Assignment A5

Problem Statement:

Write a program using UDP sockets to enable file transfer (script, text, audio and videos) between two machines. Demonstrate the packets captured traces using wireshark packet analyzer tool using peer to peer method.

Objectives:

1) To understand UDP sockets using c/c++.
2) Analyze the packets using wireshark.

S/W & H/W requirements:

1) Ubuntu
2) Code editor
3) gcc compiler
4) Wireshark Packet analyzer Tool

Theory:

Datagram packets are used to implement a connectionless packet delivery service supported by UDP protocol. Each message is transferred from source machine to destination based on information contained within that packet. That means, each packet needs to have destination address & each packet might be routed differently and might arrive in any order. Packet delivery is not guaranteed.

```
        Server                          Client
      ┌─────────┐                     ┌─────────┐
      │ socket()│                     │ Socket()│
      └─────────┘                     └─────────┘
           ↓                               ↓
      ┌─────────┐                          
      │ bind()  │                          
      └─────────┘                          ↓
           ↓                          ┌─────────┐
      ┌──────────────┐                │ Sendto()│←──┐
      │ recvfrom()   │                └─────────┘   │
      └──────────────┘                              │
      wait for datagram arrival                     │
           ↓                                        │
      Process request                               │
           ↓                                        │
      ┌─────────┐      reply        ┌───────────┐   │
      │ Sendto()│ ─────────────────→│ recvfrom()│───┘
      └─────────┘                   └───────────┘
                                         ↓
                                    ┌───────┐
                                    │ Close │
                                    └───────┘
```
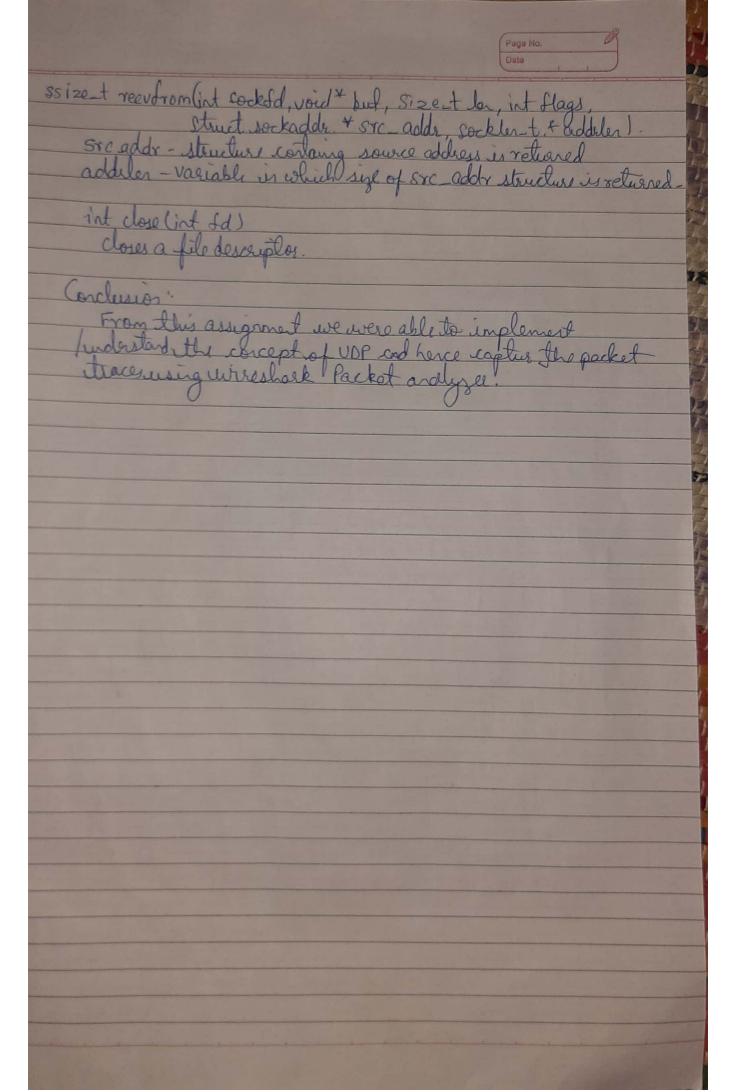
UDP server
1) Create UDP socket
2) Bind the socket to server address
3) Wait until datagram packet arrives from client
4) Process the datagram packet & send a reply to client.
5) Go back to step 3.

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

Function
1) int socket (int domain, int type, int protocol).
   Domain - AF_INET OR AF_INET6.
   Type -- SOCK_STREAM OR SOCK_DGRAM
   Protocol - 0 means use default protocol.

2) int bind (int sockfd, const struct sockaddr *addr, socklen_t addrlen)
   Sockfd - File descriptor of socket to be binded
   addr - Structure in which address to be binded is specified
   addrlen - Size of addr structure.

3) ssize_t sendto (int sockfd, const void *buf, size_t len, int flags,
                   const struct sockaddr *dest_addr, socklen_t addrlen)
   buf - Application buffer containing the data to be sent
   len - Size of buf application buffer
   flags - Bitwise OR of flags to modify socket behaviour
   dest_addr - structure containing address of destination
   addrlen - size of dest addr structure

ssize_t recvfrom(int sockfd, void* buf, size_t len, int flags,
        struct sockaddr * src_addr, socklen_t * addrlen).
src_addr - structure containing source address is returned
addrlen - variable in which size of src_addr structure is returned.

int close(int fd)
    closes a file descriptor.

Conclusion:
    From this assignment we were able to implement
/understand the concept of UDP and hence capture the packet
trace using wireshark Packet analyzer.

```
        printf("Waiting for client...\n");
50
51      bytes = recvfrom(sockfd,file_name,1024,ser
52
53      bytes = recvfrom(sockfd,net_buf,700000,ser
54
55      fp = fopen(file_name,"wb");
56
57      char file_buf[1024];
58      int i = 0;
59      size_t f;
60      // while(1) {
61      //      if(net_buf[i] == '\0') break;
62      //      file_buf[i] = net_buf[i];
63      //      i++;
64      // }
65      // file_buf[i] = '\0';
66      printf("%s Hey",net_buf);
67      if((fwrite(net_buf,strlen(net_buf),1,fp))
68          printf("\nFile Transfer failed!\n");
69      } else {
70          printf("\nFile Successfully Transferr
71      }
72
73      if (fp != NULL)
74          fclose(fp);
75      }
76
77  } else {
78      printf("\nBind Failed\n");
79  }
80
81
82  return 0;
83  }
```

Terminal — Client:
```
kkaneki@loading:~$ cd CNL/FILE/Client/
kkaneki@loading:~/CNL/FILE/Client$ gcc client.c
kkaneki@loading:~/CNL/FILE/Client$ ./a.out

Client Socket created

Please enter file name to receive:
```

Terminal — Server:
```
kkaneki@loading:~$ cd CNL/
kkaneki@loading:~/CNL$ cd FILE/
kkaneki@loading:~/CNL/FILE$ cd Server/
kkaneki@loading:~/CNL/FILE/Server$ gcc server.c
kkaneki@loading:~/CNL/FILE/Server$ ./a.out

Socket Created

Bind Success
Waiting for client...
```

Edit Selection View Go Run Terminal Help

EXPLORER   C client.c   C server.c ✕

> OPEN EDITORS   Server > C server.c > ✷ main()

∨ FILE
  ∨ Client
    ≡ a.out
    C client.c
    ≡ client.txt
    JS two.js
  ∨ Server
    ≡ a.out
    ≡ client.txt
    C server.c

```c
49          printf("waiting for client...\n");
50
51          bytes = recvfrom(sockfd,file_name,1024,ser
52
53          bytes = recvfrom(sockfd,net_buf,700000,se
54
55          fp = fopen(file_name,"wb");
56
57          char file_buf[1024];
58          int i = 0;
59          size_t f;
60          // while(1) {
61          //      if(net_buf[i] == '\0') break;
62          //      file_buf[i] = net_buf[i];
63          //      i++;
64          // }
65          // file_buf[i] = '\0';
66          printf("%s Hey",net_buf);
67          if((fwrite(net_buf,strlen(net_buf),1,fp))
68              printf("\nFile Transfer failed!\n");
69          } else {
70              printf("\nFile Successfully Transferr
71          }
72
73          if (fp != NULL)
74              fclose(fp);
75      }
76
77      } else {
78          printf("\nBind Failed\n");
79      }
80
81
82      return 0;
83  }
```

> OUTLINE
> MAVEN

Terminal: kkaneki@loading: ~/CNL/FILE/Client

```
kkaneki@loading:~$ cd CNL/FILE/Client/
kkaneki@loading:~/CNL/FILE/Client$ gcc client.c
kkaneki@loading:~/CNL/FILE/Client$ ./a.out

Client Socket created

Please enter file name to receive:
client.txt
adfsdfsfsdf
Please enter file name to receive:
```

Terminal: kkaneki@loading: ~/CNL/FILE/Server

```
kkaneki@loading:~$ cd CNL/
kkaneki@loading:~/CNL$ cd FILE/
kkaneki@loading:~/CNL/FILE$ cd Server/
kkaneki@loading:~/CNL/FILE/Server$ gcc server.c
kkaneki@loading:~/CNL/FILE/Server$ ./a.out

Socket Created

Bind Success
Waiting for client...
adfsdfsfsdf Hey
File Successfully Transferred!
Waiting for client...
```

VS Code editor showing two.js file:

```
Server > JS two.js
  1    console.log("Hey");
```

Explorer panel:
- FILE
  - Client
    - a.out
    - client.c
    - client.txt
    - two.js
  - Server
    - a.out
    - client.txt
    - server.c
    - two.js

Client terminal — kkaneki@loading: ~/CNL/FILE/Client

```
kkaneki@loading:~$ cd CNL/FILE/Client/
kkaneki@loading:~/CNL/FILE/Client$ gcc client.c
kkaneki@loading:~/CNL/FILE/Client$ ./a.out

Client Socket created

Please enter file name to receive:
client.txt
adfsdfsfsdf
Please enter file name to receive:
two.js
console.log("Hey");
Please enter file name to receive:
```

Server terminal — kkaneki@loading: ~/CNL/FILE/Server

```
kkaneki@loading:~$ cd CNL/
kkaneki@loading:~/CNL$ cd FILE/
kkaneki@loading:~/CNL/FILE$ cd Server/
kkaneki@loading:~/CNL/FILE/Server$ gcc server.c
kkaneki@loading:~/CNL/FILE/Server$ ./a.out

Socket Created

Bind Success
Waiting for client...
adfsdfsfsdf Hey
File Successfully Transferred!
Waiting for client...
console.log("Hey"); Hey
File Successfully Transferred!
Waiting for client...
```

```c
            bytes = recvfrom(sockfd,file_name,1024,sendrecvflag,(struct sockaddr*)&addr_con, &addrlen);
            FILE* fp = fopen(file_name,"w");
            size_t f;
            printf("FileName: %s\n",file_name);
            f = recvfrom(sockfd,net_buf,700000,sendrecvflag,(struct sockaddr*)&addr_con, &addrlen);


            printf("%lu\n",f);
            if((fwrite(net_buf,f,1,fp)) < 0) {
                printf("\nFile Transfer failed!\n");
            } else {
                printf("\nFile Successfully Transferred!\n");
            }
            memset(file_name, '\0', sizeof(file_name));
            if (fp != NULL)
                fclose(fp);
        }

    } else {
        printf("\nBind Failed\n");
```

TERMINAL

```
 3.VIDEO
4.EXIT^C
kkaneki@kkaneki:~/CNL/FILE/Client$ gcc client.c
kkaneki@kkaneki:~/CNL/FILE/Client$ ./a.out

Client Socket created

Please enter file name to receive:
video.mp4
31036
Please enter file name to receive:
```

```
kkaneki@kkaneki:~/CNL/FILE/Server$ ./a.out

Socket Created

Bind Success
Waiting for client...
FileName: video.mp4
31036

File Successfully Transferred!
Waiting for client...
```