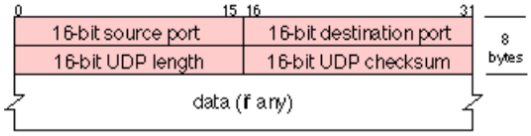
Sathyanarayana Ramesh
UDP Client-Server Interaction

PHASE 1: SPECIFICATION:

The User Datagram Protocol (UDP) is a transport layer protocol defined for use with the IP network layer protocol. UDP provides a minimal, unreliable, best-effort, message-passing transport to applications and upper-layer protocols. The service provided by UDP is an unreliable service that provides no guarantees for delivery and no protection from duplication. To transmit a UDP datagram, a computer completes the appropriate fields in the UDP header and forwards the data together with the header for transmission by the IP network layer.



The UDP protocol header consists of 8 bytes of Protocol Control Information (PCI)

The UDP header consists of four fields each of 2 bytes in length:

- **Source Port** - UDP packets from a client use this to indicate the session on the local client that originated the packet.
- **Destination Port** - UDP packets from a client use this to indicate the service required from the remote server.
- **UDP length** - The number of bytes comprising the combined UDP header information and payload data.
- **UDP Checksum** - A checksum to verify that the end to end data has not been corrupted by routers or bridges in the network.

A UDP datagram is carried in a single IP packet and is hence limited to a maximum payload of 65,507 bytes for IPv4 and 65,527 bytes for IPv6. To transmit a UDP datagram, a computer completes the appropriate fields in the UDP header and forwards the data together with the header for transmission by the IP network layer.

UDP Client Server Interaction:

A. UDP Server

- The UDP server initialize a UDP socket. UDP socket routines enable simple IP communication using the user datagram protocol (UDP).
- Binding the socket to a specific port. Binding of a socket is done to address and port in order to receive data on this socket or to use this address/port as the source of the data when sending data.
- The server continuously listens for the incoming UDP packets on this port.
- When the sever receives the message, it reads the data from the packets and prints the data(integers sent from client) to the console.

B. UDP Client

- The UDP client initializes a UDP socket.

- The client reads from an input file, each line in the file represents a set hexadecimal bytes. Client read bytes from a file, convert them to integers, and send them to the server.
- A delay of 10 seconds is introduced between sending each packets.

C. BUFFER SIZE

- The buffer size, in the context of network programming, specifies the amount of memory allocated to temporarily store data that is being sent or received over the network.
- Both the client and server need to agree on the size of the data chunks they are exchanging. If the client sends data using one buffer size and the server receives using a different buffer size, leads to misinterpretation of data.
- The choice of the buffer size has been set to the size of '***unsigned long long***' which is typically **8 bytes.** Choice have been done with the assumption that the data being sent between the client and server is of type unsigned long long.

[2023-12-05 12:41:50] Server: Server started and listening on port 8080
[2023-12-05 12:41:52] Client: Client started
[2023-12-05 12:41:52] Client: Sent integer to server: 28772997619311
[2023-12-05 12:41:52] Server: Received message from client at IP 127.0.0.1: 28772997619311
[2023-12-05 12:42:02] Client: Sent integer to server: 187723572702975
[2023-12-05 12:42:02] Server: Received message from client at IP 127.0.0.1: 187723572702975
[2023-12-05 12:42:12] Client: Sent integer to server: 17730434519136

PHASE 2 DESIGN:

2.1 Functions and its purpose

The implementation of the program can break down into two individual server and client codes. Below are the methods commonly used in both client and server.

A. Log message - ***void logMessage(string source, string message)***
This is a custom function defined to log messages to both the standard output(stdout) and a log file. This function accepts two parameter source which indicates who is sending the message and message being the actual message being sent.

B. Main method – ***int main()***
The runtime system starts the program by calling main() function first. The main() function then calls all the other functions required to run the program. If the program accepts command line arguments then the definition changes to ***'int main(int argc, char *argv[])'*** where argc is the number of arguments and argv is the list of arguments. The return value of the function is integer, '0' which indicates success no error in execution, -1 indicates program exited due to error.

C. Socket function - ***socket(int domain, int type,int protocol)***
Thid function creates a new socket, accepts three parameters. AF_INET is the communication domain (IPv4/IPv6). ***SOCK_DGRAM*** is the type of socket for UDP. The protocol to be used (usually set to 0 for default protocol). The method could be found in **<sys/socket.h>** header file.

D. ***'memset'*** Function - ***void* memset(void* ptr, int value, size_t num)***

The purpose of **memset** is to fill a block of memory with a particular value. This method needs three parameters. *void* ptr* which is a Pointer to the memory block, *int value* is the value to be set. *size_t num* the number of bytes to be set. The method could be found in **<cstring>** header.

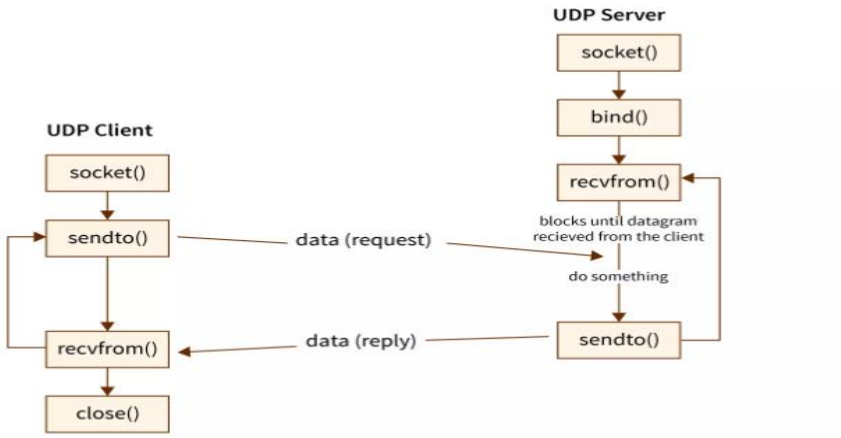
E. ***'bind'*** Function - ***int bind(int sockfd, const struct sockaddr* addr, socklen_t addrlen)***
This function associates a socket with a specific IP address and port. Parameters are int sockfd which is Socket file descriptor, const struct sockaddr* addr is the pointer to a structure containing the details of the address to bind to and socklen_t addrlen is the size of the address structure. This function could be found in **<sys/socket.h>** header.

F. ***recvfrom*** Function- ***ssize_t recvfrom(int sockfd, void* buf, size_t len, int flags, struct sockaddr* src_addr, socklen_t* addrlen)***
The main purpose of this method is to receive a message from a socket and captures the sender's address. The Parameters are int sockfd - Socket file descriptor. void* buf - Pointer to the buffer where the message will be stored. size_t len - Maximum number of bytes to be received. int flags - Flags (usually set to 0). struct sockaddr* src_addr - Pointer to the structure where the sender's address will be stored. socklen_t* addrlen - Pointer to the size of the address structure. This method could be found in **<sys/socket.h>** header.

G. ***sendto*** Function - ***ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,const struct sockaddr *dest_addr, socklen_t addrlen)***
The sendto function is used to send a message from a socket (sockfd) to a specific destination address (dest_addr). This function is commonly used in UDP socket programming. The Parameters remains same as the recvfrom except Pointer to a structure containing the destination address information. socklen_t addrlen - Size of the destination address structure. Returns the number of bytes sent on success, or -1 on error. This method could be found in <sys/socket.h> header.

H. ***close*** Function:
This method closes a file descriptor (in this case, the socket). Ex close(serverSocket). Parameters are the file descriptor to be closed. This method could be found in Header <unistd.h>.

2.2 Block Diagram:



PHASE 3: RISK ANALYSIS

UDP is a simple protocol because it doesn't require connection setup(handshake) or acknowledgement exchanges to send data packets to their destinations. It just transfers the packet and doesnot know if the data reaches the destination or drops off somewhere along the path. This limited packet verification subjects UDP to network vulnerabilities. Bad actors might use port scan attack to gauge UDP services as a potential target.

PHASE 4: VERIFICATION

To view the overall client-server interaction, server has to be up and running first, listening to the port to receive the messages. Client has to be started subsequently which sends the messages to the server. The client reads bytes from an input file, and converts those hexa decimal values to integers and sends to the server. It is crucial for the input file to be present in the specified path else the client program would end up in the error. The program has been tested with different input files of different sizes to ensure the validity of the programs.

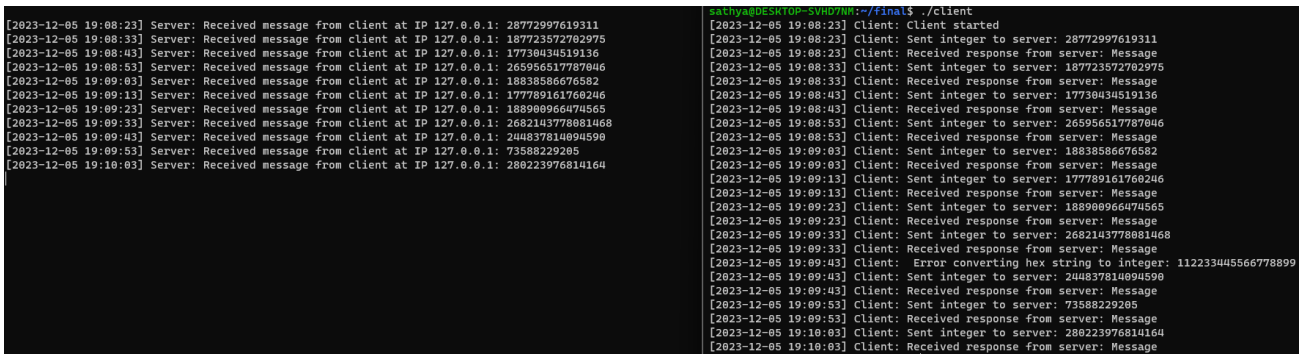
PHASE 5: CODING

The code has been pushed to the git hub, and the link has been provided below.

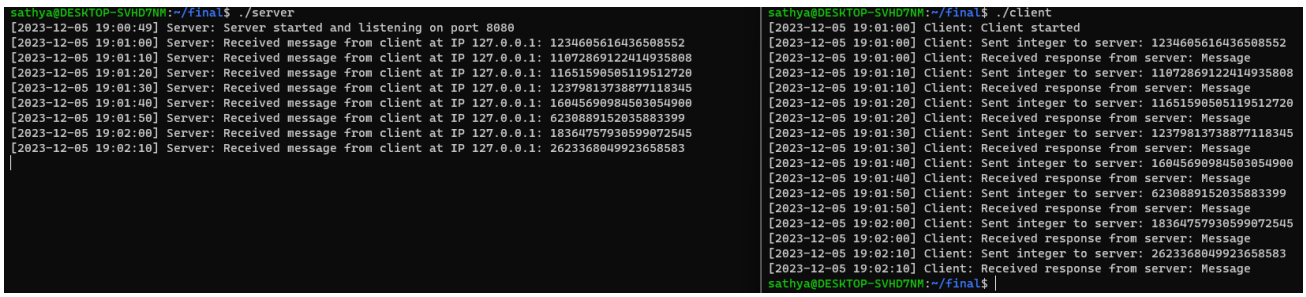
https://github.com/akgWMU/pa6-udp-client-server-interaction-Sathya-Ramesh

PHASE 6: TESTING

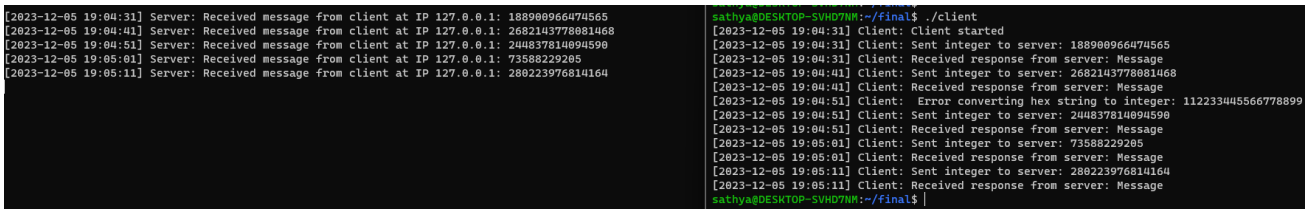
1. Testing of client server interaction with ‘hw6input.txt’ as input file.



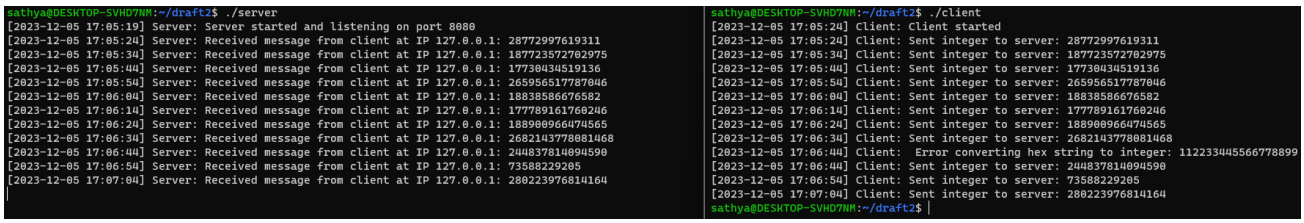
2. Testing of client server interaction with ‘input1.txt’ as input file.



3. Testing of client server interaction with ‘input2.txt’ as input file.

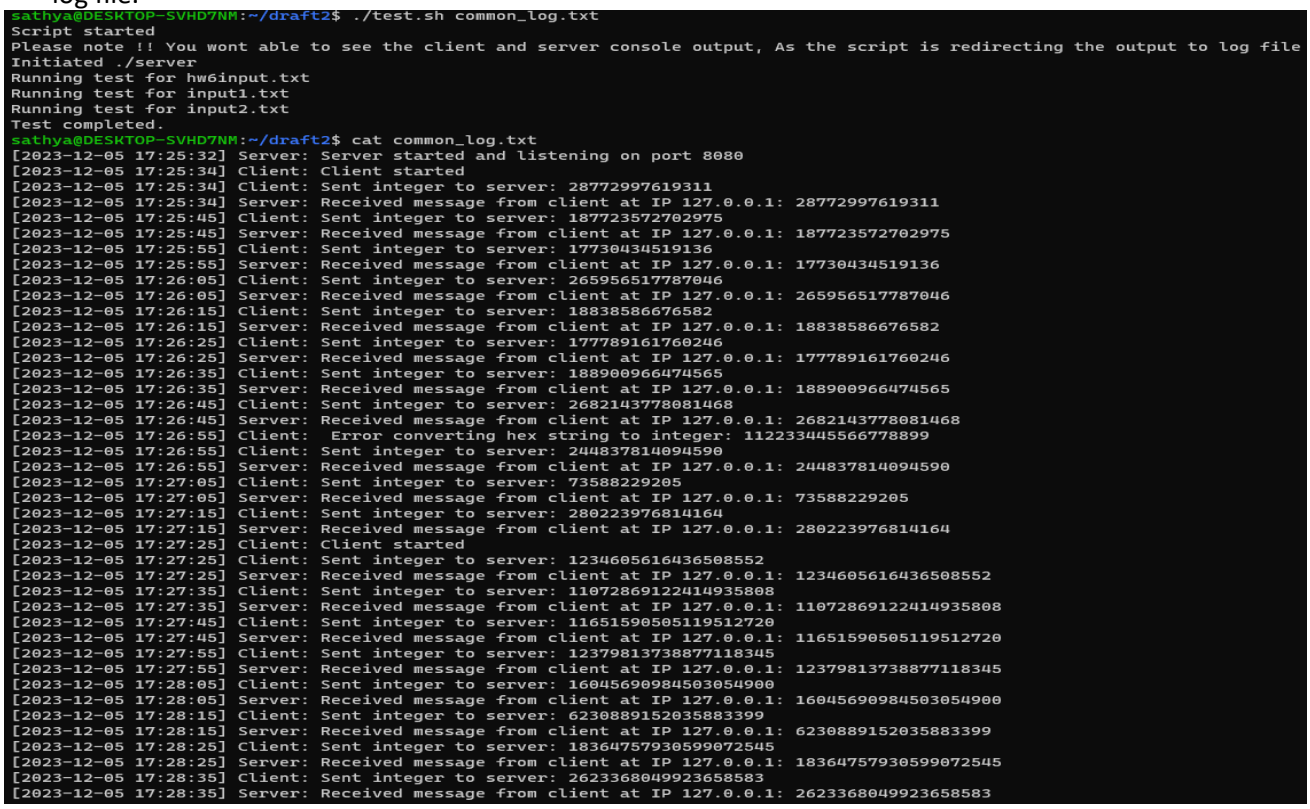


4. Testing of client server interaction with buffer size as 2048.



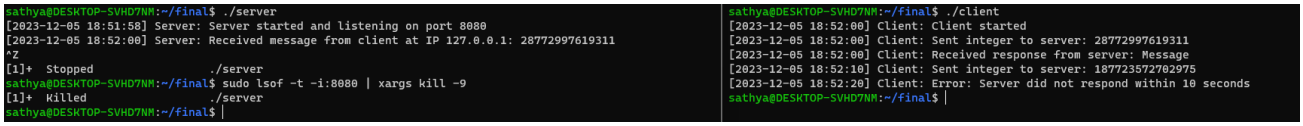
5. Server port binding error.

6. Additional testing. This script runs the testcases and logs both client server message to same log file.



7. Server not started.

8. Server stopped in between.



PHASE 7: REFINING THE PROGRAM

For enhanced UDP reliability, future improvements include implementing error detection, acknowledgment, and retransmission strategies. Integrating checksums or CRC ensures data integrity, while acknowledgment mechanisms and selective repeat protocols address packet loss. Dynamic timeout adjustments, flow control, and congestion avoidance optimize performance.

PHASE 8: CHALLENGES

Security concerns arise without built-in encryption. Balancing UDP's lightweight design with reliability needs presents difficulties. Adapting to dynamic network changes, handling large data transfers efficiently, and managing buffer sizes for optimal performance pose additional challenges.

PHASE 9: PRODUCTION

The important program files client.cpp, server.cpp, hw6input.txt, input1.txt, input2.txt and test.sh script has been pushed to the github and the same has been compressed and attached along with the report in dropbox submission.

PHASE 10: REFERENCES

[1] https://www.geeksforgeeks.org/udp-server-client-implementation-c/
[2] https://www.guru99.com/cpp-file-read-write-open.html
[3] https://www.codingninjas.com/studio/library/learning-socket-programming-in-c
[4] https://erg.abdn.ac.uk/users/gorry/course/inet-pages/udp.html#:~:text=A%20UDP%20datagram%20is%20carried,packets%20usually%20requires%20IP%20fragmentation.
[5] https://stackoverflow.com/questions/39314086/what-does-it-mean-to-bind-a-socket-to-any-address-other-than-localhost
[6] https://www.keil.com/pack/doc/mw/Network/html/group__net_u_d_p__func.html#:~:text=Description,features%20at%20the%20transport%20layer.