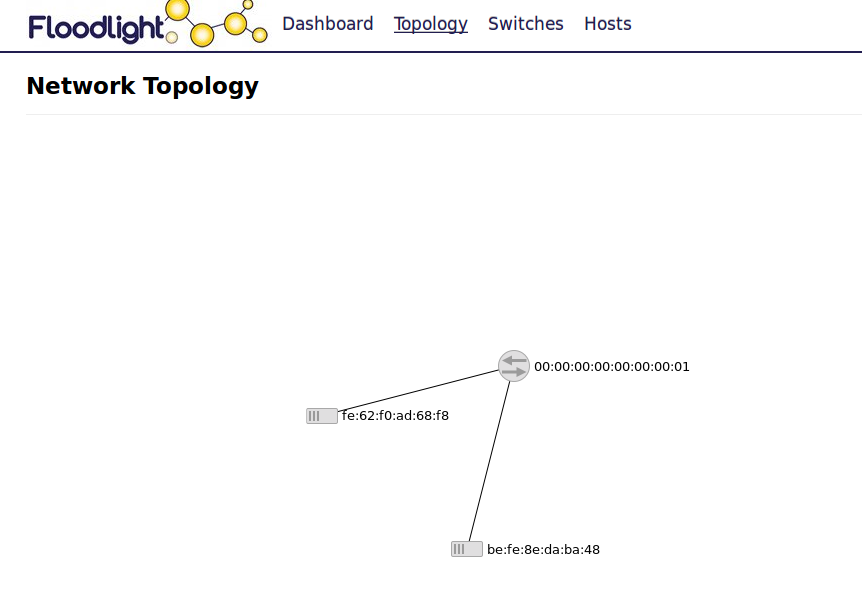
Solution:

****

**Explanation:**

**Match field:** To match against packets. These consist of the ingress port and packet headers and optionally other pipeline fields such as metadata specified by a previous table.

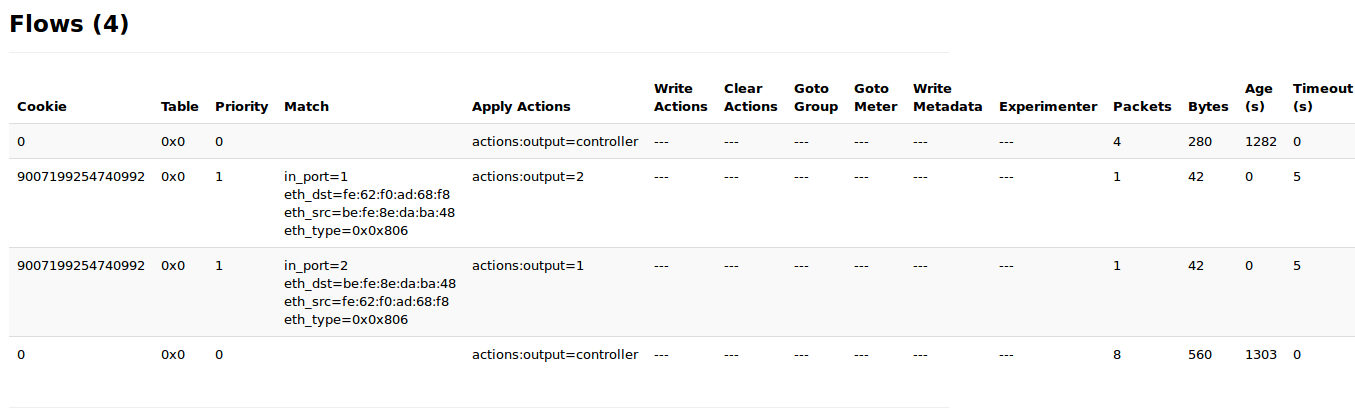
In the above topology be:fe:8e:da:ba:48 and fe:62:f0:ad:68:f8 are the two hosts and are connected by a switch

****

**Priority:** matching precedence of the flow entry.

**Timeouts:** maximum amount of time or idle time before flow is expired by the switch

**Cookie:** opaque data value chosen by the controller. May be used by the controller to filter flow statistics, flow modification and flow deletion. Not used when processing packets.



**Meta data**: the mask specifies which bits of the metadata register should be modified.

**Apply actions:** applies the specific actions immediately without any change to the action set. This instruction may be used to modify the packet between two tables or to execute multiple actions of the same type. The actions are specified as an action list.s