

CSN-361

Assignment -3

Abhijeet Shakya

17114002

Problem Statement 1

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

SOLUTION:

```
#include<stdio.h>
```

```
#include<string.h>
```

```
char findClass(char str[])
```

```
{
```

```
    // storing first octet in arr[] variable
```

```
    char arr[4];
```

```
    int index = 0;
```

```
    while (str[index] != '.')
```

```
    {
```

```
        arr[index] = str[index];
```

```
        index++;
```

```
    }
```

```
    index--;
```

```
    int ip = 0, j = 1;
```

```
    while (index >= 0)
```

```
    {
```

```

        ip = ip + (str[index] - '0') * j;

        j = j * 10;

        index--;
    }

    if (ip >= 1 && ip <= 126)

        return 'A';

    else if (ip >= 128 && ip <= 191)

        return 'B';

    else if (ip >= 192 && ip <= 223)

        return 'C';

    else if (ip >= 224 && ip <= 239)

        return 'D';

    else

        return 'E';
}

// Function to util_separate Network ID as well as
// Host ID and print them
void util_separate(char str[], char ipClass)

```

```

{
    // Initializing ntwork and host array to NULL

    char ntwork[12], host[12];

    for (int k = 0; k < 12; k++)

        ntwork[k] = host[k] = '\0';


    // for class A, only first octet is Network ID

    // and rest are Host ID

    if (ipClass == 'A')
    {
        int index = 0, j = 0;

        while (str[j] != '.')

            ntwork[index++] = str[j++];

        index = 0;

        j++;

        while (str[j] != '\0')

            host[index++] = str[j++];

        printf("Network ID is %s\n", ntwork);

        printf("Host ID is %s\n", host);
    }


    // for class B, first two octet are Network ID

    // and rest are Host ID

    else if (ipClass == 'B')

```

```

{
    int index = 0, j = 0, dotCount = 0;

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

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

    printf("Network ID is %s\n", ntwor);
    printf("Host ID is %s\n", host);
}

// for class C, first three octet are Network ID
// and rest are Host ID

```

```
else if (ipClass == 'C')
{
    int index = 0, j = 0, dotCount = 0;

    // storing in ntworck[] up to 3rd dot
    // dotCount keeps track of number of
    // dots or octets passed
    while (dotCount < 3)
    {
        ntworck[index++] = str[j++];

        if (str[j] == '.')
            dotCount++;
    }

    index = 0;
    j++;

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

    printf("Network ID is %s\n", ntworck);
    printf("Host ID is %s\n", host);
}
```

```

// Class D and E are not divided in Network

// and Host ID

else

    printf("for this particular class "

        " divided into Network and Host ID are not defined \n");

}

int main()

{

    char str[] = "1.4.5.5";

    char ipClass = findClass(str);

    printf("This address has the Class %c\n", ipClass);

    util_separate(str, ipClass);

    return 0;

}

```

Screenshots:

```

as@as-Inspiron-7570: ~/csn-361/assignment3
File Edit View Search Terminal Help
as@as-Inspiron-7570:~/csn-361/assignment3$ gcc ipv4.c -o ipv4
as@as-Inspiron-7570:~/csn-361/assignment3$ ./ipv4
This address has the Class A
Network ID is 1
Host ID is 4.5.5
as@as-Inspiron-7570:~/csn-361/assignment3$ 

```

Problem Statement 2

Write a C program to demonstrate both Zombie and Orphan process.

SOLUTION:

The solution has 2 programs:

Server : This program creates a node which has a socket to listen on the port 8080 (localhost).

`int sockfd = socket(AF_INET, SOCK_STREAM, 0)` creates a socket with IPv4 communication domain and TCP communication type.

`setsockopt()` method helps in manipulating options for the socket referred by the file descriptor `sockfd`.

`bind()` method binds the socket to the address and port number specified in `addr(custom data structure)`.

`listen()` method puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection.


`accept()` method extracts the first connection request on the queue of pending connections for the listening socket, `sockfd`, creates a new connected socket, and returns a new file descriptor referring to that socket.

`sockaddr_in` is a struct for all syscalls and functions that deal with internet addresses.

```
#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 BUFF_SIZE 32

#define cipherKey 'S'

#define sendrecvflag 0

#define nofile "File Not Found!"


void clear_buffer(char* b)

{

    int i;

    for (i = 0; i < BUFF_SIZE; i++)

        b[i] = '\0';

}


// function to encrypt

char Cipher(char ch)

{
```

```

    return ch ^ cipherKey;
}

// 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;
}
```

```
// driver code
```

```
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[BUFF_SIZE];
```

```
    FILE* fp;
```

```
    // socket()
```


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

```
    if (sockfd < 0)
```

```
        printf("\nfile descriptor not received!!\n");
```

```
    else
```

```
        printf("\nDescriptor %d received\n", sockfd);
```



```
// bind()

if (bind(sockfd, (struct sockaddr*)&addr_con, sizeof(addr_con)) == 0)

    printf("\nbinded successfully\n");

else

    printf("\nBinding Failed!\n");


while (1) {

    printf("\nWait for file name...\n");


    // receive file name

    clear_buffer(net_buf);


    nBytes = recvfrom(sockfd, net_buf,

                                BUFF_SIZE, sendrecvflag,

                                (struct sockaddr*)&addr_con, &addrlen);


    fp = fopen(net_buf, "r");

    printf("\n File name is : %s\n", net_buf);

    if (fp == NULL)

        printf("\nFile open failed!\n");

    else

        printf("\nFile opened!\n");
```

```

while (1) {

    // process
    if (sendFile(fp, net_buf, BUFF_SIZE)) {

        sendto(sockfd, net_buf, BUFF_SIZE,

            sendrecvflag,

            (struct sockaddr*)&addr_con, addrlen);

        break;
    }

    // send
    sendto(sockfd, net_buf, BUFF_SIZE,

        sendrecvflag,

        (struct sockaddr*)&addr_con, addrlen);

    clear_buffer(net_buf);
}


if (fp != NULL)

    fclose(fp);
}

return 0;
}

```

Client: This program created a node which sends a request on port 8080 to be read by the server program.



The `connect()` system call connects the socket referred to by the file descriptor `sockfd` to the address specified by `addr`. Server's address and port is specified in `addr`.

```
#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"

#define PORT_NO 15050

#define BUFF_SIZE 32

#define cipherKey 'S'

#define sendrecvflag 0


// function for decryption

char decrypt(char ch)

{
```

```
return ch ^ cipherKey;
}

// function to clear buffer
void clear_buffer(char* b)
{
    int i;

    for (i = 0; i < BUFF_SIZE; i++)
        b[i] = '\0';
}

// function to receive file
int receive_file(char* buf, int s)
{
    int i;

    char ch;

    for (i = 0; i < s; i++) {
        ch = buf[i];

        ch = decrypt(ch);

        if (ch == EOF)
            return 1;

        else
            printf("%c", ch);
    }
}
```

```
return 0;
}
```

```
// driver code
```

```
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[BUFF_SIZE];

    FILE* fp;

    // socket()

    sockfd = socket(AF_INET, SOCK_DGRAM,

                    IP_PROTOCOL);

    if (sockfd < 0)

        printf("\nfile descriptor not received!!\n");

    else
```




```
printf("\nDescriptor %d received\n", sockfd);
```

```
while (1) {
```

```
    printf("\nEnter file name :\n");
```

```
    scanf("%s", net_buf);
```

```
    sendto(sockfd, net_buf, BUFF_SIZE,
```

```
           sendrecvflag, (struct sockaddr*)&addr_con,
```

```
           addrlen);
```

```
    printf("\n Received !\n");
```

```
while (1) {
```

```
    // receive
```

```
    clear_buffer(net_buf);
```

```
    nBytes = recvfrom(sockfd, net_buf, BUFF_SIZE,
```

```
                    sendrecvflag, (struct sockaddr*)&addr_con,
```

```
                    &addrlen);
```

```
    // process
```

```
    if (receive_file(net_buf, BUFF_SIZE)) {
```

```
        break;
```

```
    }
```

```
}
```

```
printf("\n done \n");
```

```
}  
return 0;  
}
```

Screenshots:

```
as@as-Inspiron-7570: ~/csn-361/assignment3
File Edit View Search Terminal Help
as@as-Inspiron-7570:~/csn-361/assignment3$ gcc udp_client.c -o udp_client
as@as-Inspiron-7570:~/csn-361/assignment3$ ./udp_client
```

Server waiting for filename to be entered by client.

```
as@as-Inspiron-7570: ~/csn-361/assignment3
File Edit View Search Terminal Help
as@as-Inspiron-7570:~/csn-361/assignment3$ gcc udp_server.c -o udp_server
as@as-Inspiron-7570:~/csn-361/assignment3$ ./udp_server

Descriptor 3 received

binded successfully

Wait for file name...

```

File name entered by user, and server opened the file, and sent back the information in the file.

```
as@as-Inspiron-7570: ~/csn-361/assignment3
File Edit View Search Terminal Help
as@as-Inspiron-7570:~/csn-361/assignment3$ gcc udp_client.c -o udp_client
as@as-Inspiron-7570:~/csn-361/assignment3$ ./udp_client

Descriptor 3 received

Enter file name :
test.txt

Received !
CSN-361 Assignment3 udp check

done

Enter file name :

```

File name recieved by server and file opened successfully. Data in the file is sent back to the client.

```
as@as-Inspiron-7570: ~/csn-361/assignment3
File Edit View Search Terminal Help
as@as-Inspiron-7570:~/csn-361/assignment3$ gcc udp_server.c -o udp_server
as@as-Inspiron-7570:~/csn-361/assignment3$ ./udp_server

Descriptor 3 received
binded successfully
Wait for file name...

  File name is : test.txt

File opened!
Wait for file name...
█
```

Problem Statement 3:

Write a TCL code for network simulator NS2 to demonstrate the startopology 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.

Solution:

```
set ns [new Simulator]
```

```
$ns color 0 Red
```

```
$ns color 1 Blue
```

```
$ns color 2 Azure
```

```
$ns color 3 Coral
```

```
$ns color 4 Cyan
```

```
set f [open 3.nam w]
```

```
$ns namtrace-all $f
```

```
proc finish {} {
```

```

    global ns f

    $ns flush-trace

    close $f

    exec nam 3.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%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
}

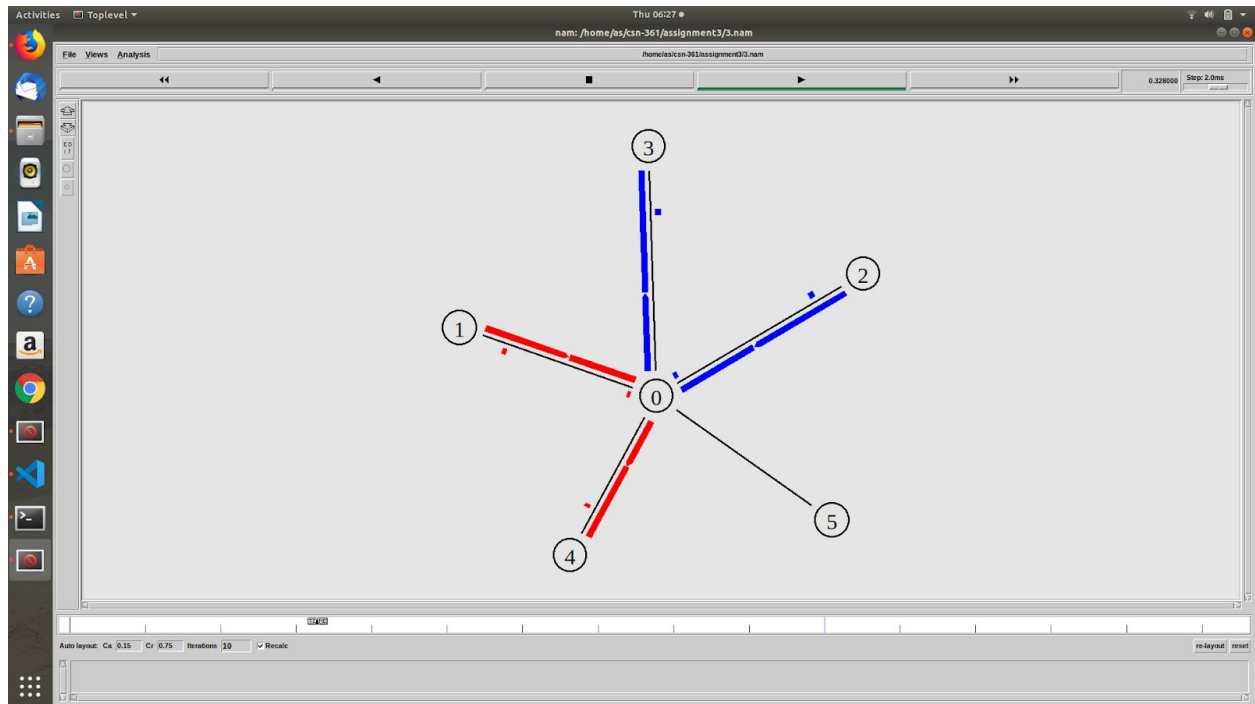
# $ns duplex-link $n0 $n1 1Mb 10ms DropTail

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
```

Screenshots:



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.

Solution:

```
puts "Enter no. of Nodes and k: "
```

```
set data [gets stdin]
```

```
scan $data "%d %d" N k
```

```
set ns [new Simulator]
```

```
$ns color 0 Red
```

```
$ns color 1 Blue
```

```
$ns color 2 Azure
```

```
$ns color 3 Coral
```

```
$ns color 4 Cyan
```

```
$ns rtproto DV
```

```
set nf [open out.nam w]
```



```
$ns namtrace-all $nf
```

```
proc finish {} {  
    global ns nf  
    $ns flush-trace  
    close $nf  
    exec nam out.nam  
    exit 0  
}
```

```
for {set i 0} {$i < $N} {incr i} {  
    set n($i) [$ns node]  
}
```

```
for {set i 0} {$i < $N} {incr i} {  
    $ns duplex-link $n($i) $n([expr ($i + 1) % $N]) 1Mb 10ms DropTail  
}
```

```
puts "Enter k pairs: "
```

```
for {set i 0} {$i < $k} {incr i} {  
    set input [gets stdin]
```

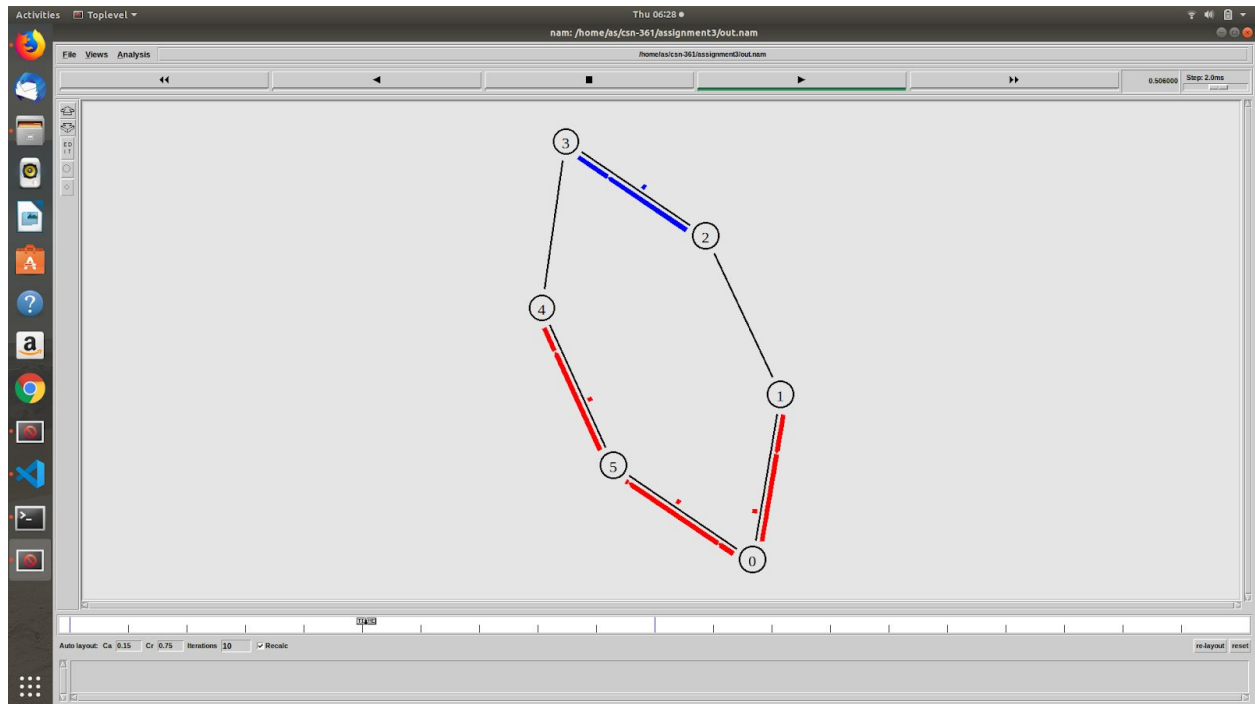
```
    scan $input "%d %d" u v
```

```
set tcp [new Agent/TCP]
$ns attach-agent $n($u) $tcp
$tcp set class_ $i
set sink [new Agent/TCPSink]
$ns attach-agent $n($v) $sink
$ns connect $tcp $sink
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp
$ns at 0.1 "$ftp0 start"
$ns at 1.5 "$ftp0 stop"
}
```

```
$ns at 2.0 "finish"
```

```
$ns run
```

Screenshots:



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.

Solution:

```
# create a simulator object
```

```
set ns [new Simulator]
```

```
# define different colors for data flows (for NAM)
```

```
$ns color 0 blue
```

```
$ns color 1 red
```

```
$ns color 2 cyan
```

```
$ns color 3 green
```

```
$ns color 4 yellow
```

```
$ns color 5 violet
```

```
# open the nam trace file
```

```
set nf [open bus.nam w]
```

```
$ns namtrace-all $nf
```



define a 'finish' procedure


```
proc finish {} {  
    global ns nf  
    $ns flush-trace  
    #Close the trace file  
    close $nf  
    #Execute nam on the trace file  
    exec nam bus.nam &  
    exit  
}
```

take N as input

```
puts "Enter N"  
set data [gets stdin]  
scan $data "%d" N
```

create N new nodes

```
for {set i 0} {$i < $N} {incr i} {  
    set n($i) [$ns node]  
}
```



```
set str "$n(0)"
```

```
for {set i 1} {$i < $N} {incr i} {
```

```
    append str " $n($i)"
```

```
}
```

```
$ns make-lan $str 0.5Mb 40ms LL Queue/DropTail Mac/802_3
```

```
# take K as input
```

```
puts "Enter K"
```

```
set data [gets stdin]
```

```
scan $data "%d" K
```

```
puts "Enter K pairs of nodes: <source> <sink>"
```

```
for {set i 0} {$i < $K} {incr i} {
```

```
    # take source and sink as input
```

```
    set data [gets stdin]
```

```
    scan $data "%d %d" j k
```

```
    # create a TCP agent and attach it to node n(j)
```

```
    set tcp($i) [new Agent/TCP]
```

```
    $ns attach-agent $n($j) $tcp($i)
```

```

# create a TCP Sink agent (delayed ACK TCP) for TCP and attach it to node n(k)
set sink($i) [new Agent/TCPSink/DelAck]
$ns attach-agent $n($k) $sink($i)

# connect the traffic source with traffic sink
$ns connect $tcp($i) $sink($i)

# set different color by using flow ID
$tcp($i) set fid_ $i
}

# create a FTP and attach to tcp
for {set i 0} {$i < $K} {incr i} {
    set ftp($i) [new Application/FTP]
    $ftp($i) attach-agent $tcp($i)
}

#Schedule events for the FTP agents
for {set i 0} {$i < $K} {incr i} {
    $ns at 0.5 "$ftp($i) start"
    $ns at 3.5 "$ftp($i) stop"
}

```


}

#Call the finish procedure after 5 seconds of simulation time

\$ns at 5.0 "finish"

#Run the simulation

\$ns run

Screenshots:

