COMPUTER NETWORK LAB

ASSIGNMENT 1

DATE: 27/7/18

A1.1 finding out what networking devices are installed in the department.

Ans.

Network Switch:

It is used to connect multiple network host and to transfer data packet. It filters the packet and sends only to the interface of the intended filter.

Modem:

It stands for (Modulator Demodulator). It modulates and demodulates the signal between the digital data of a computer and the analog signal of a telephone line.

Network Router:

A network router is responsible for routing traffic from one to another router.

Bridge:

A bridge connects two sub networks as a part of the same network.

Repeater:

It amplifies the signal it receives.

A1.2 Describe the network type and topology of the department.

Ans.

Our department most probably has a hybrid topology. It is two different types of topologies which is a mixture of two or more topologies. For example if in an office in one department ring topology is used and in another star topology is used, connecting these topologies will result in Hybrid Topology (ring topology and star topology).

A1.3 File and printer sharing in different OSs.

Ans.

File sharing is the public or private sharing of computer data or space in a network with various levels of access privilege.

File sharing in windows:

1. Start->control panel->

2. Network and home group->change advanced sharing settings

3. Turn on network discovery and file and printer sharing

4. Save changes

A1.4 Network address configuration in different OSs.

Ans.

1. Start->control panel

2. Change adapter settings

3. Click on the **Internet Protocol Version 4 (TCP/IPv4)** (you may need to scroll down to find it). Next, click on the **Properties** button.

4. Next, click the **Use the following DNS server addresses:** radio button. Next, in the **Preferred DNS server:**, and **Alternate DNS server:** number fields, input the numbers that were assigned by OIT. Then click the **OK** button.

A1.5 Finding the IP and MAC address in different OSs.

Ans.

1. In windows type command: ipconfig

2. In linux type command: ifconfig

A1.6 Work group and domain group configuration.

Ans.

A **workgroup** is a peer-to-peer network using Microsoft software. A workgroup allows all participating and connected systems to access shared resources such as files, system resources and printers.

Steps to configure workgroup in windows:

1. Navigate to Control Panel, System and Security and System to access your computer details.
2. Find Workgroup and select Change settings.
3. Select Change next to ‘To rename this computer or change its domain…’.
4. Type in the name of the Workgroup you want to join and click OK.
5. Reboot your computer for the changes to take effect.
6. Navigate to Control Panel, Network and Internet and View network computers and devices to see other machines within that Workgroup.

Homegroup is a workgroup secured with password.

Steps to configure homegroup in windows:

1. To connect a second or third computer to the homegroup, go to the first computer's control panel, then click **HomeGroup**:

2. Click the **View or print the homegroup password** to view the password. The password will appear. You may opt to print it and distribute to other people connected to your homegroup.

3. One your other Windows computers, go to Control Panel > HomeGroup and then click **Join Now**.

A1.6 use of the utilities: arp, ipconfig, tracert, nslookup.

Ans.

Arp:

The **Address Resolution Protocol** (**ARP**) is a communication protocol used for discovering the link layer address, such as a MAC address, associated with a given network layer address, typically an IPv4 address.

Ipconfig:

In computing, **ipconfig** (**i**nternet **p**rotocol **config**uration) in Microsoft Windows is a console application that displays all current TCP/IP network configuration values and can modify Dynamic Host Configuration Protocol (DHCP) and Domain Name System (DNS) settings.

Tracert:

In computing, **traceroute** is a computer network diagnostic tool for displaying the route (path) and measuring transit delays of packets across an Internet Protocol (IP) network.

Nslookup:

nslookup is a network administration command-line tool available for many computer operating systems for querying the Domain Name System (DNS) to obtain domain name or IP address mapping or for any other specific DNS record.

ASSIGNMENT 2

DATE: 3/8/18

A2.1 Examine packet flow across a network segment and see the operation of various internet protocols across the different layers in TCP/IP stack.

Ans.

TCP/IP functionality is divided into four layers, each of which include specific protocols.

- *The application layer* provides applications with standardized data exchange. Its protocols include the Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), Post Office Protocol 3 (POP3), Simple Mail Transfer Protocol (SMTP) and Simple Network Management Protocol (SNMP).
- *The transport layer* is responsible for maintaining end-to-end communications across the network. TCP handles communications between hosts and provides flow control, multiplexing and reliability. The transport protocols include TCP and User Datagram Protocol (UDP), which is sometimes used instead of TCP for special purposes.
- *The network layer*, also called the internet layer, deals with packets and connects independent networks to transport the packets across network boundaries. The network layer protocols are the IP and the Internet Control Message Protocol (ICMP), which is used for error reporting.
- *The physical layer* consists of protocols that operate only on a link -- the network component that interconnects nodes or hosts in the network. The protocols in this layer include Ethernet for local area networks (LANs) and the Address Resolution Protocol (ARP).

ASSIGNMENT 3

DATE: 10/8/18

A3.1 Use unix sockets to implement a sample client and server communication over the network.

Ans.

Client.c

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

#define KRED  "\x1B[31m"
#define KGRN  "\x1B[32m"
#define KYEL  "\x1B[33m"
#define KBLU  "\x1B[34m"
#define KMAG  "\x1B[35m"
#define KCYN  "\x1B[36m"
#define KWHT  "\x1B[37m"
#define RESET "\033[0m"

typedef enum
{
   CONNECT,
   DISCONNECT,
   GET_USERS,
   SET_USERNAME,
   PUBLIC_MESSAGE,
   PRIVATE_MESSAGE,
   TOO_FULL,
   USERNAME_ERROR,
   SUCCESS,
   ERROR
} message_type;

typedef struct
{
   message_type type;
   char username[21];
   char data[256];

} message;
```

```
typedef struct connection_info
{
   int socket;
   struct sockaddr_in address;
   char username[20];
} connection_info;

void
trim_newline (char *text)
{
   int len = strlen (text) - 1;
   if (text[len] == '\n')
     {
        text[len] = '\0';
     }
}

void
get_username (char *username)
{
   while (true)
     {
        printf ("Enter a username: ");
        fflush (stdout);
        memset (username, 0, 1000);
        fgets (username, 22, stdin);
        trim_newline (username);

        if (strlen (username) > 20)
          {

             puts ("Username must be 20 characters or less.");

          }
        else
          {
             break;
          }
     }
}

void
set_username (connection_info * connection)
{
   message msg;
   msg.type = SET_USERNAME;
   strncpy (msg.username, connection->username, 20);

   if (send (connection->socket, (void *) &msg, sizeof (msg), 0) < 0)
     {
```

```
        perror ("Send failed");
        exit (1);
     }
}

void
stop_client (connection_info * connection)
{
   close (connection->socket);
   exit (0);
}

void
connect_to_server (connection_info * connection, char *address, char *port)
{

   while (true)
    {
        get_username (connection->username);

        if ((connection->socket = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
          {
             perror ("Could not create socket");
          }

        connection->address.sin_addr.s_addr = inet_addr (address);
        connection->address.sin_family = AF_INET;
        connection->address.sin_port = htons (atoi (port));

        if (connect (connection->socket, (struct sockaddr *) &connection->address, sizeof
(connection->address)) < 0)
          {
             perror ("Connect failed.");
             exit (1);
          }

        set_username (connection);

        message msg;
        ssize_t recv_val = recv (connection->socket, &msg, sizeof (message), 0);
        if (recv_val < 0)
          {
             perror ("recv failed");
             exit (1);

          }
        else if (recv_val == 0)
          {
             close (connection->socket);
```

```
            printf ("The username \"%s\" is taken, please try another name.\n", connection->username);
            continue;
          }
        break;
      }

    puts ("Connected to server.");
    puts ("Type /help for usage.");
}

void
handle_user_input (connection_info * connection)
{
    char input[255];
    fgets (input, 255, stdin);
    trim_newline (input);

    if (strcmp (input, "/q") == 0 || strcmp (input, "/quit") == 0)
      {
        stop_client (connection);
      }
    else if (strcmp (input, "/l") == 0 || strcmp (input, "/list") == 0)
      {
        message msg;
        msg.type = GET_USERS;

        if (send (connection->socket, &msg, sizeof (message), 0) < 0)
          {
            perror ("Send failed");
            exit (1);
          }
      }
    else if (strcmp (input, "/h") == 0 || strcmp (input, "/help") == 0)
      {
        puts ("/quit or /q: Exit the program.");
        puts ("/help or /h: Displays help information.");
        puts ("/list or /l: Displays list of users in chatroom.");
        puts ("@<username> <message> Send a private message to given username.");
      }
    else if (strncmp (input, "@", 1) == 0)
      {
        message msg;
        msg.type = PRIVATE_MESSAGE;

        char *toUsername, *chatMsg;

        toUsername = strtok (input + 1, " ");

        if (toUsername == NULL)
```

```
        {
            puts (KRED "The format for private messages is: @<username> <message>"
RESET);
            return;
        }

      if (strlen (toUsername) == 0)
        {
            puts (KRED "You must enter a username for a private message." RESET);
            return;
        }

      if (strlen (toUsername) > 20)
        {
            puts (KRED "The username must be between 1 and 20 characters." RESET);
            return;
        }

      chatMsg = strtok (NULL, "");

      if (chatMsg == NULL)
        {
            puts (KRED "You must enter a message to send to the specified user." RESET);
            return;
        }

      strncpy (msg.username, toUsername, 20);
      strncpy (msg.data, chatMsg, 255);

      if (send (connection->socket, &msg, sizeof (message), 0) < 0)
        {
            perror ("Send failed");
            exit (1);
        }

    }
  else
    {
      message msg;
      msg.type = PUBLIC_MESSAGE;
      strncpy (msg.username, connection->username, 20);

      if (strlen (input) == 0)
        {
            return;
        }

      strncpy (msg.data, input, 255);

      if (send (connection->socket, &msg, sizeof (message), 0) < 0)
```

```
            {
               perror ("Send failed");
               exit (1);
            }
      }

}

void
handle_server_message (connection_info * connection)
{
   message msg;

   ssize_t recv_val = recv (connection->socket, &msg, sizeof (message), 0);
   if (recv_val < 0)
     {
        perror ("recv failed");
        exit (1);

     }
   else if (recv_val == 0)
     {
        close (connection->socket);
        puts ("Server disconnected.");
        exit (0);
     }

   switch (msg.type)
     {

     case CONNECT:
        printf (KYEL "%s has connected." RESET "\n", msg.username);
        break;

     case DISCONNECT:
        printf (KYEL "%s has disconnected." RESET "\n", msg.username);
        break;

     case GET_USERS:
        printf (KMAG "%s" RESET "\n", msg.data);
        break;

     case PUBLIC_MESSAGE:
        printf (KGRN "%s" RESET ": %s\n", msg.username, msg.data);
        break;

     case PRIVATE_MESSAGE:
        printf (KWHT "From %s:" KCYN " %s\n" RESET, msg.username, msg.data);
        break;
```

```
    case TOO_FULL:
      fprintf (stderr, KRED "Server chatroom is too full to accept new clients." RESET "\n");
      exit (0);
      break;

    default:
      fprintf (stderr, KRED "Unknown message type received." RESET "\n");
      break;
    }
}

int
main (int argc, char *argv[])
{
  connection_info connection;
  fd_set file_descriptors;

  if (argc != 3)
    {
      fprintf (stderr, "Usage: %s <IP> <port>\n", argv[0]);
      exit (1);
    }

  connect_to_server (&connection, argv[1], argv[2]);

  while (true)
    {
      FD_ZERO (&file_descriptors);
      FD_SET (STDIN_FILENO, &file_descriptors);
      FD_SET (connection.socket, &file_descriptors);
      fflush (stdin);

      if (select (connection.socket + 1, &file_descriptors, NULL, NULL, NULL) < 0)
        {
          perror ("Select failed.");
          exit (1);
        }

      if (FD_ISSET (STDIN_FILENO, &file_descriptors))
        {
          handle_user_input (&connection);
        }

      if (FD_ISSET (connection.socket, &file_descriptors))
        {
          handle_server_message (&connection);
        }
    }

  close (connection.socket);
```

```
    return 0;
}
```

Server.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdbool.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <pthread.h>

#define MAX_CLIENTS 10

typedef enum
{
    CONNECT,
    DISCONNECT,
    GET_USERS,
    SET_USERNAME,
    PUBLIC_MESSAGE,
    PRIVATE_MESSAGE,
    TOO_FULL,
    USERNAME_ERROR,
    SUCCESS,
    ERROR
} message_type;

typedef struct
{
    message_type type;
    char username[21];
    char data[256];

} message;

typedef struct connection_info
{
    int socket;
```

```
    struct sockaddr_in address;
    char username[20];
} connection_info;

void
trim_newline (char *text)
{
    int len = strlen (text) - 1;
    if (text[len] == '\n')
      {
        text[len] = '\0';
      }
}

void
initialize_server (connection_info * server_info, int port)
{
    if ((server_info->socket = socket (AF_INET, SOCK_STREAM, 0)) < 0)
      {
        perror ("Failed to create socket");
        exit (1);
      }

    server_info->address.sin_family = AF_INET;
    server_info->address.sin_addr.s_addr = INADDR_ANY;
    server_info->address.sin_port = htons (port);

    if (bind (server_info->socket, (struct sockaddr *) &server_info->address,
sizeof (server_info->address)) < 0)
      {
        perror ("Binding failed");
        exit (1);
      }

    const int optVal = 1;
    const socklen_t optLen = sizeof (optVal);
    if (setsockopt (server_info->socket, SOL_SOCKET, SO_REUSEADDR,
(void *) &optVal, optLen) < 0)
      {
        perror ("Set socket option failed");
        exit (1);
      }
```

```
    if (listen (server_info->socket, 3) < 0)
      {
         perror ("Listen failed");
         exit (1);
      }

    printf ("Waiting for incoming connections...\n");
}

void
send_public_message (connection_info clients[], int sender, char
*message_text)
{
    message msg;
    msg.type = PUBLIC_MESSAGE;
    strncpy (msg.username, clients[sender].username, 20);
    strncpy (msg.data, message_text, 256);
    int i = 0;
    for (i = 0; i < MAX_CLIENTS; i++)
      {
         if (i != sender && clients[i].socket != 0)
           {
              if (send (clients[i].socket, &msg, sizeof (msg), 0) < 0)
                {
                   perror ("Send failed");
                   exit (1);
                }
           }
      }
}

void
send_private_message (connection_info clients[], int sender, char *username,
char *message_text)
{
    message msg;
    msg.type = PRIVATE_MESSAGE;
    strncpy (msg.username, clients[sender].username, 20);
    strncpy (msg.data, message_text, 256);

    int i;
    for (i = 0; i < MAX_CLIENTS; i++)
      {
```

```
        if (i != sender && clients[i].socket != 0 && strcmp (clients[i].username,
username) == 0)
          {
            if (send (clients[i].socket, &msg, sizeof (msg), 0) < 0)
              {
                perror ("Send failed");
                exit (1);
              }
            return;
          }
      }

    msg.type = USERNAME_ERROR;
    sprintf (msg.data, "Username \"%s\" does not exist or is not logged in.",
username);

    if (send (clients[sender].socket, &msg, sizeof (msg), 0) < 0)
      {
        perror ("Send failed");
        exit (1);
      }

}

void
send_connect_message (connection_info * clients, int sender)
{
    message msg;
    msg.type = CONNECT;
    strncpy (msg.username, clients[sender].username, 21);
    int i = 0;
    for (i = 0; i < MAX_CLIENTS; i++)
      {
        if (clients[i].socket != 0)
          {
            if (i == sender)
              {
                msg.type = SUCCESS;
                if (send (clients[i].socket, &msg, sizeof (msg), 0) < 0)
                  {
                    perror ("Send failed");
                    exit (1);
                  }
```

```
            }
          else
            {
              if (send (clients[i].socket, &msg, sizeof (msg), 0) < 0)
                {
                  perror ("Send failed");
                  exit (1);
                }
            }
        }
    }
}

void
send_disconnect_message (connection_info * clients, char *username)
{
  message msg;
  msg.type = DISCONNECT;
  strncpy (msg.username, username, 21);
  int i = 0;
  for (i = 0; i < MAX_CLIENTS; i++)
    {
      if (clients[i].socket != 0)
        {
          if (send (clients[i].socket, &msg, sizeof (msg), 0) < 0)
            {
              perror ("Send failed");
              exit (1);
            }
        }
    }
}

void
send_user_list (connection_info * clients, int receiver)
{
  message msg;
  msg.type = GET_USERS;
  char *list = msg.data;

  int i;
  for (i = 0; i < MAX_CLIENTS; i++)
    {
```

```
        if (clients[i].socket != 0)
          {
            list = stpcpy (list, clients[i].username);
            list = stpcpy (list, "\n");
          }
      }

    if (send (clients[receiver].socket, &msg, sizeof (msg), 0) < 0)
      {
        perror ("Send failed");
        exit (1);
      }

}

void
send_too_full_message (int socket)
{
    message too_full_message;
    too_full_message.type = TOO_FULL;

    if (send (socket, &too_full_message, sizeof (too_full_message), 0) < 0)
      {
        perror ("Send failed");
        exit (1);
      }

    close (socket);
}

void
stop_server (connection_info connection[])
{
    int i;
    for (i = 0; i < MAX_CLIENTS; i++)
      {
        close (connection[i].socket);
      }
    exit (0);
}

void
handle_client_message (connection_info clients[], int sender)
```

```
{
   int read_size;
   message msg;

   if ((read_size = recv (clients[sender].socket, &msg, sizeof (message), 0)) ==
0)
     {
        printf ("User disconnected: %s.\n", clients[sender].username);
        close (clients[sender].socket);
        clients[sender].socket = 0;
        send_disconnect_message (clients, clients[sender].username);

     }
   else
     {

        switch (msg.type)
          {
          case GET_USERS:
             send_user_list (clients, sender);
             break;

          case SET_USERNAME:;
             int i;
             for (i = 0; i < MAX_CLIENTS; i++)
               {
                  if (clients[i].socket != 0 && strcmp (clients[i].username,
msg.username) == 0)
                    {
                       close (clients[sender].socket);
                       clients[sender].socket = 0;
                       return;
                    }
               }

             strcpy (clients[sender].username, msg.username);
             printf ("User connected: %s\n", clients[sender].username);
             send_connect_message (clients, sender);
             break;

          case PUBLIC_MESSAGE:
             send_public_message (clients, sender, msg.data);
             break;
```

```
            case PRIVATE_MESSAGE:
               send_private_message (clients, sender, msg.username, msg.data);
               break;

            default:
               fprintf (stderr, "Unknown message type received.\n");
               break;
            }
      }
}

int
construct_fd_set (fd_set * set, connection_info * server_info, connection_info
clients[])
{
   FD_ZERO (set);
   FD_SET (STDIN_FILENO, set);
   FD_SET (server_info->socket, set);

   int max_fd = server_info->socket;
   int i;
   for (i = 0; i < MAX_CLIENTS; i++)
     {
        if (clients[i].socket > 0)
          {
             FD_SET (clients[i].socket, set);
             if (clients[i].socket > max_fd)
               {
                  max_fd = clients[i].socket;
               }
          }
     }
   return max_fd;
}

void
handle_new_connection (connection_info * server_info, connection_info
clients[])
{
   int new_socket;
   int address_len;
```

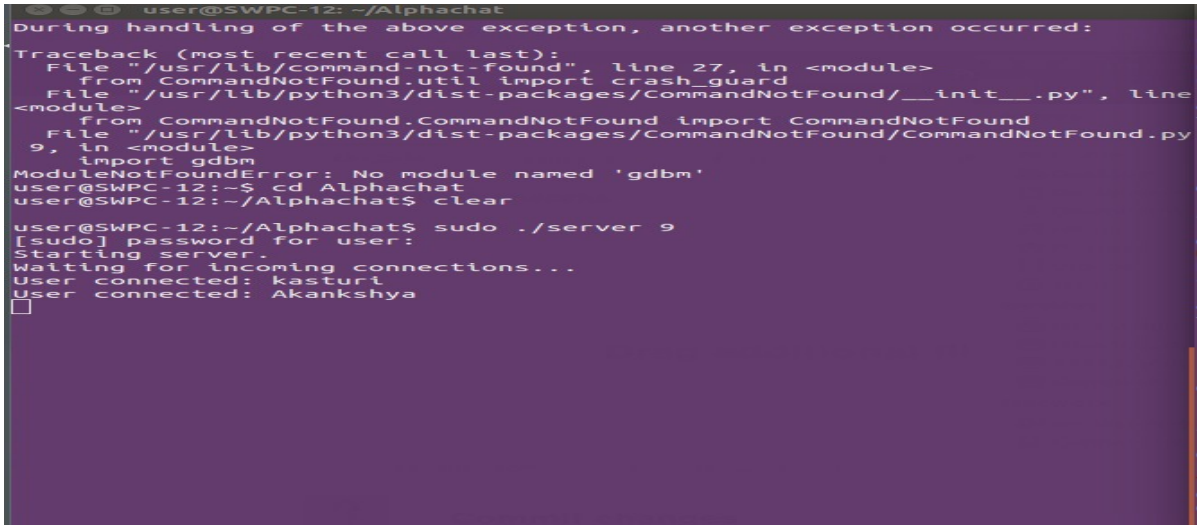
```
   new_socket = accept (server_info->socket, (struct sockaddr *) &server_info-
>address, (socklen_t *) & address_len);

   if (new_socket < 0)
     {
        perror ("Accept Failed");
        exit (1);
     }

   int i;
   for (i = 0; i < MAX_CLIENTS; i++)
     {
        if (clients[i].socket == 0)
          {
             clients[i].socket = new_socket;
             break;

          }
        else if (i == MAX_CLIENTS - 1)
          {
             send_too_full_message (new_socket);
          }
     }
}

void
handle_user_input (connection_info clients[])
{
   char input[255];
   fgets (input, sizeof (input), stdin);
   trim_newline (input);

   if (input[0] == 'q')
     {
        stop_server (clients);
     }
}

int
main (int argc, char *argv[])
{
   puts ("Starting server.");
```

```
  fd_set file_descriptors;

  connection_info server_info;
  connection_info clients[MAX_CLIENTS];

  int i;
  for (i = 0; i < MAX_CLIENTS; i++)
    {
      clients[i].socket = 0;
    }

  if (argc != 2)
    {
      fprintf (stderr, "Usage: %s <port>\n", argv[0]);
      exit (1);
    }

  initialize_server (&server_info, atoi (argv[1]));

  while (true)
    {
      int max_fd = construct_fd_set (&file_descriptors, &server_info, clients);

      if (select (max_fd + 1, &file_descriptors, NULL, NULL, NULL) < 0)
        {
          perror ("Select Failed");
          stop_server (clients);
        }

      if (FD_ISSET (STDIN_FILENO, &file_descriptors))
        {
          handle_user_input (clients);
        }

      if (FD_ISSET (server_info.socket, &file_descriptors))
        {
          handle_new_connection (&server_info, clients);
        }

      for (i = 0; i < MAX_CLIENTS; i++)
        {
          if (clients[i].socket > 0 && FD_ISSET (clients[i].socket,
&file_descriptors))
```

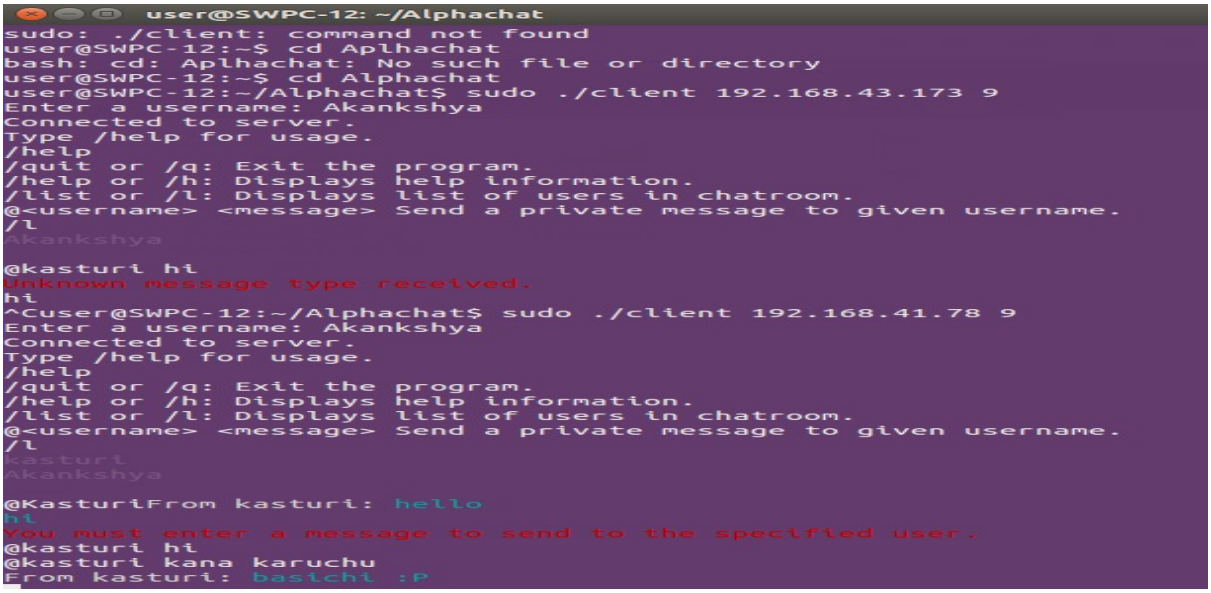
```
            {
               handle_client_message (clients, i);
            }
         }
      }

   return 0;
}
```

Output:

Server:



Client:



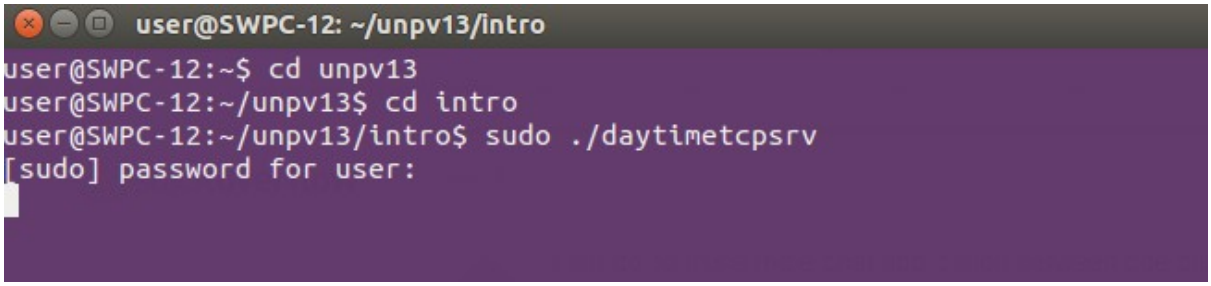
A3.2 Write a program to implement daytime server that responds with day and time to request sent by client.

Ans.

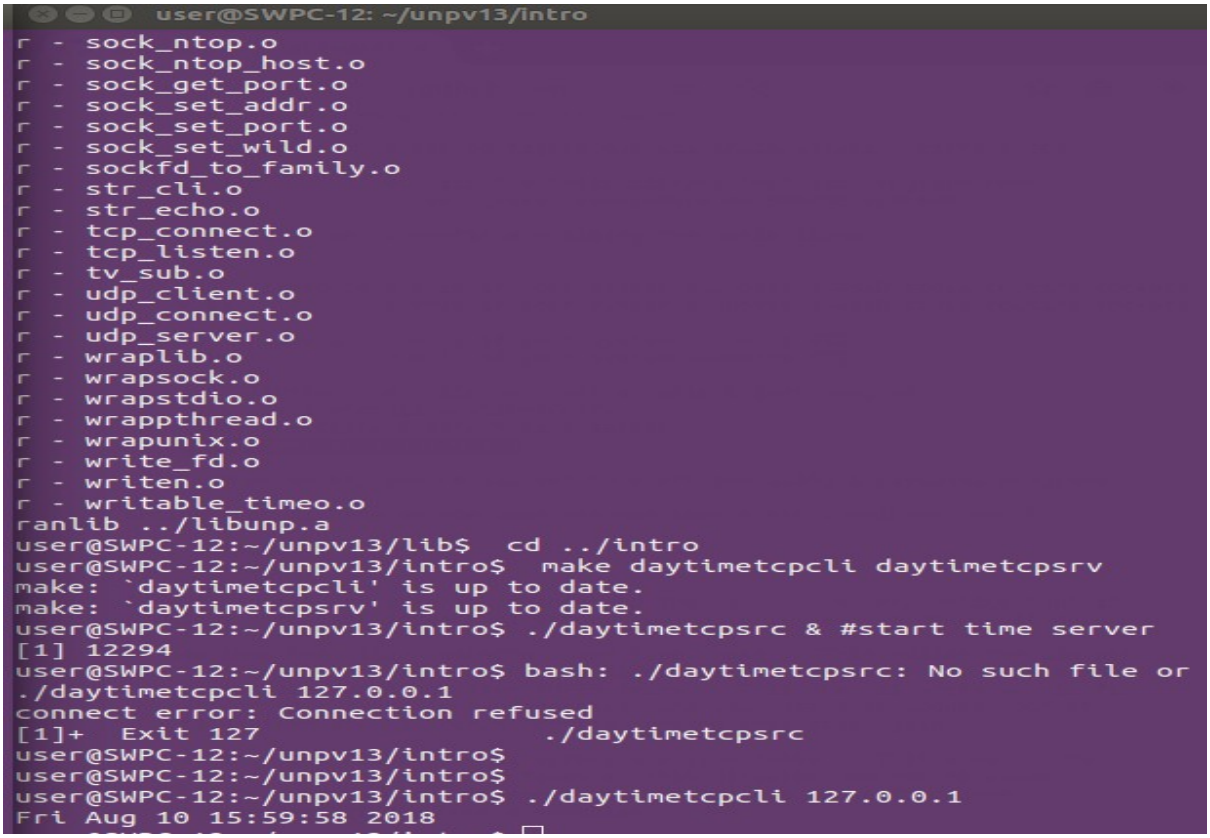
```
#include "./unp.h"
#include<stdio.h>

int main(int argc,char **argv)
{
        //client part
        int sockfd,n;
        char recvline[MAXLINE+1];
        struct sockaddr_in servaddr;

        if(argc !=2){printf("lipsa");}
                //err_quit("usage: a.out <IPaddress>");
        if(sockfd=socket(AF_INET,SOCK_STREAM,0))<0){}
                //err_sys("socket error")
        bzero(&servaddr,sizeof(servaddr))
        //same as memset set to zero

        servaddr.sin_family=AF_INET;
        servaddr.sin_port=htons(13); /*port no 13 is resereved for daytime server*/
        //converts port address into binary format
        if(inet_pton(AF_INET,argv[1],&servaddr.sin_addr)<=0){}
        //pton : presentation to network conversion of 127.0.0.1 passed as string to numeric
format
        //err_quit("inet_pton error for %s",argv[1]);

        if(connect(sockfd,(SA *) &servaddr,sizeof(servaddr))<0){}
                //err_sys("connect error");
        while((n=read(sockfd,recvline,MAXLINE))>0){
        recvline[n]=0;
        if(fputs(recvline,stdout)==EDF){}
                //err_sys("fputs error")
        }
        if(n<0)
        {
                //err_sys("read error");
        }
        exit(0);
}
```

Output:

Server:



Client:



A3.3 Implement a TCP client server application to transfer a file.

Ans.

filetransferserver.c

```
#include<stdio.h>
#include<sys/types.h>
#include<string.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<unistd.h>
#define SA struct sockaddr
#define LISTENQ 5
int main(int argc,char**argv)
{
int fd,sockfd,listenfd,connfd;
pid_t childpid;
socklen_t client;
struct sockaddr_in servaddr,cliaddr;
listenfd=socket(AF_INET,SOCK_STREAM,0);
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr.sin_port=htons(atoi(argv[1]));
bind(listenfd,(SA*)&servaddr,sizeof(servaddr));
listen(listenfd,LISTENQ);
client=sizeof(cliaddr);
connfd=accept(listenfd,(SA*)&cliaddr,&client);
char buffer[100];
```

```
FILE *fp;
read(connfd,buffer,100);
fp=fopen("add1.txt","w");
fprintf(fp,"%s",buffer);
printf("the file was received successfully");
printf("the new file created is add1.txt");
}
```

filetransferclient.c

```
#include<arpa/inet.h>
#include<unistd.h>
#define SA struct sockaddr
int main(int argc,char**argv)
{
int sockfd;
char fname[25];
int len;
struct sockaddr_in servaddr,cliaddr;
sockfd=socket(AF_INET,SOCK_STREAM,0);
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr.sin_port=htons(atoi(argv[1]));
inet_pton(AF_INET,argv[1],&servaddr.sin_addr);
connect(sockfd,(SA*)&servaddr,sizeof(servaddr));
char buffer[100];
```

```
FILE *f;

f=fopen("add.txt","r");

fscanf(f,"%s",buffer);

write(sockfd,buffer,100);

printf("the file was sent successfully");

}
```