Computer Networks Laboratory CSN361

Lab Assignment 3

Ashutosh Chaubey (17114015)

Problem Statement 1

Write a socket program in C to determine class, Network and Host ID of an IPv4 address.

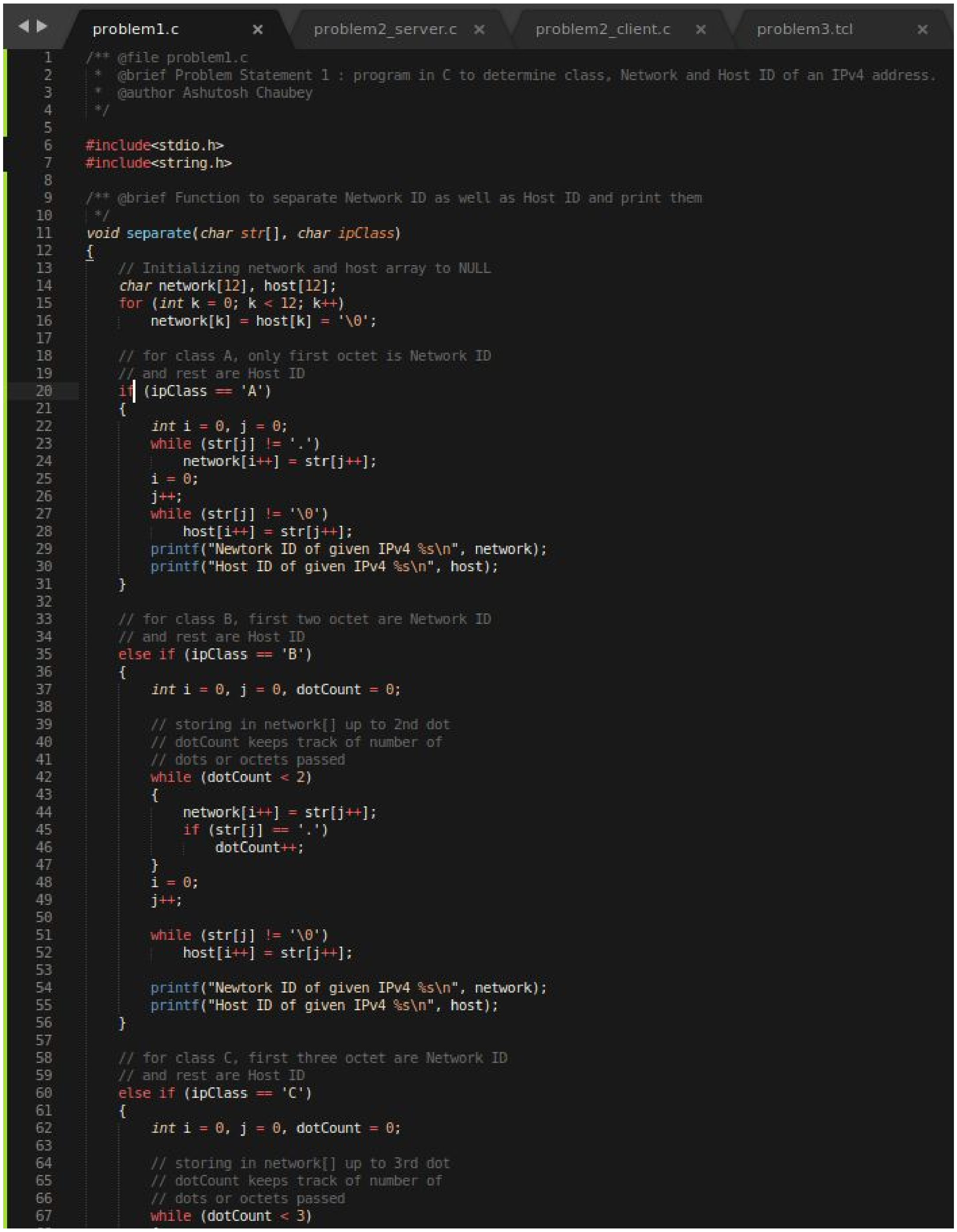
Algorithm:

- Extract the first octet
- Decide the class based on the value of the first octet
- Decide the number of octets in Network and Host ID based on the class

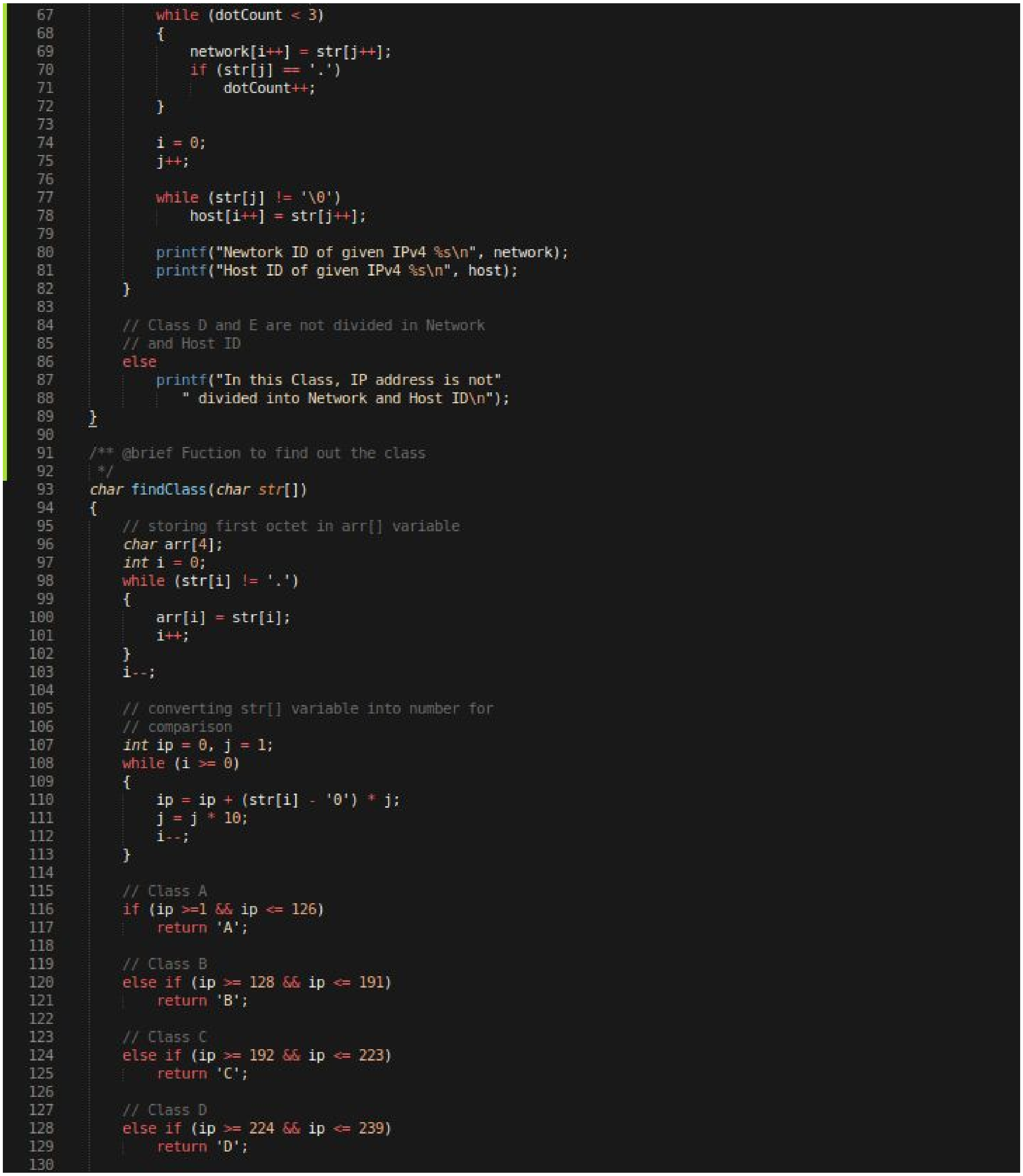
Data Structures:

Char array to store the ip address

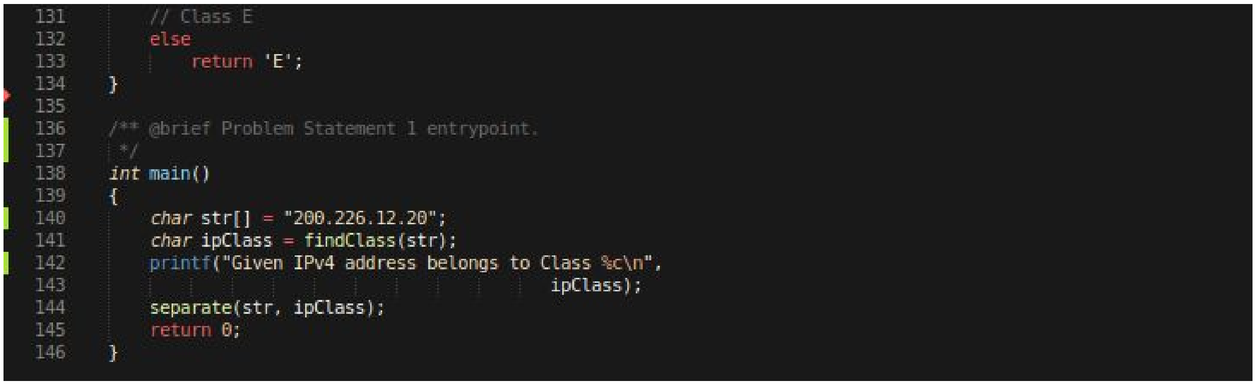
Code:



```c
/** @file problem1.c
 *  @brief Problem Statement 1 : program in C to determine class, Network and Host ID of an IPv4 address.
 *  @author Ashutosh Chaubey
 */

#include<stdio.h>
#include<string.h>

/** @brief Function to separate Network ID as well as Host ID and print them
 */
void separate(char str[], char ipClass)
{
    // Initializing network and host array to NULL
    char network[12], host[12];
    for (int k = 0; k < 12; k++)
        network[k] = host[k] = '\0';

    // for class A, only first octet is Network ID
    // and rest are Host ID
    if (ipClass == 'A')
    {
        int i = 0, j = 0;
        while (str[j] != '.')
            network[i++] = str[j++];
        i = 0;
        j++;
        while (str[j] != '\0')
            host[i++] = str[j++];
        printf("Newtork ID of given IPv4 %s\n", network);
        printf("Host ID of given IPv4 %s\n", host);
    }

    // for class B, first two octet are Network ID
    // and rest are Host ID
    else if (ipClass == 'B')
    {
        int i = 0, j = 0, dotCount = 0;

        // storing in network[] up to 2nd dot
        // dotCount keeps track of number of
        // dots or octets passed
        while (dotCount < 2)
        {
            network[i++] = str[j++];
            if (str[j] == '.')
                dotCount++;
        }
        i = 0;
        j++;

        while (str[j] != '\0')
            host[i++] = str[j++];

        printf("Newtork ID of given IPv4 %s\n", network);
        printf("Host ID of given IPv4 %s\n", host);
    }

    // for class C, first three octet are Network ID
    // and rest are Host ID
    else if (ipClass == 'C')
    {
        int i = 0, j = 0, dotCount = 0;

        // storing in network[] up to 3rd dot
        // dotCount keeps track of number of
        // dots or octets passed
        while (dotCount < 3)
```

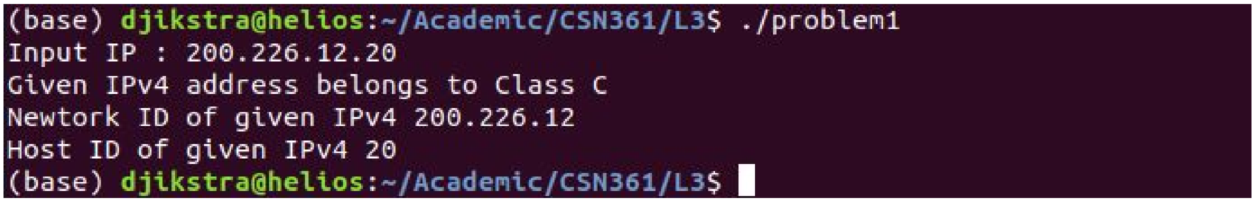


```c
        while (dotCount < 3)
        {
            network[i++] = str[j++];
            if (str[j] == '.')
                dotCount++;
        }

        i = 0;
        j++;

        while (str[j] != '\0')
            host[i++] = str[j++];

        printf("Newtork ID of given IPv4 %s\n", network);
        printf("Host ID of given IPv4 %s\n", host);
    }

    // Class D and E are not divided in Network
    // and Host ID
    else
        printf("In this Class, IP address is not"
           " divided into Network and Host ID\n");
}

/** @brief Fuction to find out the class
 */
char findClass(char str[])
{
    // storing first octet in arr[] variable
    char arr[4];
    int i = 0;
    while (str[i] != '.')
    {
        arr[i] = str[i];
        i++;
    }
    i--;

    // converting str[] variable into number for
    // comparison
    int ip = 0, j = 1;
    while (i >= 0)
    {
        ip = ip + (str[i] - '0') * j;
        j = j * 10;
        i--;
    }

    // Class A
    if (ip >=1 && ip <= 126)
        return 'A';

    // Class B
    else if (ip >= 128 && ip <= 191)
        return 'B';

    // Class C
    else if (ip >= 192 && ip <= 223)
        return 'C';

    // Class D
    else if (ip >= 224 && ip <= 239)
        return 'D';

```



```c
    // Class E
    else
        return 'E';
}

/** @brief Problem Statement 1 entrypoint.
 */
int main()
{
    char str[] = "200.226.12.20";
    char ipClass = findClass(str);
    printf("Given IPv4 address belongs to Class %c\n",
                                            ipClass);
    separate(str, ipClass);
    return 0;
}
```

Output



```
(base) djikstra@helios:~/Academic/CSN361/L3$ ./problem1
Input IP : 200.226.12.20
Given IPv4 address belongs to Class C
Newtork ID of given IPv4 200.226.12
Host ID of given IPv4 20
(base) djikstra@helios:~/Academic/CSN361/L3$ 
```

Problem Statement 2

Write a C program to demonstrate File Transfer using UDP.

Algorithm:

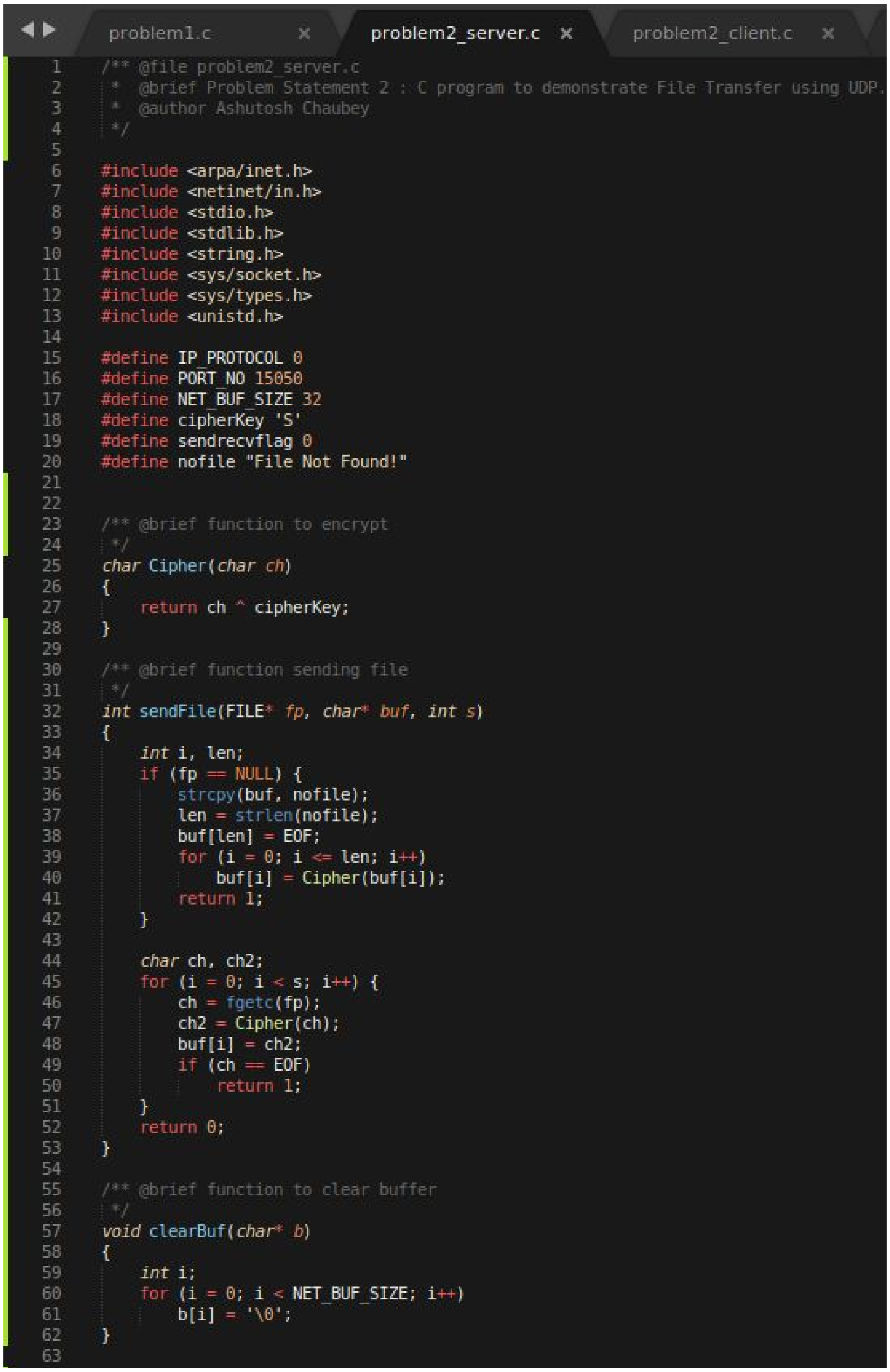
- Create a socket for the server and the client
- Make the socket ready to receive file name to be read
- Send file name from client

Data Structures:

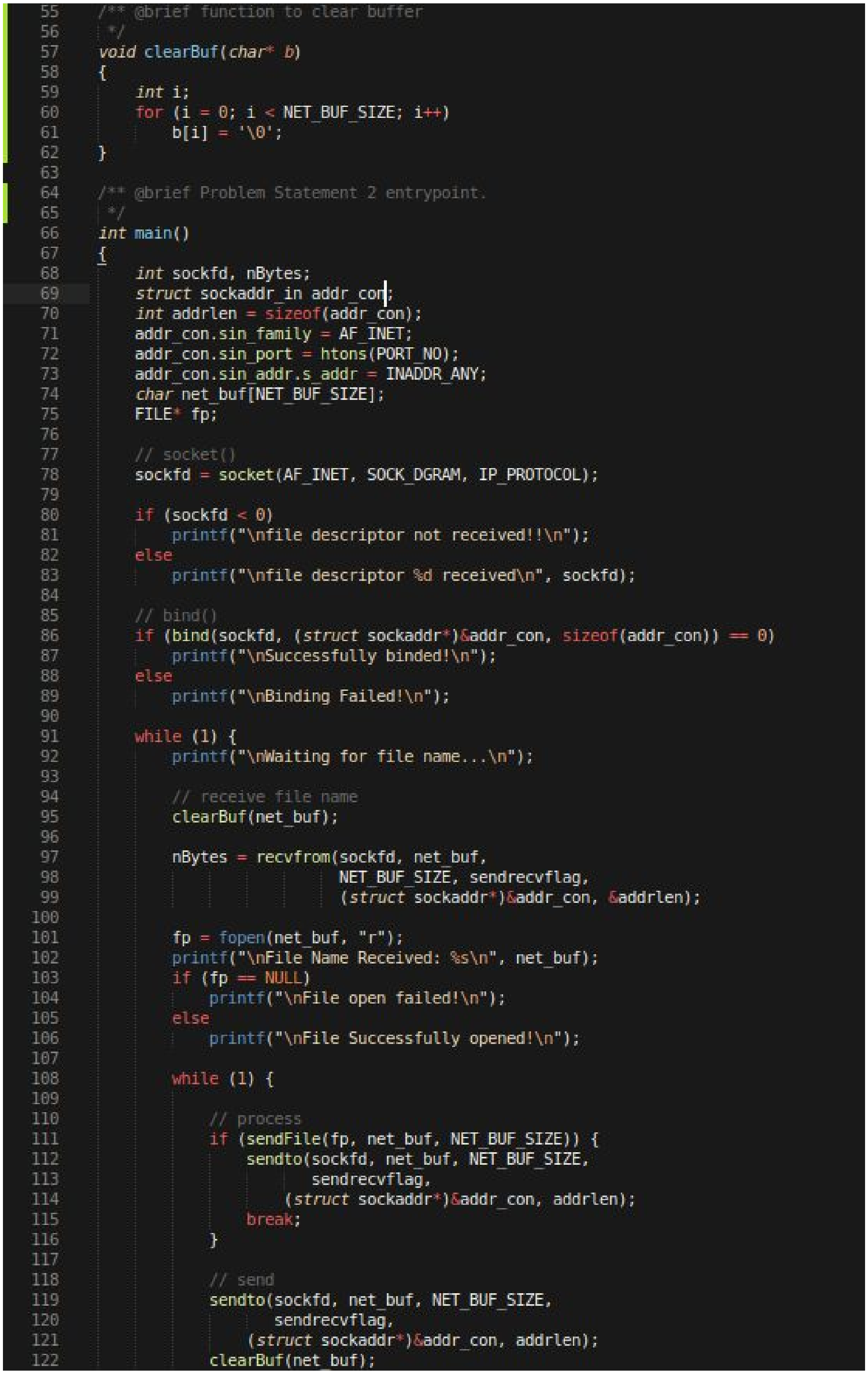
- Int fd: File descriptor
- Sockaddr_in : TO store the info about address

Code:

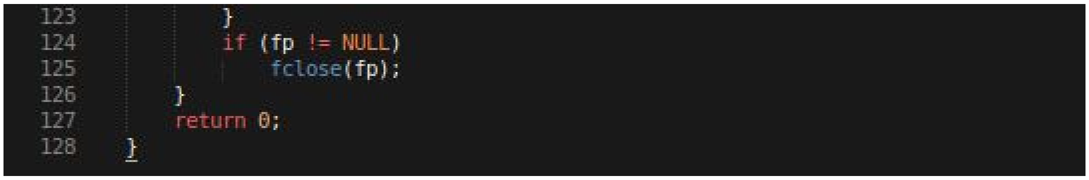
Server -



```c
/** @file problem2_server.c
 *  @brief Problem Statement 2 : C program to demonstrate File Transfer using UDP.
 *  @author Ashutosh Chaubey
 */

#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define IP_PROTOCOL 0
#define PORT_NO 15050
#define NET_BUF_SIZE 32
#define cipherKey 'S'
#define sendrecvflag 0
#define nofile "File Not Found!"


/** @brief function to encrypt
 */
char Cipher(char ch)
{
    return ch ^ cipherKey;
}

/** @brief function sending file
 */
int sendFile(FILE* fp, char* buf, int s)
{
    int i, len;
    if (fp == NULL) {
        strcpy(buf, nofile);
        len = strlen(nofile);
        buf[len] = EOF;
        for (i = 0; i <= len; i++)
            buf[i] = Cipher(buf[i]);
        return 1;
    }

    char ch, ch2;
    for (i = 0; i < s; i++) {
        ch = fgetc(fp);
        ch2 = Cipher(ch);
        buf[i] = ch2;
        if (ch == EOF)
            return 1;
    }
    return 0;
}

/** @brief function to clear buffer
 */
void clearBuf(char* b)
{
    int i;
    for (i = 0; i < NET_BUF_SIZE; i++)
        b[i] = '\0';
}

```

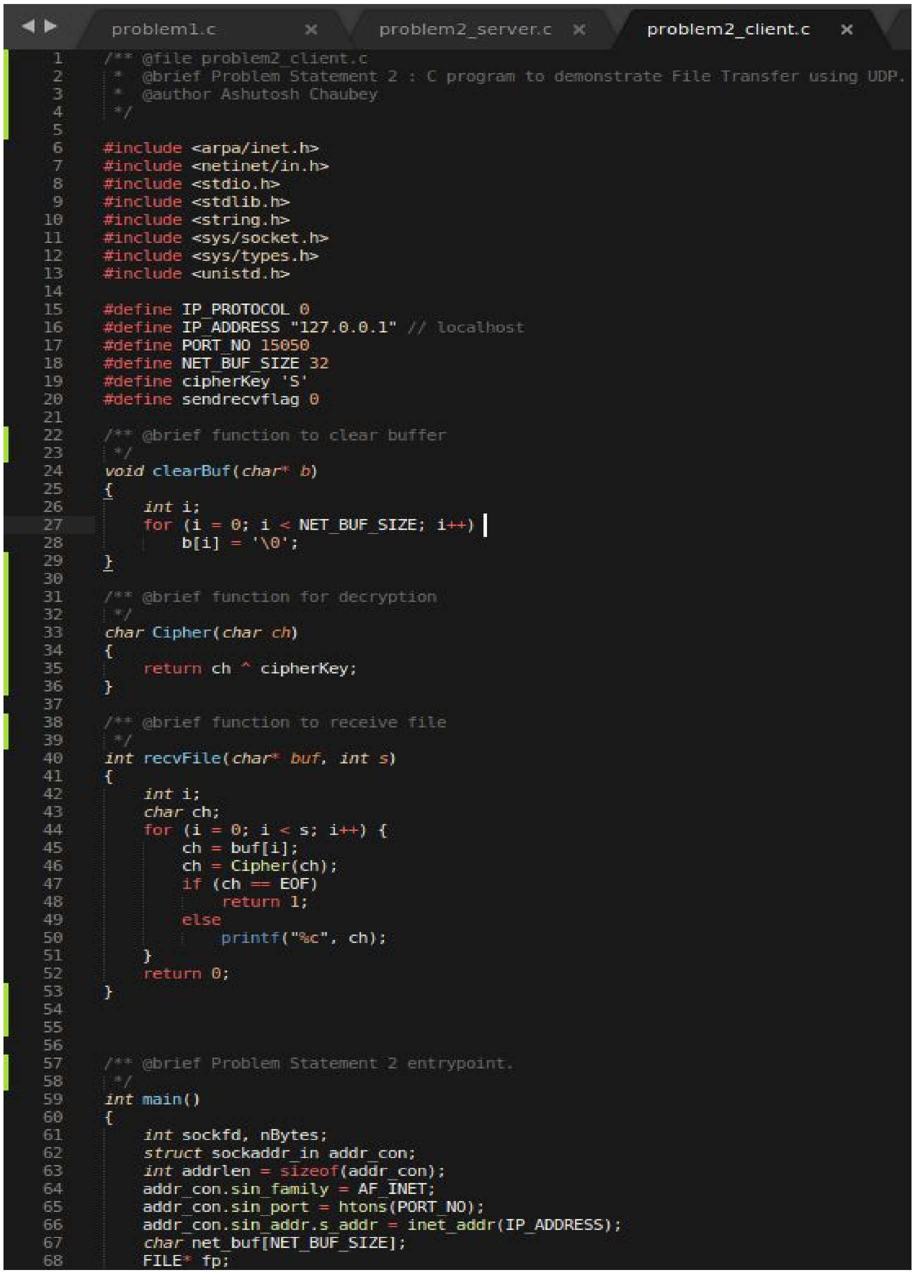


```
/** @brief function to clear buffer
 */
void clearBuf(char* b)
{
    int i;
    for (i = 0; i < NET_BUF_SIZE; i++)
        b[i] = '\0';
}

/** @brief Problem Statement 2 entrypoint.
 */
int main()
{
    int sockfd, nBytes;
    struct sockaddr_in addr_con;
    int addrlen = sizeof(addr_con);
    addr_con.sin_family = AF_INET;
    addr_con.sin_port = htons(PORT_NO);
    addr_con.sin_addr.s_addr = INADDR_ANY;
    char net_buf[NET_BUF_SIZE];
    FILE* fp;

    // socket()
    sockfd = socket(AF_INET, SOCK_DGRAM, IP_PROTOCOL);

    if (sockfd < 0)
        printf("\nfile descriptor not received!!\n");
    else
        printf("\nfile descriptor %d received\n", sockfd);

    // bind()
    if (bind(sockfd, (struct sockaddr*)&addr_con, sizeof(addr_con)) == 0)
        printf("\nSuccessfully binded!\n");
    else
        printf("\nBinding Failed!\n");

    while (1) {
        printf("\nWaiting for file name...\n");

        // receive file name
        clearBuf(net_buf);

        nBytes = recvfrom(sockfd, net_buf,
                          NET_BUF_SIZE, sendrecvflag,
                          (struct sockaddr*)&addr_con, &addrlen);

        fp = fopen(net_buf, "r");
        printf("\nFile Name Received: %s\n", net_buf);
        if (fp == NULL)
            printf("\nFile open failed!\n");
        else
            printf("\nFile Successfully opened!\n");

        while (1) {

            // process
            if (sendFile(fp, net_buf, NET_BUF_SIZE)) {
                sendto(sockfd, net_buf, NET_BUF_SIZE,
                       sendrecvflag,
                    (struct sockaddr*)&addr_con, addrlen);
                break;
            }

            // send
            sendto(sockfd, net_buf, NET_BUF_SIZE,
                   sendrecvflag,
                (struct sockaddr*)&addr_con, addrlen);
            clearBuf(net_buf);
```

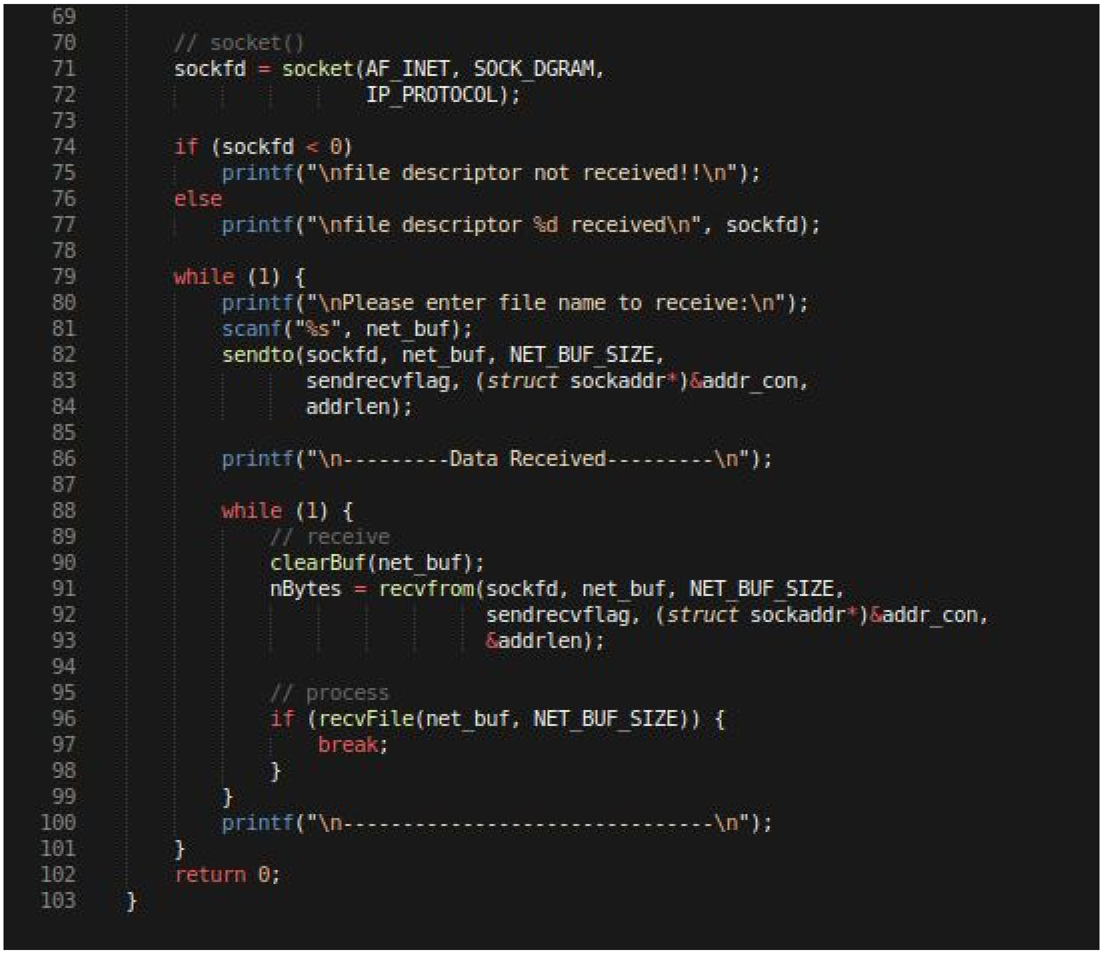


```
        }
        if (fp != NULL)
            fclose(fp);
    }
    return 0;
}
```

Client -

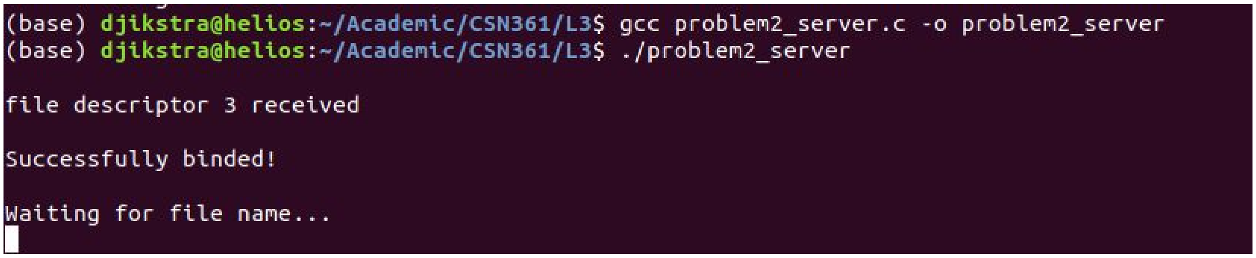


```
/** @file problem2_client.c
 *  @brief Problem Statement 2 : C program to demonstrate File Transfer using UDP.
 *  @author Ashutosh Chaubey
 */

#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define IP_PROTOCOL 0
#define IP_ADDRESS "127.0.0.1" // localhost
#define PORT_NO 15050
#define NET_BUF_SIZE 32
#define cipherKey 'S'
#define sendrecvflag 0

/** @brief function to clear buffer
 */
void clearBuf(char* b)
{
    int i;
    for (i = 0; i < NET_BUF_SIZE; i++)
        b[i] = '\0';
}

/** @brief function for decryption
 */
char Cipher(char ch)
{
    return ch ^ cipherKey;
}

/** @brief function to receive file
 */
int recvFile(char* buf, int s)
{
    int i;
    char ch;
    for (i = 0; i < s; i++) {
        ch = buf[i];
        ch = Cipher(ch);
        if (ch == EOF)
            return 1;
        else
            printf("%c", ch);
    }
    return 0;
}



/** @brief Problem Statement 2 entrypoint.
 */
int main()
{
    int sockfd, nBytes;
    struct sockaddr_in addr_con;
    int addrlen = sizeof(addr_con);
    addr_con.sin_family = AF_INET;
    addr_con.sin_port = htons(PORT_NO);
    addr_con.sin_addr.s_addr = inet_addr(IP_ADDRESS);
    char net_buf[NET_BUF_SIZE];
    FILE* fp;
```

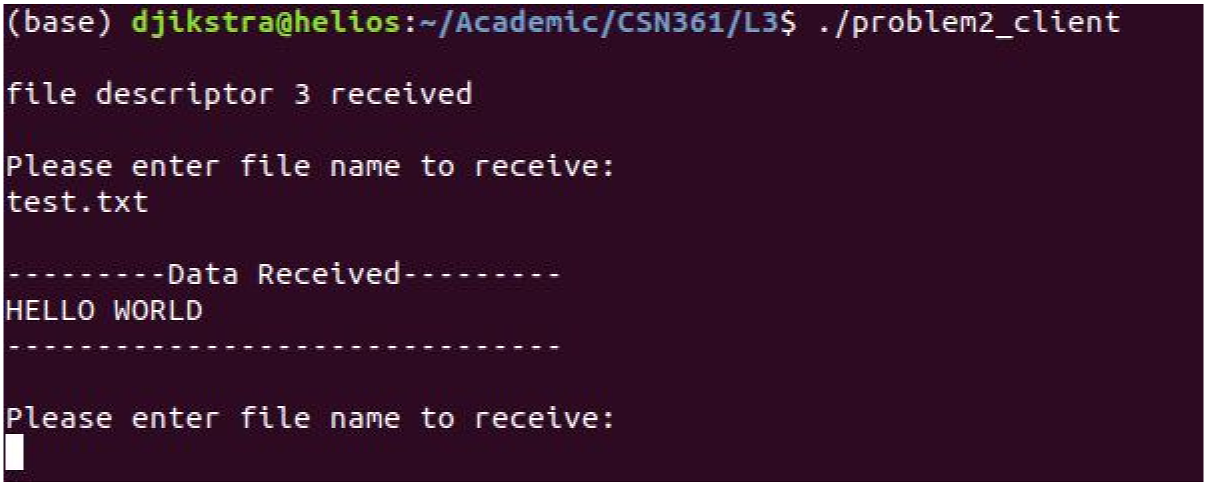


```c

    // socket()
    sockfd = socket(AF_INET, SOCK_DGRAM,
                    IP_PROTOCOL);

    if (sockfd < 0)
        printf("\nfile descriptor not received!!\n");
    else
        printf("\nfile descriptor %d received\n", sockfd);

    while (1) {
        printf("\nPlease enter file name to receive:\n");
        scanf("%s", net_buf);
        sendto(sockfd, net_buf, NET_BUF_SIZE,
               sendrecvflag, (struct sockaddr*)&addr_con,
               addrlen);

        printf("\n---------Data Received---------\n");

        while (1) {
            // receive
            clearBuf(net_buf);
            nBytes = recvfrom(sockfd, net_buf, NET_BUF_SIZE,
                              sendrecvflag, (struct sockaddr*)&addr_con,
                              &addrlen);

            // process
            if (recvFile(net_buf, NET_BUF_SIZE)) {
                break;
            }
        }
        printf("\n-------------------------------\n");
    }
    return 0;
}
```

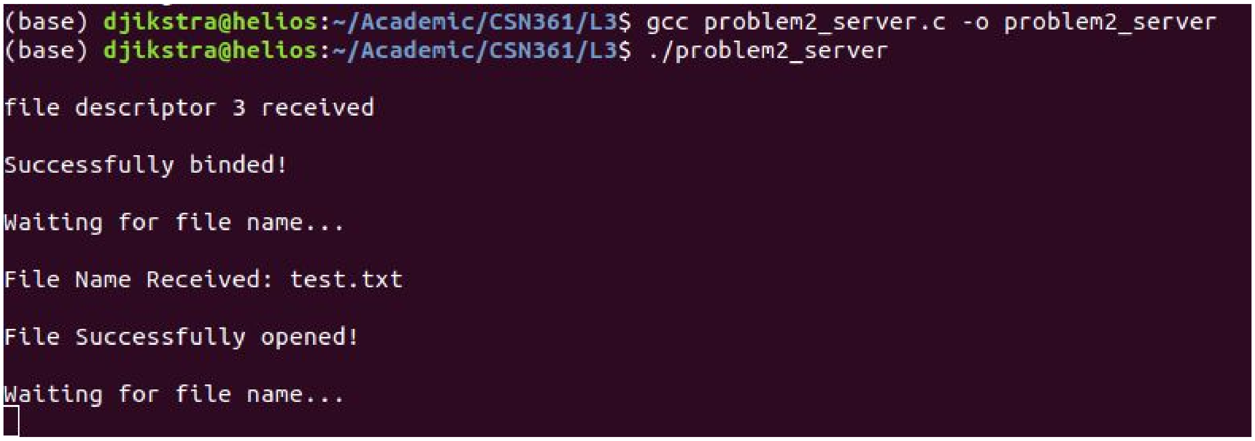
Output



```
(base) djikstra@helios:~/Academic/CSN361/L3$ gcc problem2_server.c -o problem2_server
(base) djikstra@helios:~/Academic/CSN361/L3$ ./problem2_server

file descriptor 3 received

Successfully binded!

Waiting for file name...
```



```
(base) djikstra@helios:~/Academic/CSN361/L3$ ./problem2_client

file descriptor 3 received

Please enter file name to receive:
test.txt

---------Data Received---------
HELLO WORLD
-------------------------------

Please enter file name to receive:
```



```
(base) djikstra@helios:~/Academic/CSN361/L3$ gcc problem2_server.c -o problem2_server
(base) djikstra@helios:~/Academic/CSN361/L3$ ./problem2_server

file descriptor 3 received

Successfully binded!

Waiting for file name...

File Name Received: test.txt

File Successfully opened!

Waiting for file name...
```

Problem Statement 3

Write a TCL code for network simulator NS2 to demonstrate the star topology among a set of computer nodes. Given N nodes, one node will be assigned as the central node and the other nodes will be connected to it to form the star. You have to set up a TCP connection between k pairs of nodes and demonstrate the packet transfer between them using Network Animator (NAM). Use File Transfer protocol (FTP) for the same. Each link should have different color of packets to differentiate the packets transferred between each pair of nodes. The program should take the number of nodes (N) as input followed by k pairs of nodes.

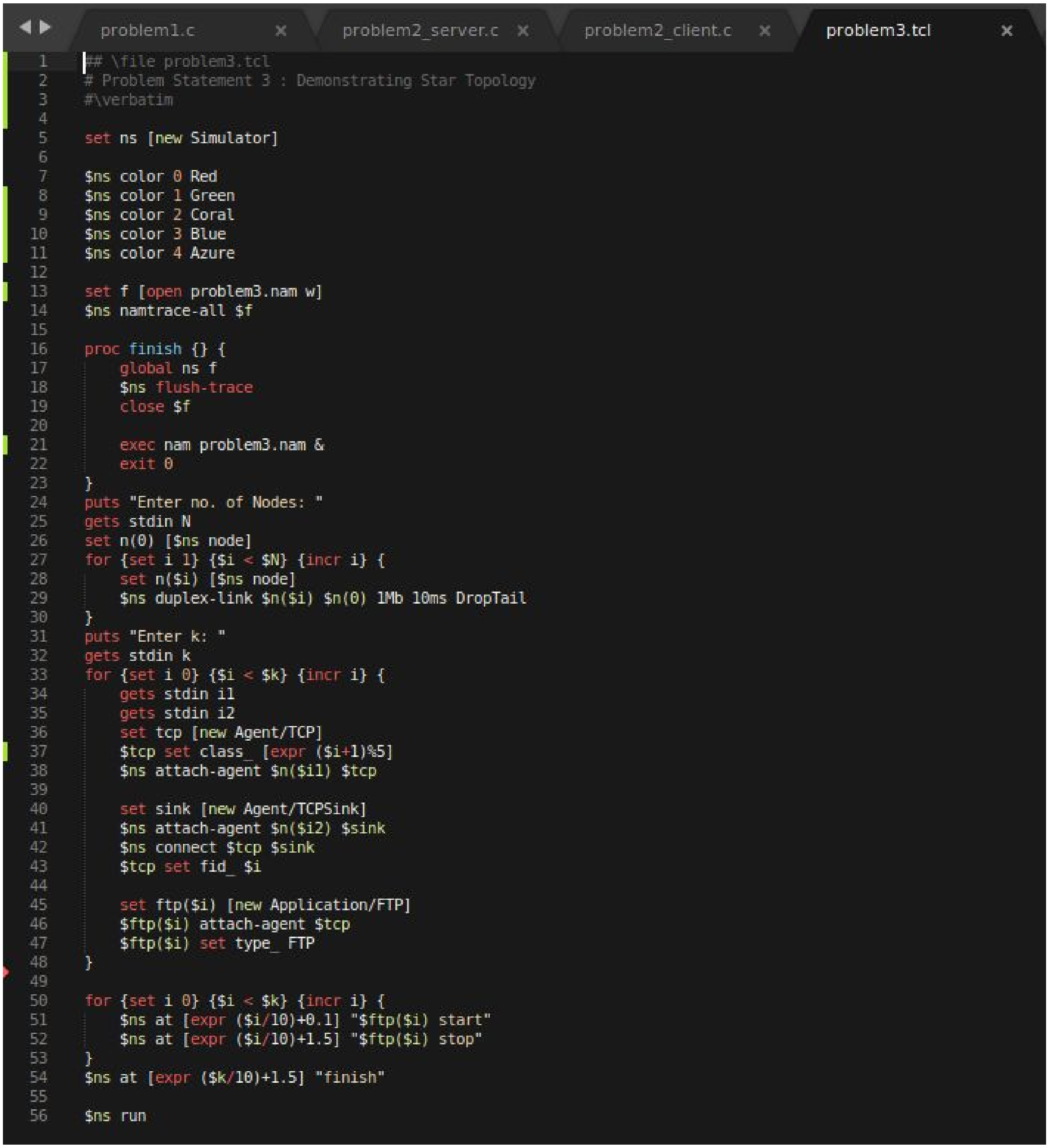
Algorithm:

- Create an empty nam file
- Bind the nam file to current tcl code's nam-trace
- Input n and k pairs
- Create n nodes and connect every created node to first node
- Create k FTP connections among specified nodes
- Specify time to start and stop transaction

Data Structures:

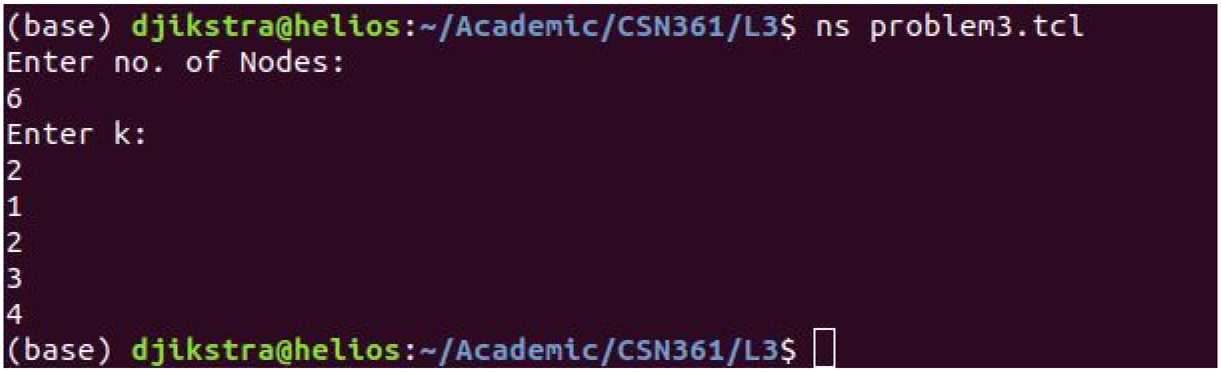
- Ns node to represent nodes

Code:

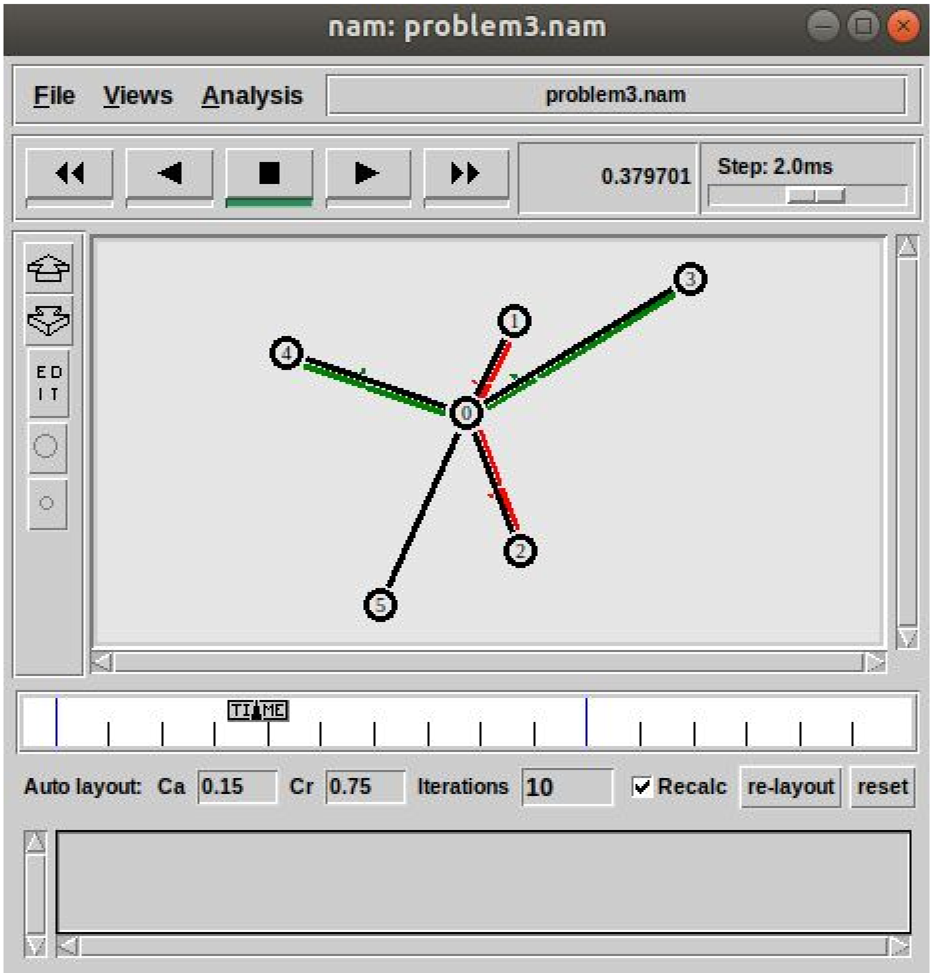


```tcl
## \file problem3.tcl
# Problem Statement 3 : Demonstrating Star Topology
#\verbatim

set ns [new Simulator]

$ns color 0 Red
$ns color 1 Green
$ns color 2 Coral
$ns color 3 Blue
$ns color 4 Azure

set f [open problem3.nam w]
$ns namtrace-all $f

proc finish {} {
    global ns f
    $ns flush-trace
    close $f

    exec nam problem3.nam &
    exit 0
}
puts "Enter no. of Nodes: "
gets stdin N
set n(0) [$ns node]
for {set i 1} {$i < $N} {incr i} {
    set n($i) [$ns node]
    $ns duplex-link $n($i) $n(0) 1Mb 10ms DropTail
}
puts "Enter k: "
gets stdin k
for {set i 0} {$i < $k} {incr i} {
    gets stdin i1
    gets stdin i2
    set tcp [new Agent/TCP]
    $tcp set class_ [expr ($i+1)%5]
    $ns attach-agent $n($i1) $tcp

    set sink [new Agent/TCPSink]
    $ns attach-agent $n($i2) $sink
    $ns connect $tcp $sink
    $tcp set fid_ $i

    set ftp($i) [new Application/FTP]
    $ftp($i) attach-agent $tcp
    $ftp($i) set type_ FTP
}

for {set i 0} {$i < $k} {incr i} {
    $ns at [expr ($i/10)+0.1] "$ftp($i) start"
    $ns at [expr ($i/10)+1.5] "$ftp($i) stop"
}
$ns at [expr ($k/10)+1.5] "finish"

$ns run
```

Output



```
(base) djikstra@helios:~/Academic/CSN361/L3$ ns problem3.tcl
Enter no. of Nodes:
6
Enter k:
2
1
2
3
4
(base) djikstra@helios:~/Academic/CSN361/L3$ 
```



Ca 0.15 Cr 0.75 Iterations 10 ☑ Recalc

Problem Statement 4

Write a TCL code for network simulator NS2 to demonstrate the ring topology among a set of computer nodes. Given N nodes, each node will be connected to two other nodes in the form of a ring. You have to set up a TCP connection between k pairs of nodes and demonstrate packet transfer between them using Network Animator (NAM). Use File Transfer protocol (FTP) for the same. Each link should have different color of packets to differentiate the packets transferred between each pair of nodes. The program should take the number of nodes (N) as input followed by k pairs of nodes.

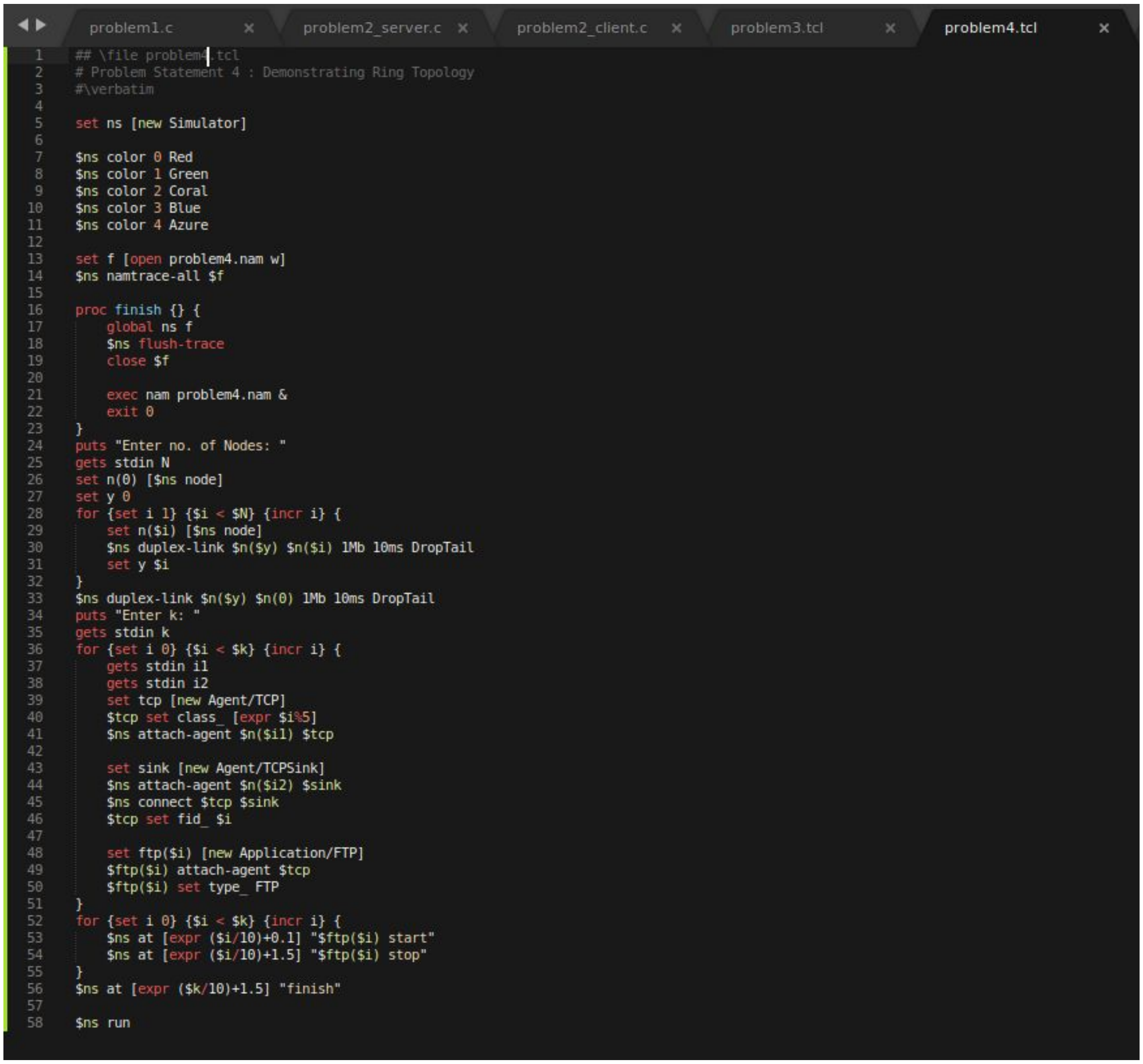
Algorithm:

- Create an empty nam file
- Bind the nam file to current tcl code's nam-trace
- Input n and k pairs
- Create n nodes and connect every created node to it's previous node
- Connect last node to the first node
- Create k FTP connections among specified nodes
- Specify time to start and stop transaction

Data Structures:

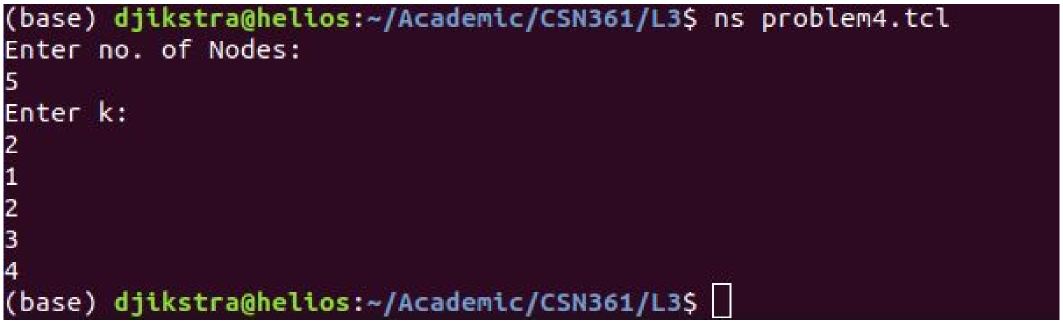
- Ns node to represent nodes

Code:

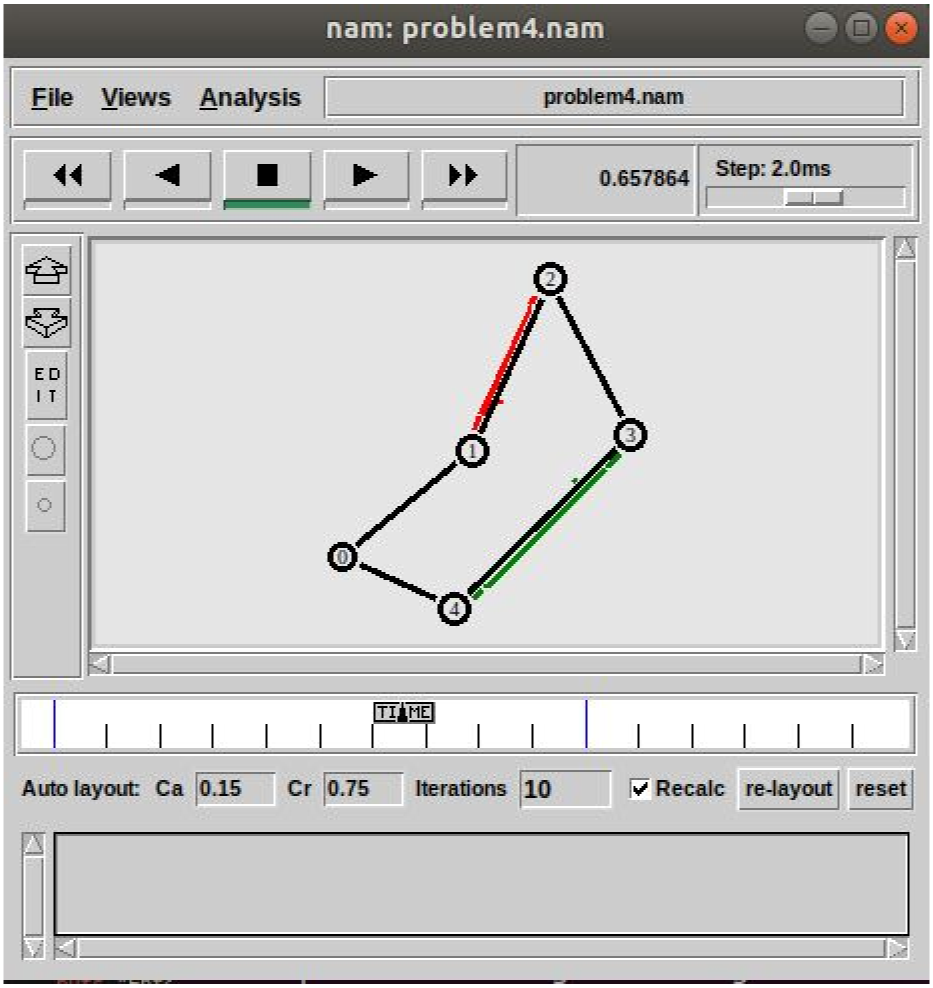


```
## \file problem4.tcl
# Problem Statement 4 : Demonstrating Ring Topology
#\verbatim

set ns [new Simulator]

$ns color 0 Red
$ns color 1 Green
$ns color 2 Coral
$ns color 3 Blue
$ns color 4 Azure

set f [open problem4.nam w]
$ns namtrace-all $f

proc finish {} {
    global ns f
    $ns flush-trace
    close $f

    exec nam problem4.nam &
    exit 0
}
puts "Enter no. of Nodes: "
gets stdin N
set n(0) [$ns node]
set y 0
for {set i 1} {$i < $N} {incr i} {
    set n($i) [$ns node]
    $ns duplex-link $n($y) $n($i) 1Mb 10ms DropTail
    set y $i
}
$ns duplex-link $n($y) $n(0) 1Mb 10ms DropTail
puts "Enter k: "
gets stdin k
for {set i 0} {$i < $k} {incr i} {
    gets stdin i1
    gets stdin i2
    set tcp [new Agent/TCP]
    $tcp set class_ [expr $i%5]
    $ns attach-agent $n($i1) $tcp

    set sink [new Agent/TCPSink]
    $ns attach-agent $n($i2) $sink
    $ns connect $tcp $sink
    $tcp set fid_ $i

    set ftp($i) [new Application/FTP]
    $ftp($i) attach-agent $tcp
    $ftp($i) set type_ FTP
}
for {set i 0} {$i < $k} {incr i} {
    $ns at [expr ($i/10)+0.1] "$ftp($i) start"
    $ns at [expr ($i/10)+1.5] "$ftp($i) stop"
}
$ns at [expr ($k/10)+1.5] "finish"

$ns run
```

Output



```
(base) djikstra@helios:~/Academic/CSN361/L3$ ns problem4.tcl
Enter no. of Nodes:
5
Enter k:
2
1
2
3
4
(base) djikstra@helios:~/Academic/CSN361/L3$ 
```


nam: problem4.nam
File
Views
Analysis
problem4.nam
0.657864
Step: 2.0ms
ED IT
2
1
3
0
4
TIME
Auto layout:
Ca
0.15
Cr
0.75
Iterations
10
☑ Recalc
re-layout
reset

Problem Statement 5

Write a TCL code for network simulator NS2 to demonstrate the bus topology among a set of computer nodes. Given N nodes, each node will be connected to a common link. You have to set up a TCP connection between k pairs of nodes and demonstrate packet transfer between them using Network Animator (NAM). Use File Transfer protocol (FTP) for the same. Each link should have different color of packets to differentiate the packets transferred between each pair of nodes. The program should take the number of nodes (N) as input followed by k pairs of nodes.

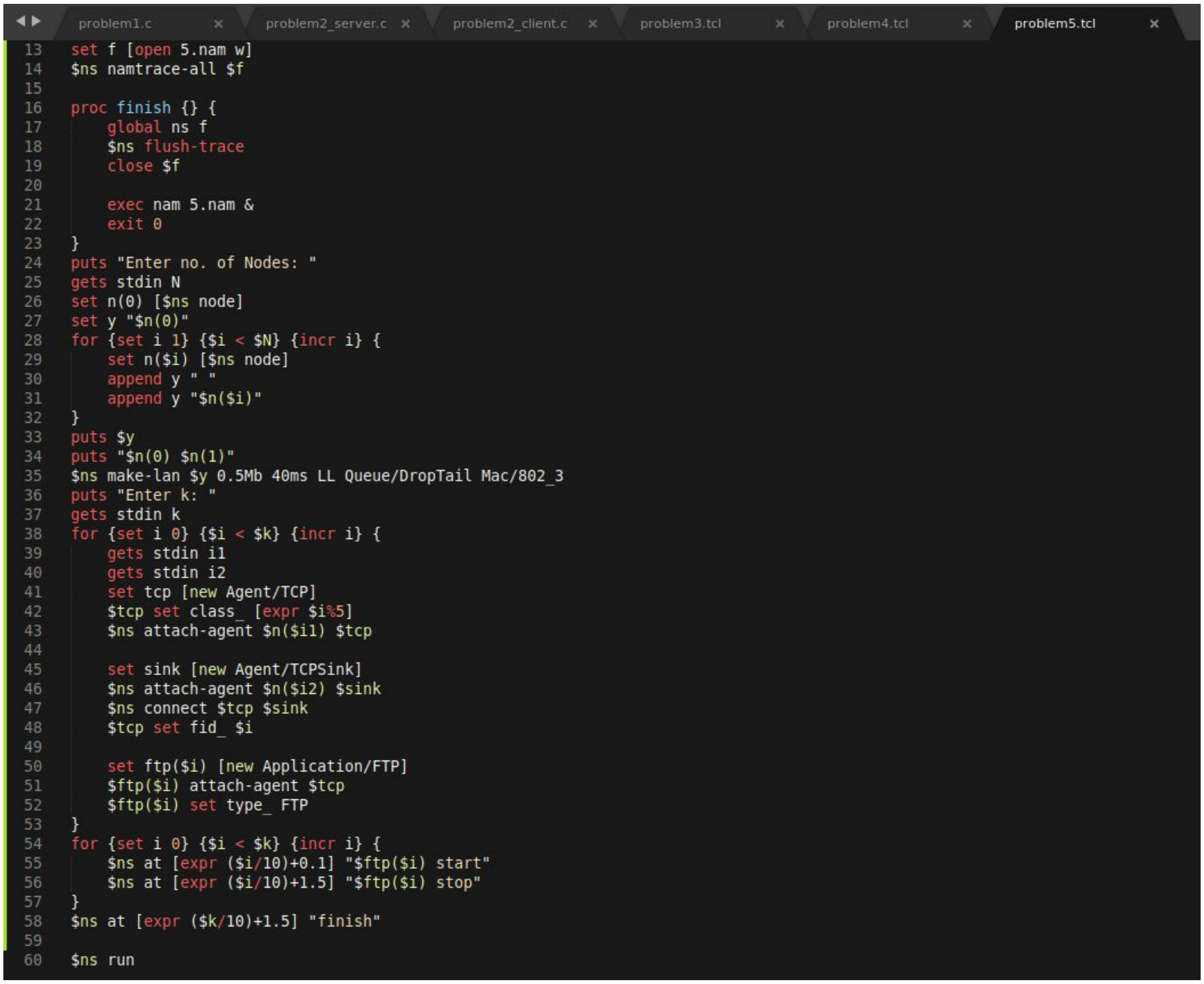
Algorithm:

- Create an empty nam file
- Bind the nam file to current tcl code's nam-trace
- Input n and k pairs
- Create n nodes
- Make a LAN connection among the nodes
- Create k FTP connections among specified nodes
- Specify time to start and stop transaction

Data Structures:

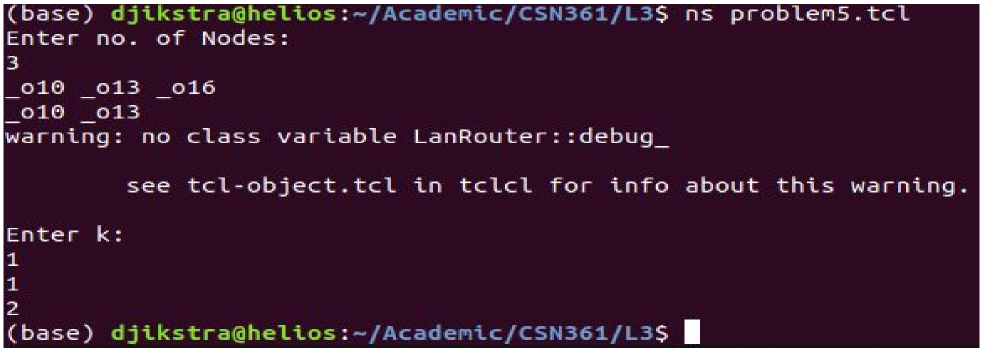
- Ns node to represent nodes

Code:

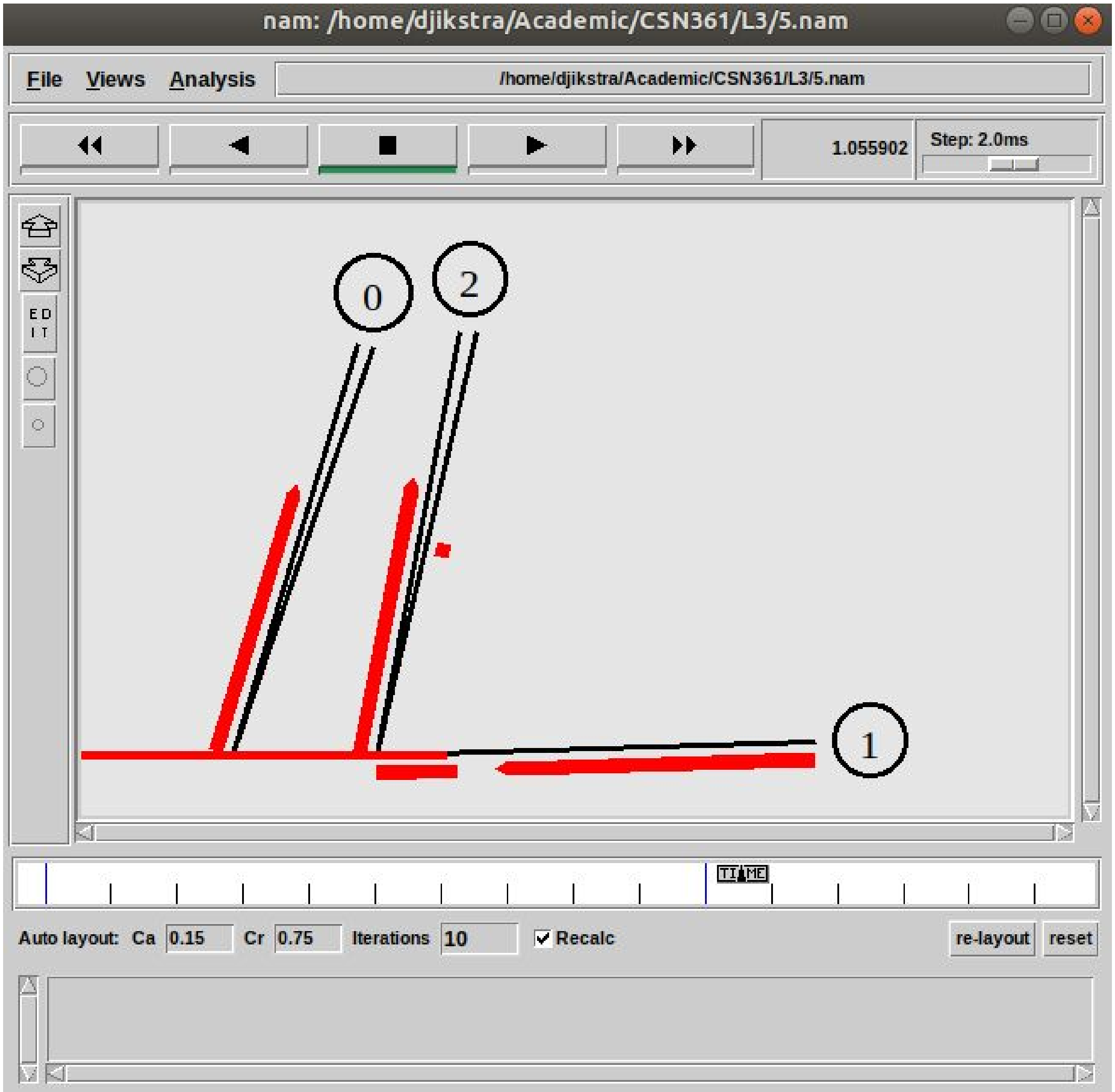


```
set f [open 5.nam w]
$ns namtrace-all $f

proc finish {} {
    global ns f
    $ns flush-trace
    close $f

    exec nam 5.nam &
    exit 0
}
puts "Enter no. of Nodes: "
gets stdin N
set n(0) [$ns node]
set y "$n(0)"
for {set i 1} {$i < $N} {incr i} {
    set n($i) [$ns node]
    append y " "
    append y "$n($i)"
}
puts $y
puts "$n(0) $n(1)"
$ns make-lan $y 0.5Mb 40ms LL Queue/DropTail Mac/802_3
puts "Enter k: "
gets stdin k
for {set i 0} {$i < $k} {incr i} {
    gets stdin i1
    gets stdin i2
    set tcp [new Agent/TCP]
    $tcp set class_ [expr $i%5]
    $ns attach-agent $n($i1) $tcp

    set sink [new Agent/TCPSink]
    $ns attach-agent $n($i2) $sink
    $ns connect $tcp $sink
    $tcp set fid_ $i

    set ftp($i) [new Application/FTP]
    $ftp($i) attach-agent $tcp
    $ftp($i) set type_ FTP
}
for {set i 0} {$i < $k} {incr i} {
    $ns at [expr ($i/10)+0.1] "$ftp($i) start"
    $ns at [expr ($i/10)+1.5] "$ftp($i) stop"
}
$ns at [expr ($k/10)+1.5] "finish"

$ns run
```

Output



```
(base) djikstra@helios:~/Academic/CSN361/L3$ ns problem5.tcl
Enter no. of Nodes:
3
_o10 _o13 _o16
_o10 _o13
warning: no class variable LanRouter::debug_

        see tcl-object.tcl in tclcl for info about this warning.

Enter k:
1
1
2
(base) djikstra@helios:~/Academic/CSN361/L3$
```



nam: /home/djikstra/Academic/CSN361/L3/5.nam

File Views Analysis /home/djikstra/Academic/CSN361/L3/5.nam

1.055902 Step: 2.0ms

Auto layout: Ca 0.15 Cr 0.75 Iterations 10 ☑ Recalc re-layout reset