COLLEGE OF ENGINEERING ADOOR

DEPARTMENT OF COMPUTER SCIECE AND ENGINEERING

NETWORK PROGRAMMING

LAB MANUAL

Experiment 1

Getting started with Basics of Network configurations files and Networking Commands in Linux.

The important network configuration files in Linux operating systems are

1. **/etc/hosts**

This file is used to resolve hostnames on small networks with no DNS server. This text file contains a mapping of an IP address to the corresponding host name in each line. This file also contains a line specifying the IP address of the loopback device i.e, *127.0.0.1* is mapped to *localhost*.

A typical *hosts* file is as shown

127.0.0.1 localhost

127.0.1.1 anil-300E4Z-300E5Z-300E7Z

2. **/etc/resolv.conf**

This configuration file contains the IP addresses of DNS servers and the search domain.

A sample file is shown

DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN

nameserver 127.0.1.1

3. **/etc/sysconfig/network**

This configuration file specifies routing and host information for all network interfaces. It contains directives that are global specific. For example if NETWORKING=yes, then /etc/init.d/network activates network devices.

4. **/etc/nsswitch.conf**

This file includes database search entries. The directive specifies which database is to be searched first. The important

Linux networking commands are

1. ifconfig

This command gives the configuration of all interfaces in the system. It can be run

with an interface name to get the details of the interface. ifconfig *wlan0*

Link encap:Ethernet HWaddr b8:03:05:ad:6b:23

inet addr:192.168.43.15 Bcast:192.168.43.255 Mask:255.255.255.0

inet6 addr: 2405:204:d206:d3b1:ba03:5ff:fead:6b23/64 Scope:Global inet6 addr:

fe80::ba03:5ff:fead:6b23/64 Scope:Link

inet6 addr: 2405:204:d206:d3b1:21ee:5665:de59:bd4e/64 Scope:Global UP

BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:827087 errors:0 dropped:0 overruns:0 frame:0 TX

packets:433391 errors:0 dropped:0 overruns:0 carrier:0 collisions:0

txqueuelen:1000

RX bytes:1117797710 (1.1 GB) TX bytes:53252386 (53.2 MB)

This gives the IP address, subnet mask, and broadcast address of the wireless LAN adapter. Also tells that it can support multicasting.

If eth0 is given as the parameter, the command gives the details of the Ethernet adapter.

2. netstat

This command command gives network status information.

Netstat -*i*

Iface	MTU	Met	RX-OK	RX-ERR	RX-DRP	RX-OVR	TX-OK	TX-ERR	TX-DRP	TX-OVR	Flg
eth0	1500	0	0	0	0	0	0	0	0	0	BMU
lo	65536	0	12166	0	0	0	12166	0	0	0	LRU
wlan0	1500	0	827946	0	0	0	434246	0	0	0	BMRU

As shown above, the command with *-i* flag provides information on the interfaces. lo stands for loopback interface.

3. **ping**

This is the most commonly used command for checking connectivity.

ping *www.google.com*

PING www.google.com (172.217.163.36) 56(84) bytes of data.

64 bytes from maa05s01-in-f4.1e100.net (172.217.163.36): icmp_seq=1 ttl=53 time=51.4 ms 64 bytes from maa05s01-in-f4.1e100.net (172.217.163.36): icmp_seq=2 ttl=53 time=50.3 ms 64 bytes from maa05s01-in-f4.1e100.net (172.217.163.36): icmp_seq=3 ttl=53 time=48.5 ms 64 bytes from maa05s01-in-f4.1e100.net (172.217.163.36): icmp_seq=4 ttl=53 time=59.8 ms 64 bytes from maa05s01-in-f4.1e100.net (172.217.163.36): icmp_seq=5 ttl=53 time=57.8 ms 64 bytes from maa05s01-in-f4.1e100.net (172.217.163.36): icmp_seq=6 ttl=53 time=59.2 ms 64 bytes from maa05s01-in-f4.1e100.net (172.217.163.36): icmp_seq=7 ttl=53 time=68.2 ms 64 bytes from maa05s01-in-f4.1e100.net (172.217.163.36): icmp_seq=8 ttl=53 time=58.8 ms

^C

--- www.google.com ping statistics ---

8 packets transmitted, 8 received, 0% packet loss, time 7004ms rtt

min/avg/max/mdev = 48.533/56.804/68.266/6.030 ms

A healthy connection is determined by a steady stream of replies with consistent times. Packet loss is shown by discontinuity of sequence numbers. Large scale packet loss indicates problem along the

path.

Experiment 2

To familiarize and understand the use and functioning of System Calls used for Operating system and network programming in Linux.

Some system calls of Linux operating systems

1. ps

This command tells which all processes are running on the system when *ps* runs. ps -ef

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	13:55	?	00:00:01	/sbin/init
root	2	0	0	13:55	?	00:00:00	[kthreadd]
root	3	2	0	13:55	?	00:00:00	[ksoftirqd/0]
root	4	2	0	13:55	?	00:00:01	[kworker/0:0]
root	5	2	0	13:55	?	00:00:00	[kworker/0:0H]
root	7	2	0	13:55	?	00:00:00	[rcu_sched]
root	8	2	0	13:55	?	00:00:00	[rcuos/0]

This command gives processes running on the system, the owners of the processes and the names of the processes. The above result is an abridged version of the output.

2. fork

This system call is used to create a new process. When a process makes a *fork* system call, a new process is created which is identical to the process creating it. The process which calls *fork* is called the parent process and the process that is created is called the child process. The child and parent processes are identical, i.e, child gets a copy of the parent's data space, heap and stack, but have different physical address spaces. Both processes start execution from the line next to *fork*. Fork returns the process id of the child in the parent process and returns 0 in the child process.

```
#include<stdio.h> void

main()

{

    int pid;

    pid = fork(); if(pid >

    0)

    {

        printf (“ Iam parent\n”);

    }

    else

    {

        printf(“Iam child\n”);

    }

}
```

The parent process prints the first statement and the child prints the next statement.

3. exec

New programs can be run using *exec* system call. When a process calls exec, the process is completely replaced by the new program. The new program starts executing from its main function.

A new process is not created, process id remains the same, and the current process's text, data, heap, and stack segments are replaced by the new program. *exec* has many flavours one of which is *execv.*

execv takes two parameters. The first is the pathname of the program that is going to be executed. The second is a pointer to an array of pointers that hold the addresses of arguments. These arguments are the command line arguments for the new program.

4. wait

When a process terminates, its parent should receive some information reagarding the process like the process id, the termination status, amount of CPU time taken etc. This is possible only if the parent process waits for the termination of the child process. This waiting is done by calling the *wait* system call. When the child process is running, the parent blocks when *wait* is called. If the child terminates normally or abnormally, *wait* immedaitely returns with the termination status of the child. The wait system call takes a parameter which is a pointer to a location in which the termination status is stored.

5. exit

When *exit* function is called, the process undergoes a normal termination.

6. open

This system call is used to open a file whose pathname is given as the first parameter of the function. The second parameter gives the options that tell the way in which the file can be used.

open(filepathname , O_RDWR);

This causes the file to be read or written. The function returns the file descriptor of the file.

7. read

This system call is used to read data from an open file. read(fd,

buffer, sizeof(buffer));

The above function reads sizeof(buffer) bytes into the array named buffer. If the end of file is encountered, 0 is returned, else the number of bytes read is returned.

8. write

Data is written to an open file using *write* function. write(fd,

buffer, sizeof(buffer));

System calls for network programming in Linux

1. **Creating a socket**

int *socket* (**int** *domain*, **int** *type*, **int** *protocol*);

This sytem call creates a socket and returns a socket descriptor. The *domain* parameter specifies a communication domain; this selects the protocol family which will be used for communication. These families are defined in <sys/socket.h>. In this program the AF_INET family is used. The *type*

parameter indicates the communication semantics. SOCK_STREAM is used for tcp connection while SOCK_DGRAM is used for udp connection. The *protocol* parameter specifies the protocol used and is always 0. The header files used are <sys/types.h> and <sys/socket.h>.

Experiment 3

Familiarization and implementation of programs related to Process and thread.

Process

A process is an active entity which has a text section which contains the program code, data section which contains global variables, a heap that is dynamically allocated during process run time, and a process stack which contains temporary data. A process is created when an executable program is loaded into memory. Two or more processes can be associated with the same program and even though they may share the same text section, they have different stack, data and heap sections.

Threads

A thread is a basic unit of CPU utilization. It has an ID, register set and a stack. Normally a process has a single thread of control. However, processes can have multiple threads of control which will allow them to perform more than one task at a time. Threads within a process share the same code section, data section and operating system resources like open files and signals. Since process creation is time consuming and resource intensive, it is often advantageous to use one process that contains multiple threads to do the same task.

Thread libraries provide users with an API for creating and managing threads. A thread library may be created at the user level or at the kernel level. In the former case, invoking a function in the library results in a local function call, while in the latter case such an invocation will be a system call. One important thread library is the POSIX Pthreads. POSIX standard gives a specification for thread creation and its behaviour. Operating systems, mostly UNIX type systems implement this standard in different ways.

Some Pthread APIs are

1. *pthread_init_attr(&attr)*
   This function sets the attributes of the thread like stack size, scheduling information etc.

2. *pthread_create(&tid, &attr, func, arg)*
   This function is used for creating a thread. It takes the following parameters

i) *tid* - thread ID

ii) *attr* – thread attributes

iii) *func* – the new function where thread execution begins

iv) *arg* – parameter that is passed into the function

3. *pthread_join(tid, NULL)*

The parent thread waits till the child threads terminate and join with the parent. Afterwards, the parent continues execution.

Example:

This program shows how a child process is created to do a task. The program in *expt2.c* creates a child process for finding the sum of two numbers.

expt2.c

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

main(int argc , char * argv[])
{
   int pid;

   pid = fork();

   if(pid == 0) // child process
   {
      execv("/home/anil/NetworkLab/EXPT3/prog2",    argv);
   }

   wait(NULL); exit(0);

}
```

prog2.c

```
#include<stdio.h>

void main ( int argc, char * argv[])
{
   int a,b, sum;

    a = atoi(argv[1]);
    b = atoi(argv[2]);
    sum = a + b;
    printf("sum = %d\n", sum);
}
```

How to get output ?

Find the sum of 3 and 4,

1. Compile *prog2.c* using the command gcc -o
   prog2 prog2.c
2. Compile *expt2.c* using the command gcc -o
   expt2 expt2.c
3. *./expt2* 3 4

will give 7 as the sum. The path for *execv* should be substituted with your path.

After the fork system call, the child process executes the *prog2* program. The parent process waits for the termination of the child process.

The same program can be done using threads
expt3.c

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

void * func_sum(void*); int
sum ;

main(int argc, char * argv[])
{
   pthread_t tid;
   int val[2];
   val[0] = atoi(argv[1]);
   val[1] = atoi(argv[2]); pthread_create(&tid,NULL,func_sum,val);
  pthread_join(tid,NULL);
  printf(" sum = %d\n", sum);
}
```

The command for compiling the program is gcc -o

expt3 expt3.c -lpthread

Run the command

./expt3 3 4

Experiment 4

Implementation of first readers writers problem

Description

Readers writers problem is a synchronization problem. There are two types of processes:- writers and readers which are sharing a common resource. A reader process can share the process with other readers but not writers. A writer process requires exclusive access to the resource. A very good example is a file being shared among a set of processes. As long as a reader holds the resource and there are new readers arriving, any writer must wait for the resource to become available.

The first reader accessing the resource must compete with any writers, but once a writer succeeds the other readers can pass directly into the critical section provided that at least one reader is still in the critical section. The *readCount* variable gives the number of readers in the critical section. Only when the last reader leaves the critical section can the writers enter the critical section one at a time which will based on *writeBlock* variable.

Algorithm

The algorithm for the readers writers problem is given below (Gary Nutt)

```
reader()
  { while(TRUE) {
    <other computing>
    P(mutex);
       readCount = readCount +1; if(readCount == 1)
          P(writeBlock);
    V(mutex);

/* critical section */
access resource

   P(mutex);
      readCount = readCount – 1; if(readCount == 0)
          V(writeBlock);
   V(mutex);
  }
}

writer()
  { while(TRUE) {
    <other computing>;
     P(writeBlock);

/* critical section */
 access resource
```

```
    V(writeBlock);
  }
}
```

The above algorithm is implemented using threads. *Mutex* and *writeBlock* are two semaphores. The first semaphore guards the *readCount* variable while the latter protects the critical section where the resource is shared.

Program

```
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>
#include<stdlib.h>


void * reader (void *); void *
writer (void *);

sem_t mutex; sem_t
writeBlock;

int readCount =0;

main(int argc, char * argv[])
{
   int i, j, k;

   int N_readers, N_writers;
   int readers_num[100], writers_num[100]; pthread_t

   tid_readers[100], tid_writers[100];

   printf("Enter the number of readers:");
   scanf("%d", &N_readers);

   printf("Enter the number of writers:");
   scanf("%d", &N_writers);


   for(k =0; k < N_readers; k++) readers_num[k] =
   k;

   for(k =0; k < N_writers; k++)
```

```
  writers_num[k] = k;

if(sem_init(&mutex, 0, 1) < 0)
 {
     perror("Could not init semaphore mutex"); exit(1);
 }


if(sem_init(&writeBlock, 0, 1) < 0)
 {
     perror("Could not init semaphore writeBlock"); exit(1);
 }

for( i =0; i < N_readers; i++)
 {
    if(pthread_create(&tid_readers[i], NULL, reader,  &readers_num[i]  ))
    {
       perror("could not create reader thread"); exit(1);
    }

 }

for( j=0; j < N_writers; j++)
 {
 if(pthread_create(&tid_writers[j],  NULL,  writer,  &writers_num[j]))
 {
   perror("could not create writer thread"); exit(1);
 }
 }

 for (i =0; i < N_readers; i++)
 {
    pthread_join(tid_readers[i],   NULL);
 }

 for ( j=0; j < N_writers; j++)
 {
   pthread_join(tid_writers[j],   NULL);
 }

 sem_destroy(&mutex);
```

```
  sem_destroy(&mutex);

}


void * reader (void* param)
{

    int i = *((int *) param);

     while(1)
      {
          sleep(1); if(sem_wait(&mutex) <
         0)
          {
             perror("cannot decrement the semaphore mutex"); exit(1);
          }

          readCount = readCount + 1; if(readCount
          == 1)
           {
              if(sem_wait(&writeBlock)  <  0)
               {
                  perror("cannot decrement the semaphore writeBlock"); exit(1);
               }
           }

              if( sem_post(&mutex) < 0)
              {
                   perror("cannot increment semaphore mutex"); exit(1);
              }

             // READ RESOURCES
              printf("READER %d is READING \n", i); sleep(1);



         if(sem_wait(&mutex) < 0)
          {
             perror("cannot decrement the semaphore mutex"); exit(1);
```

```
        }
        readCount = readCount - 1; if(readCount ==
        0)
         {
            if( sem_post(&writeBlock) < 0)
            {
                perror("cannot increment semaphore mutex"); exit(1);
            }
          }

            if( sem_post(&mutex) < 0)
            {
                perror("cannot increment semaphore mutex"); exit(1);
            }

         }

}

void * writer (void * param)
{
    int i = *((int *) param);

    while(1)
     {
        sleep(1);
            if(sem_wait(&writeBlock)  <  0)
             {
               perror("cannot decrement the semaphore writeBlock"); exit(1);
             }

             // WRITE RESOURCES
            printf("WRITER %d IS WRITING \n", i);
            if( sem_post(&writeBlock) < 0)
            {
```

```
                perror("cannot increment semaphore writeBlock"); exit(1);
            }

    }

}
```

Output

```
anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab$    ./expt4

Enter the number of readers:5 Enter the
number of writers:3

READER   0   is   READING
READER   1   is   READING
READER   2   is   READING
READER   4   is   READING
READER   3   is   READING
WRITER   0   IS   WRITING
WRITER   1   IS   WRITING
WRITER   2   IS   WRITING
READER   2   is   READING
READER   3   is   READING
READER   0   is   READING
READER   1   is   READING
READER   4   is   READING
WRITER   0   IS   WRITING
WRITER   1   IS   WRITING
WRITER   2   IS   WRITING
READER   2   is   READING
READER   4   is   READING
READER   3   is   READING
READER   1   is   READING
READER   0   is   READING
WRITER   0   IS   WRITING
WRITER   1   IS   WRITING
WRITER   2   IS   WRITING
READER   4   is   READING
READER   3   is   READING
READER   0   is   READING
READER   1   is   READING
READER   2   is   READING
WRITER 0 IS WRITING
```

WRITER 1 IS WRITING
WRITER 2 IS WRITING
READER 1 is READING
READER 2 is READING
READER 0 is READING
READER 4 is READING
READER 3 is READING

Experiment 5

Implementation of second readers writers problem

Description

In the first readers writers problem, readers can dominate the resource and it will not be possible for a writer to access the resource. In order to give preference to writers, it is necessary to prevent a subsequent reader process from gaining access to the shared resource till the writer accesses the shared resource and releases it. An algorithm to achieve this task is given below (Gary Nutt).

Here a stream of readers can enter the critical section till a writer arrives. When a writer arrives, it takes access of the resource after all existing readers leave the critical section. When the first writer arrives, it will obtain the *readBlock* semaphore. Then it blocks on the *writeBlock* semaphore, waiting for all readers to clear the critical section. The next reader to arrive will obtain the *writePending* semaphore and then block on the *readBlock* semaphore. If another writer arrives during this time, it will block on the *writeBlock* semaphore. If a second reader arrives, it will block on the *writePending* semaphore.

Algorithm

reader()

```
{ while(TRUE) {
    <other computing>
    P(writePending);
    P(readBlock);
       P(mutex1);
      readCount = readCount +1; if(readCount == 1)
         P(writeBlock);
      V(mutex1);
    V(readBlock);
   V(writePending);


/* critical section */
access resource
```

```
   P(mutex1);
      readCount = readCount – 1; if(readCount == 0)
          V(writeBlock);
   V(mutex1);
  }
}
```

writer()

```
  { while(TRUE) {
    <other computing>;
   P(mutex2);
     writeCount = writeCount + 1;
     if(writeCount == 1) P(readBlock);
  V(mutex2);
     P(writeBlock);
/* critical section */
 access resource
     V(writeBlock);
 P(mutex2);
     writeCount = writeCount - 1;
     if(writeCount == 0) V(readBlock);
V(mutex2);
  }
}
```

The above algorithm is implemented using threads. *Mutex1*, mutex2, *writeBlock, readBlock and writePending* are the semaphores used.

Program

```
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>
#include<stdlib.h>
void * reader (void *); void *
writer (void *);

sem_t mutex1, mutex2;
sem_t writeBlock; sem_t
readBlock; sem_t
writePending;  int
readCount =0;
int writeCount =0;
main(int argc, char * argv[])
{
   int i, j, k;

   int N_readers, N_writers;
   int readers_num[100], writers_num[100]; pthread_t
   tid_readers[100], tid_writers[100]; printf("Enter the
   number of readers:"); scanf("%d", &N_readers);

   printf("Enter the number of writers:");
   scanf("%d", &N_writers);

   for(k =0; k < N_readers; k++)
```

```
   readers_num[k] = k;
   for(k =0; k < N_writers; k++)
 writers_num[k] = k;
 if(sem_init(&mutex1, 0, 1) < 0)
  {
      perror("Could not init semaphore mutex1"); exit(1);
  }
 if(sem_init(&mutex2, 0, 1) < 0)
  {
      perror("Could not init semaphore mutex2"); exit(1);
  }
if(sem_init(&writeBlock, 0, 1) < 0)
  {
      perror("Could not init semaphore writeBlock"); exit(1);
  }

 if(sem_init(&readBlock, 0, 1) < 0)
  {
      perror("Could not init semaphore readBlock"); exit(1);
  }
 if(sem_init(&writePending, 0, 1) < 0)
  {
      perror("Could not init semaphore writePending"); exit(1);
```

```
  }
for( i =0; i < N_readers; i++)
  {
     if(pthread_create(&tid_readers[i], NULL, reader,  &readers_num[i]  ))
     {
        perror("could not create reader thread"); exit(1);
     }
  }
 for( j=0; j < N_writers; j++)
  {
    if(pthread_create(&tid_writers[j],  NULL,  writer,  &writers_num[j]))
    {
       perror("could not create writer thread"); exit(1);
    }
  }

  for (i =0; i < N_readers; i++)
  {
     pthread_join(tid_readers[i],   NULL);
  }

  for ( j=0; j < N_writers; j++)
  {
    pthread_join(tid_writers[j],   NULL);
  }
    sem_destroy(&mutex1);
```

```
  sem_destroy(&mutex2);
  sem_destroy(&readBlock);
  sem_destroy(&writeBlock);
  sem_destroy(&writePending);
}
void * reader (void* param)
{
    int i = *((int *) param);
     while(1)
      {
          sleep(1); if(sem_wait(&writePending) < 0)
         {
             perror("cannot decrement the semaphore writePending"); exit(1);
         }
        if(sem_wait(&readBlock) < 0)
         {
             perror("cannot decrement the semaphore readBlock"); exit(1);
         }
        if(sem_wait(&mutex1) < 0)
         {
             perror("cannot decrement the semaphore mutex1"); exit(1);
         }
         readCount = readCount + 1; if(readCount
         == 1)
```

```
{
    if(sem_wait(&writeBlock) < 0)
    {
        perror("cannot decrement the semaphore writeBlock"); exit(1);
    }
}
 if( sem_post(&mutex1) <0)
 {
    perror("cannot increment semaphore mutex1"); exit(1);
 }
 if( sem_post(&readBlock) <0)
 {
    perror("cannot increment semaphore readBlock"); exit(1);
 }
if( sem_post(&writePending) <0)
{
    perror("cannot increment semaphore writePending"); exit(1);
}
            // READ RESOURCES
   printf("READER %d is READING \n", i); sleep(1);

if(sem_wait(&mutex1) < 0)
 {
```

```
            perror("cannot decrement the semaphore mutex"); exit(1);
        }
        readCount = readCount - 1; if(readCount ==
        0)
         {
            if( sem_post(&writeBlock) < 0)
             {
                  perror("cannot increment semaphore mutex"); exit(1);
             }
          }
             if( sem_post(&mutex1) < 0)
             {
                  perror("cannot increment semaphore mutex"); exit(1);
             }
      }
}
void * writer (void * param)
{
    int i = *((int *) param);
    while(1)
     {
        sleep(1);
             if(sem_wait(&mutex2) < 0)
              {
                 perror("cannot decrement the semaphore mutex2");
```

```
    exit(1);
 }
 writeCount = writeCount +1;
 if(writeCount == 1)
 {
    if(sem_wait(&readBlock) < 0)
     {
        perror("cannot decrement the semaphore readBlock"); exit(1);
     }
 }

if( sem_post(&mutex2) < 0)
{
      perror("cannot increment semaphore mutex2"); exit(1);
}
if(sem_wait(&writeBlock) < 0)
 {
    perror("cannot decrement the semaphore writeBlock"); exit(1);
 }

 // WRITE RESOURCES
printf("WRITER %d IS WRITING \n", i);
if( sem_post(&writeBlock) < 0)
{
      perror("cannot increment semaphore writeBlock");
```

```
            exit(1);
        }
        if(sem_wait(&mutex2) < 0)
        {
            perror("cannot decrement the semaphore mutex2"); exit(1);
        }
        writeCount = writeCount - 1;
        if(writeCount == 0)
        {
        if( sem_post(&readBlock) < 0)
        {
            perror("cannot increment semaphore readBlock"); exit(1);
        }
        }
        if( sem_post(&mutex2) < 0)
        {
            perror("cannot increment semaphore mutex2"); exit(1);
        }
    }
}
```

Output

```
anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab$    ./expt5
```

Enter the number of readers:3

Enter the number of writers:2

READER 2 is READING
READER 0 is READING
READER 1 is READING
WRITER 0 IS WRITING
WRITER 1 IS WRITING
READER 2 is READING
READER 0 is READING
READER 1 is READING
WRITER 0 IS WRITING
WRITER 1 IS WRITING
READER 2 is READING
READER 0 is READING
READER 1 is READING
WRITER 0 IS WRITING
WRITER 1 IS WRITING
READER 2 is READING
READER 0 is READING
READER 1 is READING
WRITER 0 IS WRITING
WRITER 1 IS WRITING
READER 2 is READING
READER 0 is READING
READER 1 is READING
WRITER 0 IS WRITING
WRITER 1 IS WRITING
READER 2 is READING
READER 1 is READING
READER 0 is READING

Experiment 6

Inter-process Communication using Pipes, Message queues, and Shared Memory Aim: To communicate between two processes using pipes and message queues.

Description:

In this experiment a pipe is used to connect a client with the server. The client reads the name of a file from the standard input. It then writes the name of the file on to the pipe. The server reads the file from the read end of the pipe. After that, the server opens the file and reads the contents of the file line by line. Each line that it reads is sent to the client and written to the standard output. The data is transferred using the message structure. It has a header, which gives the length of the message and the type of the message which is a positive integer. The data is carried in an array within the message structure. The data flow through pipes is as shown in the figure.

PICTURE

Algorithm

1. Create pipe 1 (fd1[0], fd1[1])
2. Create pipe 2 (fd2[0], fd2[1])
3. Fork a child process
4. Parent closes read end of pipe 1 (fd1[0])
5. Parent closes write end of pipe 2 (fd2[1])
6. Child closes write end of pipe 1 (fd1[1])
7. Child closes read end of pipe 2 (fd2[0]) Parent

process (client process)

1. Read the filename from the standard input into the data portion of the message
2. Construct the message structure
3. Write the message to pipe 1
4. Read the message from the client on pipe2 and write to the standard output Child process

(server process)

1. Read the message sent by the client from pipe 1
2. Open the file
3. If there is an error, send an error message to the client
4. Otherwise, read each line from the file and send it as a message to the client on pipe 2

Program

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include<string.h> #include
“pipe.h”

void client(int, int); void
server(int, int);

int main(int argc, char ** argv)
 {
      int fd1[2], fd2[2]; // file descriptors for pipes pid_t
      childpid;

      // creation of pipes
      if(pipe(fd1) < 0)
       {
            perror(“pipe creation error”);
            exit(1);
       }

      if(pipe(fd2) < 0)
       {
            perror(“pipe creation error”);
            exit(1);
       }

      if((childpid = fork()) < 0)
       {
            perror(“fork error”);
            exit(1);
       }
       elseif(childpid == 0) // child process (server process)
        {
            close(fd2[0]); // child closes the read end of pipe 2 close(fd1[1]); //
            child closes the write end of pipe 1

            server(fd1[0], fd2[1]); exit(0);
       }
        else   // parent process (client process)
         {
              close( fd1[0]);
              close(fd2[1]);
```

```
        client(fd2[0], fd1[1]); if(waitpid(childpid,
        NULL,0) < 0)
        {
              perror("waitpid  error");
              exit(1);
         }
        exit(0);
      }
   }

void client(int readfd, int writefd)
  {
          int length;
          ssize_t n;
          struct message mesg;

           printf("Give the name of the file\n");
           fgets(mesg.message_data, MAXMESSAGEDATA, stdin); length =

           strlen(mesg.message_data);

          if(mesg.message_data[length -1] == '\n')
            length--;

          mesg.message_length = length; mesg.message_type =
          1;

           // write message to the pipe
           write( writefd, &mesg, MESGHDRSIZE + mesg.message_length);


         // read from pipe and write to the standard output while(1)
           {
              if(n = read(readfd, &mesg, MESGHDRSIZE)) == -1)
               {
                      perror("read error");
                       exit(1);
                }

              if( n!= MESGHDRSIZE)
               {
                     fprintf(stderr, "header size not same");
                     exit(1);
                }
```

```
                length = mesg.message_length; if(length ==
                0) break;

                n = read(readfd, mesg.message_data, length);
                write(STDOUT_FILENO, mesg.message_data, n);
              }
            }

void server(int readfd, int writefd)
 {
        FILE * fp;
        ssize_t n;
        struct message mesg; size_t
        length;

        mesg.message_type = 1;
        n = read(readfd, &mesg, MESGHDRSIZE);

        if( n!=MESGHDRSIZE)
         {
            fprintf(stderr, "header size not same \n"); exit(1);
         }

       length =mesg.message_length;
       n = read(readfd, mesg.message_data, length);
       mesg.message_data[n] = '\0';

       if( (fp = fopen(mesg.message_data, "r")) ==NULL)
         {
           snprintf(mesg.message_data + n, sizeof(mesg.message_data) -n, ":cant open\n");
           mesg.message_length = strlen(mesg.message_data);  write(writefd, &mesg,
           MESGHDRSIZE + mesg.message_length);
         }
          else
          {
              while(fgets(mesg.message_data,  MAXMESSAGEDATA,  fp)  !=  NULL)
                {
                    mesg.message_length = strlen(mesg.message_data); mesg.message_type = 1;
                    write(writefd, &mesg, MESGHDRSIZE + mesg.message_length);
                }

                fclose(fp);
          }
       mesg.message_length = 0;
```

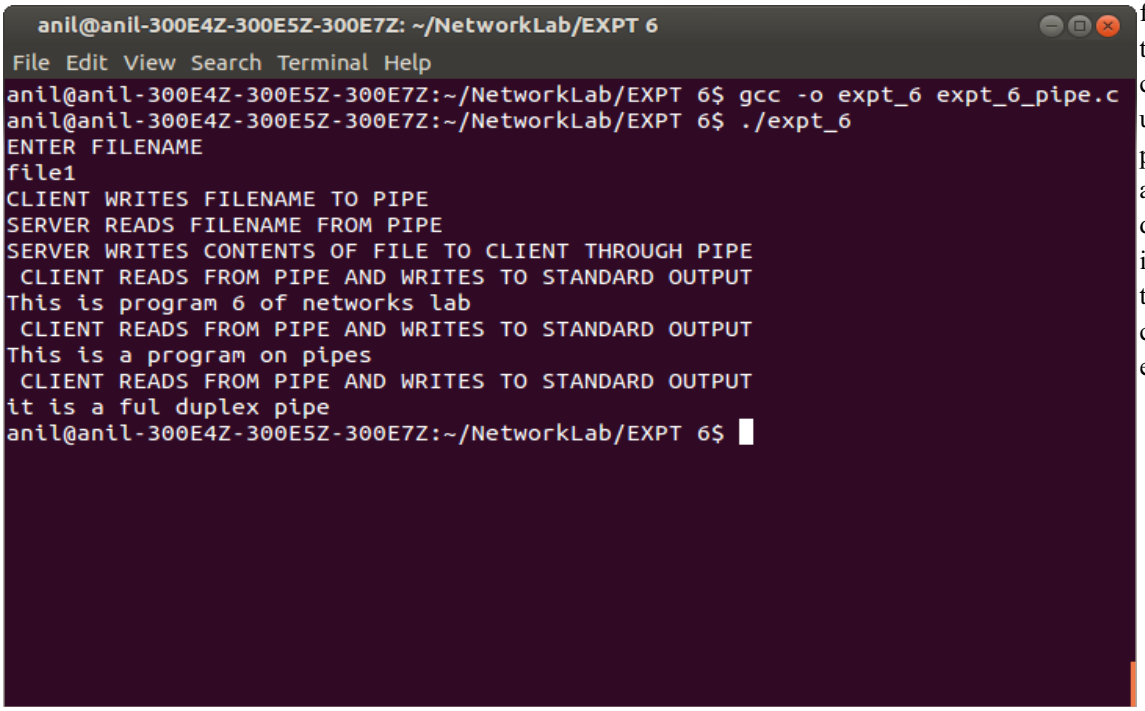
```
        write(writefd, &mesg, MESGHDRSIZE + mesg.message_length);
    }
```

Header file

```
#ifndef _PIPE #define
_PIPE
#include<stdio.h>
#include<limits.h>

#define MAXMESSAGEDATA(PIPE_BUF -2*sizeof(long)) // PIPE_BUF is the maximum
amount of data that can be written to a pipe

#define MESGHDRSIZE (sizeof(struct message) – MAXMESSAGEDATA)

struct message {
                long message_length;
                long message_type;
                char message_data[MAXMESSAGEDATA];
        };
#endif
```

Output

Using pipes send a filename from a client to a server. Open the file in the server and send the contents of the file to the client using pipe and displa y it on the consol e.



Experiment 7

Implementation of Client-Server communication using Socket Programming and TCP as transport layer protocol

Aim: Client sends a string to the server using tcp protocol. The server reverses the string and returns it to the client, which then displays the reversed string.

Description:

Steps for creating a TCP connection by a client are:

1. **Creation of client socket**

int socket(int domain, int type, int protocol);

This function call creates a socket and returns a socket descriptor. The domain parameter specifies a communication domain; this selects the protocol family which will be used for communication. These families are defined in <sys/socket.h>. In this program, the domain **AF_INET** is used. The socket has the indicated type, which specifies the communication semantics. **SOCK_STREAM** type provides sequenced, reliable, two-way, connection based byte streams. The **protocol** field specifies the protocol used. We always use 0. If the system call is a failure, a -1 is returned. The header files used are **sys/types.h** and **sys/socket.h**.

2. **Filling the fields of the server address structure.**

The socket address structure is of type **struct sockaddr_in**.

```
struct sockaddr_in {
        u_short sin_family;
        u_short sin_port;
        struct in_addr sin_addr;
        char sin_zero[8];          /*unused, always zero*/
};

struct in_addr {
        u_long s_addr;
};
```

The fields of the socket address structure are
sin_family which in our case is AF_INET
sin_port which is the port number where socket binds
sin_addr which is the IP address of the server machine The

header file that is to be used is **netinet/in.h**

Example

struct sockaddr_in servaddr; servaddr.sin_family =
AF_INET; servaddr.sin_port = htons(port_number);

Why htons is used ?. Numbers on different machines may be represented differently (big-endian machines and little-endian machines). In a little-endian machine the low order byte of an integer appears at the lower address; in a big-endian machine instead the low order byte appears at the higher address. Network order, the order in which numbers are sent on the internet is big-endian. It is necessary to ensure that the right representation is used on each machine. Functions are used to convert from host to network form before transmission- htons for short integers and htonl for long integers.

The value for servaddr.sin_addr is assigned using the following function

inet_pton(AF_INET, "IP_Address", &servaddr.sin_addr);

The binary value of the dotted decimal IP address is stored in the field when the function returns.

3. **Binding of the client socket to a local port**

This is optional in the case of client and we usually do not use the *bind* function on the client side.

4. Connection of client to the server

A server is identified by an IP address and a port number. The connection operation is used on the client side to identify and start the connection to the server.

int connect(int sd, struct sockaddr * addr, int addrlen);

sd – file descriptor of local socket
addr – pointer to protocol address of other socket addrlen – length
in bytes of address structure

The header files to be used are sys/types.h and sys/socket.h It returns 0

on sucess and -1 in case of failure.

5. **Reading from socket**
In the case of TCP connection reading from a socket can be done using the *read* system call

int read(int sd, char * buf, int length);

6. writing to a socket

In the case of TCP connection writing to a socket can be done using the *write* system call

int write(int sd, char * buf, int length);

7. **closing the connection**

The connection can be closed using the close system call

int close(int sd);

Steps for TCP Connection for server

1. **Creating a listening socket**

int socket(int domain, int type, int protocol);

This system call creates a socket and returns a socket descriptor. The *domain* field used is **AF_INET**. The socket type is **SOCK_STREAM**. The **protocol** field is 0. If the system call is a failure, a -1 is returned. Header files used are **sys/types.h** and **sys/socket.h**.

2. **Binding to a local port**

int bind(int sd, struct sockaddr * addr, int addrlen);

This call is used to specify for a socket the protocol port number where it will wait for messages. A call to *bind* is optional on the client side, but required on the server side. The first field is the *socket* descriptor of local socket. Second is a pointer to protocol address structure of this socket. The third is the length in bytes of the structure referenced by *addr.* This system call returns an integer. It is 0 for success and -1 for failure. The header files are **sys/types.h** and **sys/socket.h**.

3. **Listening on the port**

The listen function is used on the server in the connection oriented communication to prepare a socket to accept messages from clients.

int listen(int fd, int qlen);

fd – file descriptor of a socket that has already been bound

qlen – specifies the maximum number of messages that can wait to be processed by the server while the server is busy servicing another request. Usually it is taken as 5. The header files used are *sys/types.h* and *sys/socket.h.* This function returns 0 on success and -1 on failure.

4. **Accepting a connection from the client**

The accept function is used on the server in the case of connection oriented communication to accept a connection request from a client.

int accept(int fd, struct sockaddr * addressp, int * addrlen);

The first field is the descriptor of server socket that is listening. The second parameter *addressp* points to a socket address structure that will be filled by the address of calling client when the function returns. The third parameter *addrlen* is an integer that will contain the actual length of address structure of client. It returns an integer that is a descriptor of a new socket called the connection socket. Server sockets send data and read data from this socket. The header files used are *sys/types.h* and *sys/socket.h.*

Algorithm

Client

1. Create socket
2. Connect the socket to the server
3. Read the string to be reversed from the standard input and send it to the server Read the matrices from the standard input and send it to server using socket
4. Read the reversed string from the socket and display it on the standard output Read product matrix from the socket and display it on the standard output
5. Close the socket

Server

1. Create listening socket
2. bind IP address and port number to the socket
3. listen for incoming requests on the listening socket
4. accept the incoming request
5. connection socket is created when *accept* returns
6. Read the string using the connection socket from the client
7. Reverse the string
8. Send the string to the client using the connection socket
9. close the connection socket
10. close the listening socket

Client Program

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


int main( int argc, char *argv[])
{

struct sockaddr_in server; int
sd ;
char buffer[200];

if((sd   = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
    perror("Socket failed:");
    exit(1);
}

// server socket address structure initialisation

bzero(&server, sizeof(server) ); server.sin_family =
AF_INET; server.sin_port = htons(atoi(argv[2]));
inet_pton(AF_INET,  argv[1],  &server.sin_addr);

if(connect(sd, (struct sockaddr *)&server, sizeof(server))< 0)
{
    perror("Connection failed:");
    exit(1);
}

fgets(buffer, sizeof(buffer), stdin);
buffer[strlen(buffer) - 1] = '\0';

write (sd,buffer, sizeof(buffer)); read(sd,buffer,
sizeof(buffer));

printf("%s\n",  buffer);
```

```
close(fd);

}
```

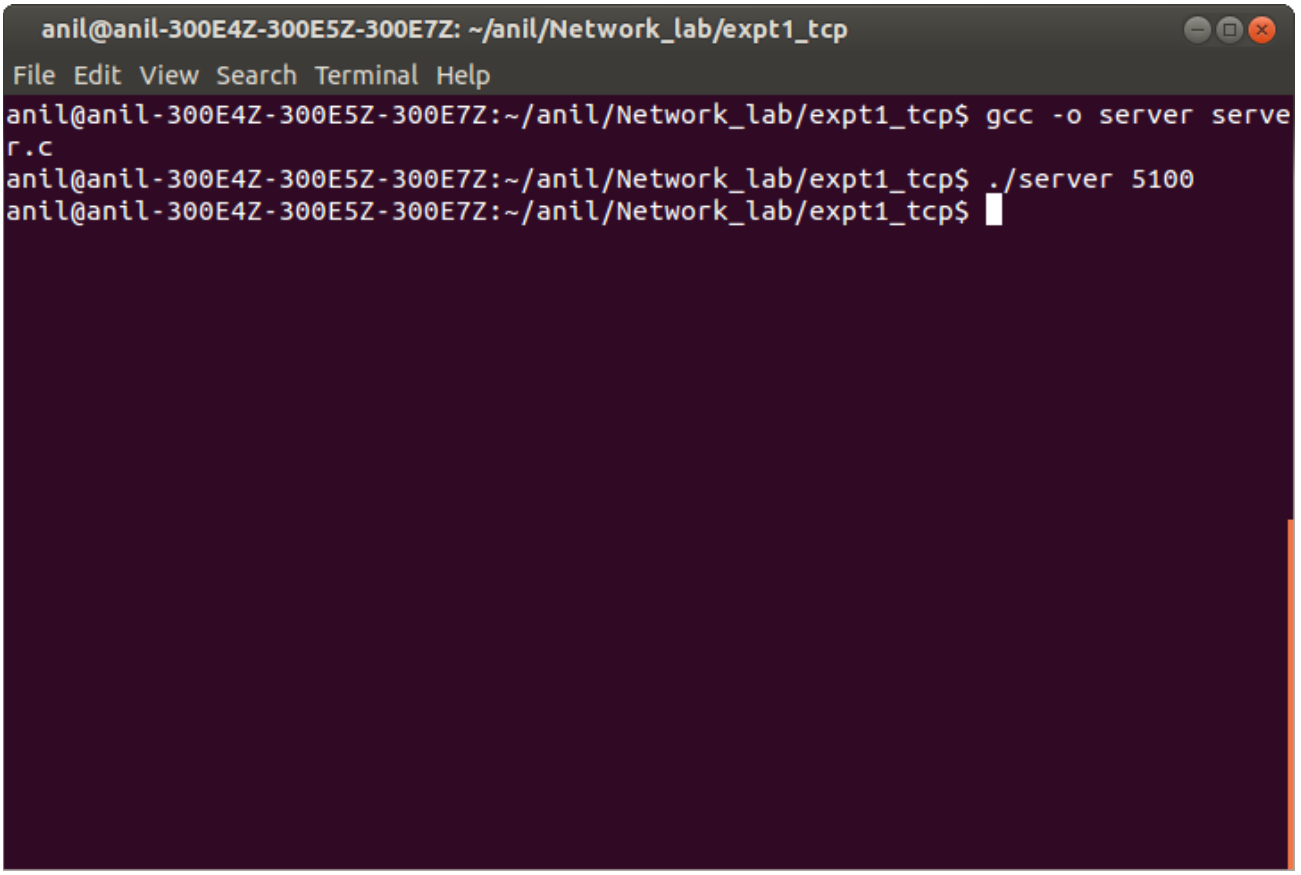
Server Program

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


int main( int argc, char *argv[])
{

struct sockaddr_in server, cli; int
cli_len;
int sd, n, i, len; int
data, temp;

char buffer[100];

if((sd   = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
    perror("Socket failed:");
    exit(1);
}



// server socket address structure initialisation

bzero(&server, sizeof(server) ); server.sin_family =
AF_INET; server.sin_port = htons(atoi(argv[1]));
server.sin_addr.s_addr = htonl(INADDR_ANY);

if(bind(sd, (struct sockaddr*)&server, sizeof(server)) < 0)
{
    perror("bind failed:");
    exit(1);
}
```

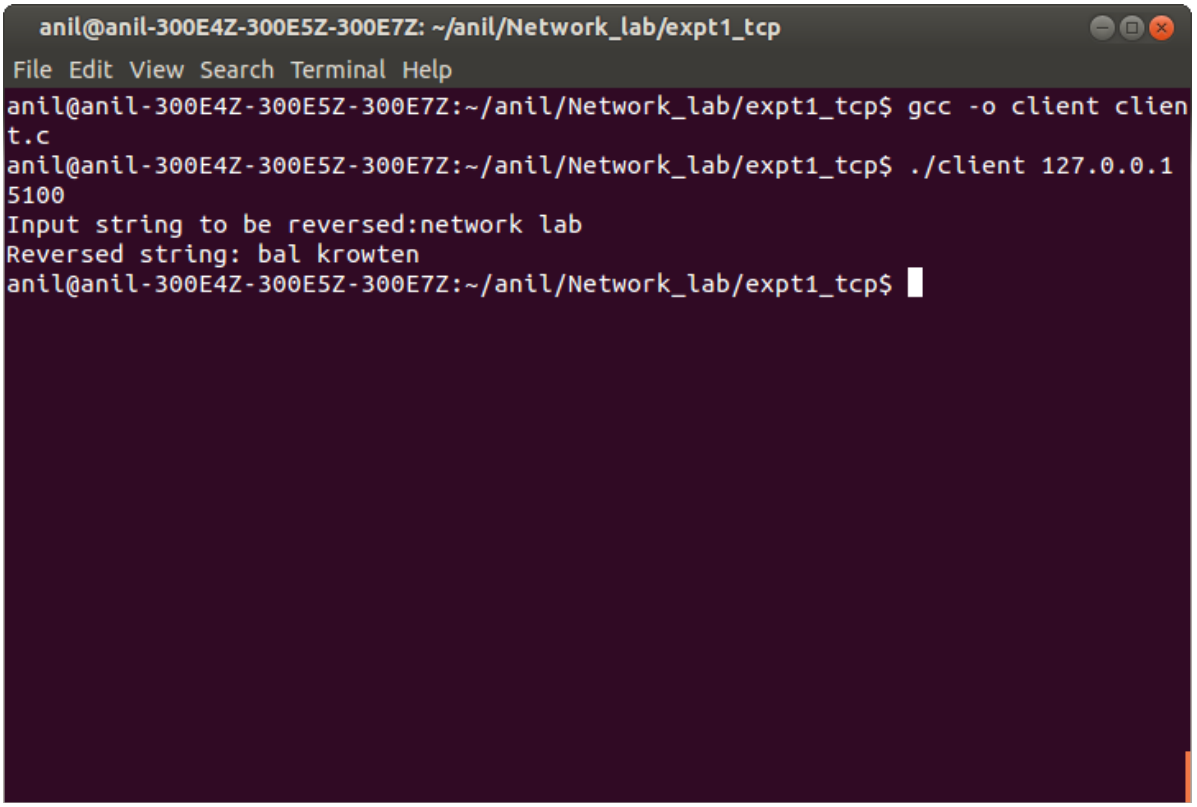
```
listen(sd,5);

if((data =  accept(sd , (struct sockaddr *) &cli, &cli_len)) < 0)
{
    perror("accept failed:");
    exit(1);
}

read(data,buffer,  sizeof(buffer));

len = strlen(buffer);
for( i =0; i<= len/2; i++)
 {
   temp =  buffer[i];
   buffer[i] = buffer[len - 1-i];
   buffer[len-1-i] = temp;
}

write (data,buffer, sizeof(buffer));

close(data); close(sd);
}
```

Output

Server



Client



Experiment 8

Implementation of Client-Server communication using Socket Programming and UDP as transport layer protocol

Aim: Client sends two matrices to the server using udp protocol. The server multiplies the matrices and sends the product to the client, which then displays the product matrix.

Description:

Steps for transfer of data using UDP

1. **Creation of UDP socket**

The function call for creating a UDP socket is

int socket(int domain, int type, int protocol);

The *domain* parameter specifies a communication domain; this selects the protocol family which will be used for communication. These families are defined in <sys/socket.h>. In this program, the domain **AF_INET** is used. The next field *type* has the value **SOCK_DGRAM.** It supports datagrams (connectionless, unreliable messages of a fixed maximum length). The *protocol* field specifies the protocol used. We always use 0. If the socket function call is successful, a socket descriptor is returned. Otherwise -1 is returned. The header files necessary for this function call are **sys/types.h** and **sys/socket.h**.

2. **Filling the fields of the server address structure.**

The socket address structure is of type **struct sockaddr_in**.

```
struct sockaddr_in {
        u_short sin_family;
        u_short sin_port;
        struct in_addr sin_addr;
        char sin_zero[8];          /*unused, always zero*/
};

struct in_addr {
        u_long s_addr;
};
```

The fields of the socket address structure are
sin_family which in our case is AF_INET
sin_port which is the port number where socket binds
sin_addr is used to store the IP address of the server machine and is of type **struct in_addr**

The header file that is to be used is **netinet/in.h**

The value for **servaddr.sin_addr** is assigned using the following function

inet_pton(AF_INET, "IP_Address", &servaddr.sin_addr);

The binary value of the dotted decimal IP address is stored in the field when the function returns.

3. **Binding of a port to the socket in the case of server**

This call is used to specify for a socket the protocol port number where it will wait for messages. A call to bind is optional in the case of client and compulsory on the server side.

int bind(int sd, struct sockaddr* addr, int addrlen);

The first field is the socket descriptor. The second is a pointer to the address structure of this socket. The third field is the length in bytes of the size of the structure referenced by *addr*. The header files are **sys/types.h** and **sys/socket.h**. This function call returns an integer, which is 0 for success and -1 for failure.

4. **Receiving data**

ssize_t recvfrom(int s, void * buf, size_t len, int flags, struct sockaddr * from, socklen_t * fromlen);

The *recvfrom* calls are used to receive messages from a socket, and may be used to receive data on a socket whether or not it is connection oriented. The first parameter *s* is the socket descriptor to read from. The second parameter *buf* is the buffer to read information into. The third parameter *len* is the maximum length of the buffer. The fourth parameter is *flag.* It is set to zero. The fifth parameter *from* is a pointer to **struct sockaddr** variable that will be filled with the IP address and port of the orginating machine. The sixth parameter *fromlen* is a pointer to a local int variable that should be initialized to **sizeof(struct sockaddr)**. When the function returns, the integer variable that *fromlen* points to will contain the actual number of bytes that is contained in the socket address structure. The header files required are **sys/types.h** and **sys/socket.h**. When the function returns, the number of bytes received is returned or -1 if there is an error.

5. **Sending data**

sendto- sends a message from a socket

ssize_t sendto(int s, const void * buf, size_t len, int flags, const struct sockaddr * to, socklen_t tolen);

The first parameter *s* is the socket descriptor of the sending socket. The second parameter *buf* is the array which stores data that is to be sent. The third parameter *len* is the length of that data in bytes. The

fourth parameter is the *flag* parameter. It is set to zero. The fifth parameter *to* points to a variable that contains the destination IP address and port. The sixth parameter *tolen* is set to **sizeof(struct sockaddr)**. This function returns the number of bytes actually sent or -1 on error. The header files used are **sys/types.h** and **sys/socket.h**.

Algorithm

Client

1. Create socket
2. Read the matrices from the standard input and send it to server using socket
3. Read product matrix from the socket and display it on the standard output
4. Close the socket

Server

1. Create socket
2. bind IP address and port number to the socket
3. Read the matrices socket from the client using socket
4. Find product of matrices
5. Send the product matrix to the client using socket
6. close the socket

Client program

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<fcntl.h>
#include<stdlib.h>
main(int argc, char * argv[])
{
   int i,j,n;
   int sock_fd;

   struct sockaddr_in servaddr;
```

```
int matrix_1[10][10], matrix_2[10][10], matrix_product[10][10]; int size[2][2];

int num_rows_1, num_cols_1, num_rows_2, num_cols_2; if(argc != 3)
{
  fprintf(stderr, "Usage: ./client IPaddress_of_server port\n"); exit(1);
}

printf("Enter the number of rows of first matrix\n"); scanf("%d",
&num_rows_1);

printf("Enter the number of columns of first matrix\n"); scanf("%d",
&num_cols_1);

printf("Enter the values row by row one on each line\n" ); for ( i = 0;

i < num_rows_1; i++)
for( j=0; j<num_cols_1; j++)
{
   scanf("%d", &matrix_1[i][j]);
}

size[0][0] = num_rows_1; size[0][1] =
num_cols_1;

printf("Enter the number of rows of second matrix\n"); scanf("%d",
&num_rows_2);

printf("Enter the number of columns of second matrix\n"); scanf("%d",
&num_cols_2);

if( num_cols_1 != num_rows_2)
{
   printf("MATRICES CANNOT BE MULTIPLIED\n");
   exit(1);
}

printf("Enter the values row by row one on each line\n");

for (i = 0; i < num_rows_2; i++)
for(j=0; j<num_cols_2; j++)
{
```

```
    scanf("%d", &matrix_2[i][j]);
 }

 size[1][0] = num_rows_2; size[1][1]
 = num_cols_2;

if((sock_fd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
 {
      printf("Cannot create socket\n"); exit(1);
 }

bzero((char*)&servaddr,  sizeof(servaddr));

servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(atoi(argv[2]));
inet_pton(AF_INET,  argv[1],  &servaddr.sin_addr);


// SENDING MATRIX  WITH SIZES OF MATRICES 1 AND 2


n = sendto(sock_fd, size, sizeof(size),0, (struct sockaddr*)&servaddr, sizeof(servaddr)); if( n < 0)
{
   perror("error in matrix 1  sending"); exit(1);
}

// SENDING MATRIX 1
n = sendto(sock_fd, matrix_1, sizeof(matrix_1),0, (struct sockaddr*)&servaddr, sizeof(servaddr));

if( n < 0)
{
   perror("error in matrix 1  sending"); exit(1);
}

// SENDING MATRIX 2


n = sendto(sock_fd, matrix_2, sizeof(matrix_2),0, (struct sockaddr*)&servaddr, sizeof(servaddr)); if( n < 0)
{
   perror("error in matrix 2  sending"); exit(1);
```

```
  }

  if((n=recvfrom(sock_fd, matrix_product, sizeof(matrix_product),0, NULL, NULL)) == -1)
  {
    perror("read error from server:"); exit(1);
 }

printf("\n\nTHE PRODUCT OF MATRICES IS \n\n\n");

for( i=0; i < num_rows_1; i++)
{
for( j=0; j<num_cols_2; j++)
{
   printf("%d ",matrix_product[i][j]);
}
printf("\n");
}

close(sock_fd);

}
```

Server Program

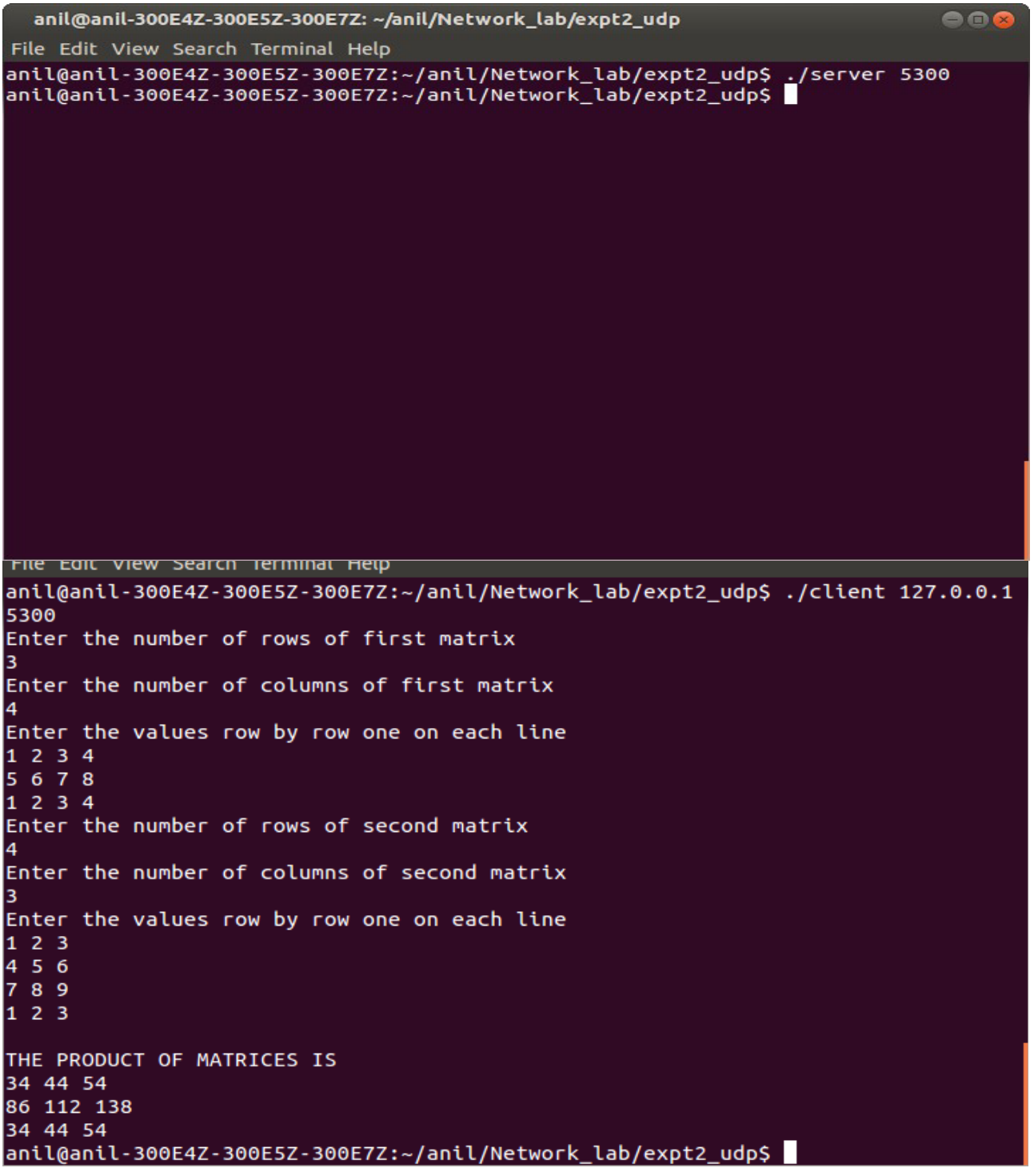
```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<fcntl.h>
#include<stdlib.h>


main(int argc, char * argv[])
{

   int n;
   int sock_fd;
   int i,j,k;

   int row_1, row_2, col_1, col_2; struct

   sockaddr_in servaddr, cliaddr;
```

```
int len = sizeof(cliaddr);

int matrix_1[10][10], matrix_2[10][10], matrix_product[10][10]; int size[2][2];

if(argc != 2)
{
  fprintf(stderr, "Usage: ./server  port\n"); exit(1);
}



if((sock_fd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
 {
       printf("Cannot create socket\n"); exit(1);
 }

bzero((char*)&servaddr,   sizeof(servaddr));

servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(atoi(argv[1]));
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);

if(bind(sock_fd, (struct sockaddr*)&servaddr, sizeof(servaddr)) < 0)
{
  perror("bind failed:");
  exit(1);
}

// MATRICES RECEIVE

if((n = recvfrom(sock_fd, size, sizeof(size), 0, (struct sockaddr *)&cliaddr, &len)) == -1)
{
    perror("size  not received:"); exit(1);
}

// RECEIVE MATRIX 1

if((n = recvfrom(sock_fd, matrix_1, sizeof(matrix_1), 0, (struct sockaddr *)&cliaddr, &len)) == -1)
{
    perror("matrix 1 not received:"); exit(1);
}
```

```
// RECEIVE MATRIX 2

if((n = recvfrom(sock_fd, matrix_2, sizeof(matrix_2), 0, (struct sockaddr *)&cliaddr, &len)) == -1)
{
   perror("matrix 2 not received:");
   exit(1);
}

row_1 = size[0][0];
col_1 = size[0][1];
row_2 = size[1][0];
col_2 = size[1][1];

for (i =0; i < row_1 ; i++) for
(j =0; j <col_2;  j++)
{
   matrix_product[i][j] = 0;
}

for(i =0; i< row_1 ; i++)
for(j=0; j< col_2 ; j++) for
(k=0; k < col_1; k++)
{
   matrix_product[i][j] += matrix_1[i][k]*matrix_2[k][j];
 }

n = sendto(sock_fd, matrix_product, sizeof(matrix_product),0, (struct sockaddr*)&cliaddr, sizeof(cliaddr));

if( n < 0)
  {
     perror("error in matrix product  sending"); exit(1);
  }
close(sock_fd);
}
```

Output

Server



Experiment 9

Implementation of a multi user chat server using TCP as transport layer protocol

Aim: To implement a chat server so that multiple users can chat simultaneously

Description

To implement chat server the *select* function is used. It is a function called by a process. When a process calls this function , the process goes to sleep and wakes up only when one or more events occur or when a specified amount of time has passed.

int select (int maxfdp1, fd_set * readset, fd_set * writeset, fd_set * exceptset, const struct timeval * timeout);

The header files required are <sys/select.h> and <sys/time.h>
The function returns -1 on error, count of ready descriptors and 0 on timeout. If we want the kernel to wait for as long as one descriptor becomes ready then we specify the timeout argument as a null pointer. *readset, writeset* and *exceptset* specify the descriptors that we want the kernel to test for reading, writing and exception conditions. *select* uses descriptor sets. Each set is an array of integers. Each bit in an integer corresponds to a descriptor. In a 32 bit integer, the first element of the array represents descriptors from 0 to 31. Second element of the array represents descriptors from 32 to 63 and so on. We have four macros for the *fd_set* data type.

void FD_ZERO(fd_set * fdset); // clears all bits in *fd_set*
void FD_SET(int fd, fd_set * fdset); // turns on the bit for *fd* in *fdset* void
FD_CLR(int fd, fd_set * fdset); // turns off the bit for *fd* in *fdset* int FD_ISSET(int
fd, fd_set * fdset); // is the bit for *fd* on in *fdset*

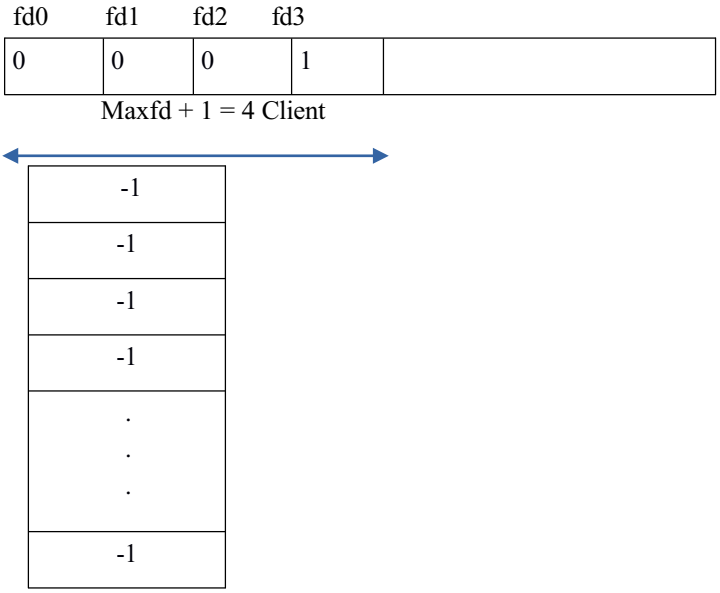
Usage:

fd_set rset;
FD_ZERO(&rset); // all bits off FD_SET(1,
&rset); // turn on bit for *fd* 1 FD_SET(4, &rset); //
turn on bit for *fd* 4

Initially you have to initialize the set. If we are not interested in a condition we can set that argument of *select* to NULL, i.e; for *readset*, *writeset* or *exceptset*.

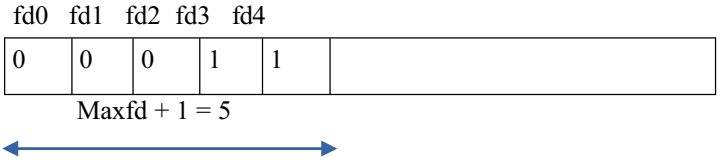
maxfdp1 argument specifies the number of descriptors to be tested. It is equal to value of max descriptor to be tested plus one. This is because descriptor starts with 0. When we call *select* we specify the values of the descriptors that we are interested in and on return the result indicates which descriptors are ready. We turn on all the bits in the descriptor sets that we are interested in. On return of the call, descriptors that are not ready will have the corresponding bit cleared in the descriptor set.

We can use *select* to create a server which can handle clients without forking process for each client. rset



Descriptors 0, 1, 2 are respectively for standard input, output and error. Next available descriptor is 3 which is set for listening socket. *Client* is an array that contains the connected socket descriptor for each client. All elements are initialized to -1. The first non zero for descriptor set is that for the listening socket. When the first client establishes a connection with the server, the listening descriptor becomes readable and server calls *accept*. The new connected descriptor will be 4. The arrays are updated.

rset

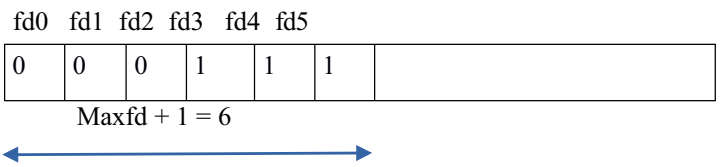


Client

4
-1
-1
-1
. . .
-1

Now if a second client connects

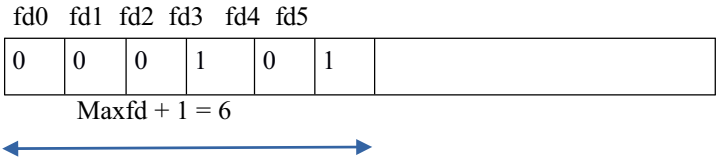
rset



Client

4
5
-1
-1
. . .
-1

If the first client terminates connection by sending a FIN segment, descriptor 4 becomes readable and *read* returns 0. This socket is closed by the server and data structures are updated. rset



Client

-1
5
-1
-1
. . .
-1

The descriptor 4 in *rset* is set to zero. When clients arrive the connected socket descriptor is placed in the first available entry, i.e; first entry with value equal to -1.

Program

client

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

main(int argc, char * argv[])
```

```
{
  int i,j,n;
  int sock_fd, max_fd, nready, fd[2]; char

  buffer[100], line[100];

  struct sockaddr_in servaddr; fd_set
  rset;

  if(argc != 3)
  {
    fprintf(stderr, "Usage: ./client IPaddress_of_server port\n"); exit(1);
  }
  if((sock_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
   {
        printf("Cannot create socket\n"); exit(1);
   }

  bzero((char*)&servaddr, sizeof(servaddr));
  bzero(line, sizeof(line));

  servaddr.sin_family = AF_INET;
  servaddr.sin_port = htons(atoi(argv[2]));
  inet_pton(AF_INET,  argv[1],  &servaddr.sin_addr);

  if(connect(sock_fd, (struct sockaddr *)&servaddr, sizeof(servaddr))< 0)
  {
    perror("Connection failed:");
    exit(1);
  }

  fd[0] = 0;
  fd[1] = sock_fd; for(;
  ; )
  {
    FD_ZERO(&rset); FD_SET(0,
    &rset); FD_SET(sock_fd,
    &rset); bzero(line,
    sizeof(line));

    max_fd = sock_fd;

    nready = select(max_fd + 1, &rset, NULL, NULL, NULL);
```

```
      for ( j = 0; j <2 ; j++)
      {
        if(FD_ISSET(fd[j],  &rset))
        {

          n =   read(fd[j], line, sizeof(line)); if(j ==


          0)
           {
              n = write(fd[j+1],  line, strlen(line));

           }
          else
           {
              printf("%s \n", line);
           }

           if(--nready == 0)
           break;
       }
     }
   }
}
```

server

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

main(int argc, char * argv[])

{
int n, i, maxi, max_fd, k;
int sock_fd, listen_fd, connfd, client_no;
int nready, num_q,  client[100], chat[100], conn[1000]; char

line[1000], buffer[1000];

fd_set rset, allset;
```

```
struct sockaddr_in servaddr, cliaddr;

int len = sizeof(cliaddr);
bzero(line, sizeof(line));

client_no = 0;

if(argc != 2)
{
      fprintf(stderr, "Usage: ./server  port\n"); exit(1);
}

if((listen_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
      printf("Cannot create socket\n"); exit(1);
}

bzero((char*)&servaddr,   sizeof(servaddr));

servaddr.sin_family = AF_INET; servaddr.sin_port
= htons(atoi(argv[1]));
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);

if(bind(listen_fd, (struct sockaddr*)&servaddr, sizeof(servaddr)) < 0)
{
     perror("bind failed:");
     exit(1);
}

listen(listen_fd, num_q);
max_fd = listen_fd;

maxi = -1;

for(i=0; i < 100; i++)
{
    client[i] = -1;
    chat[i] = -1;
}
FD_ZERO(&allset);

FD_SET(listen_fd, &allset);
```

```
for(; ;)
{
   rset = allset;
   nready = select(max_fd + 1, &rset, NULL, NULL, NULL);

   if(FD_ISSET(listen_fd,  &rset))
   {
      if((connfd = accept(listen_fd, (struct sockaddr *) &cliaddr,  &len))<0)
       {
           perror("accept failed");
           exit(1);
       }

       chat[++client_no] = connfd; conn[connfd] =
       client_no;

       k = client_no;
       sprintf(buffer, "Client %d has joined chat\n", k);

       while(k > 0 )
         {
            if(chat[k] >0)
            n =   write(chat[k--], buffer, strlen(buffer));
         }
   }

   for( i = 0; i < 100; i++)
    {
       if( client[i] < 0)
        {
           client[i] = connfd; break;
        }
    }

     FD_SET(connfd, &allset);
     if(connfd > max_fd) max_fd =
     connfd;

      if( i > maxi) maxi
       = i;

      if(--nready <= 0)
        continue;
}
```

```
for( i =0; i <=maxi; i++)
{
   bzero(line, sizeof(line));

   if((sock_fd = client[i]) <0) continue;

   if( FD_ISSET(sock_fd, &rset))
    {
       n= read(sock_fd, line, sizeof(line)); if( n
       == 0)
        {
           close(sock_fd);
           FD_CLR(sock_fd, &allset);
           client[i] = -1;

           bzero(buffer, sizeof(buffer));
           sprintf(buffer, "Client %d has left chat\n", conn[sock_fd]);

           k = client_no;
           while(k > 0 )
            {
               if(chat[k] > 0)
               n =   write(chat[k--], buffer, strlen(buffer));
             }
         }
        else
         {
            if(chat[line[0] - 48] > 0)  write(chat[line[0] - 48],
            line, strlen(line));
          }

         if(--nready <= 0)
         break;
     }
}
}

}
```

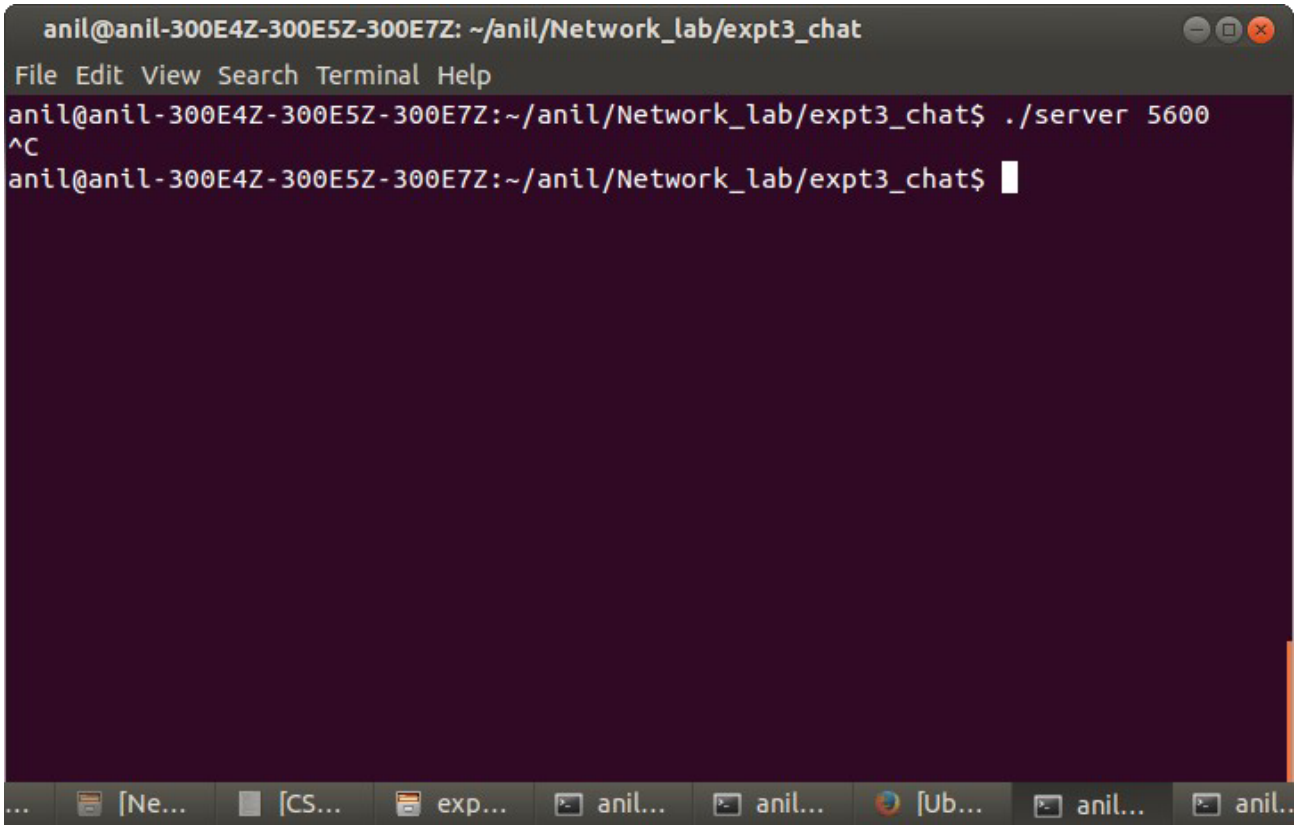
Output

The server is started first. Each client then starts. The clients are numbered consecutively. If client x wants to send message to client y, client x writes

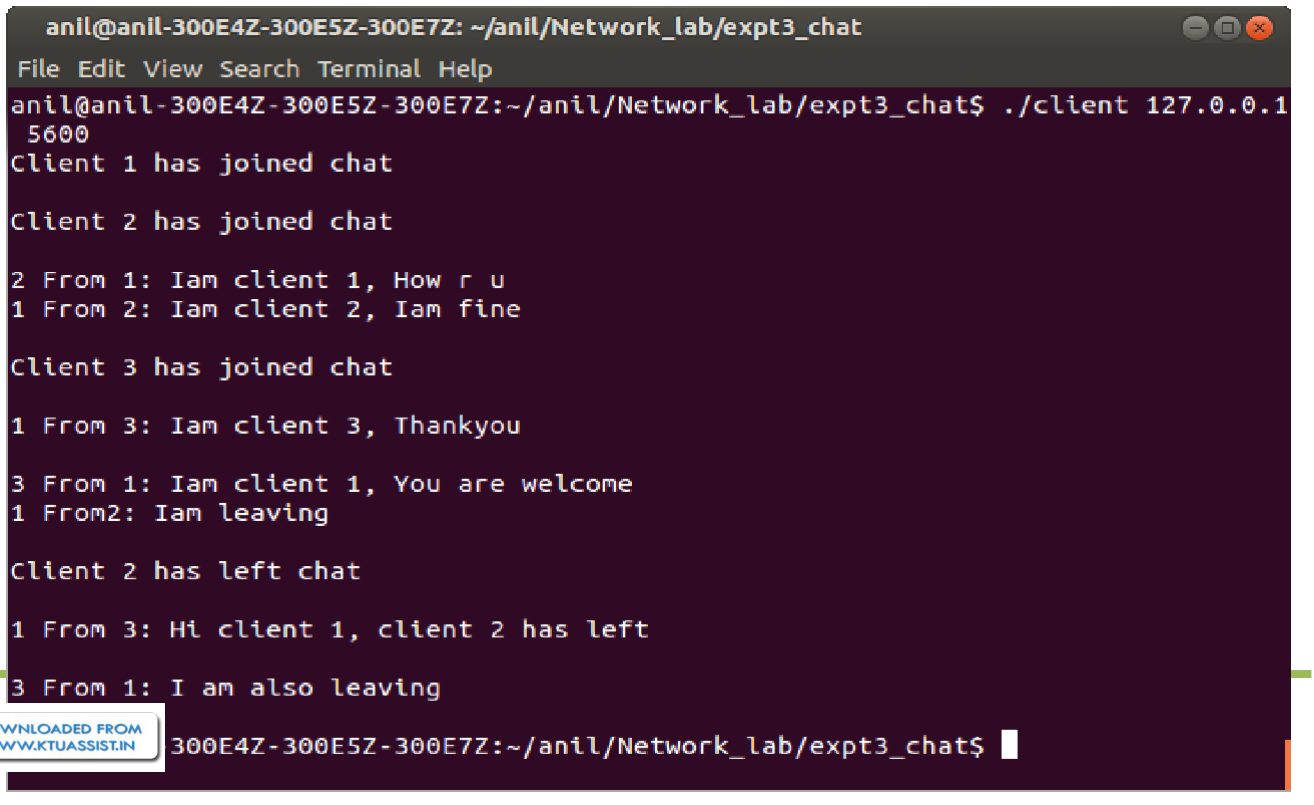
y From x: “contents of message”

The screenshots show 3 clients chatting with each other through the server running on port 5500.

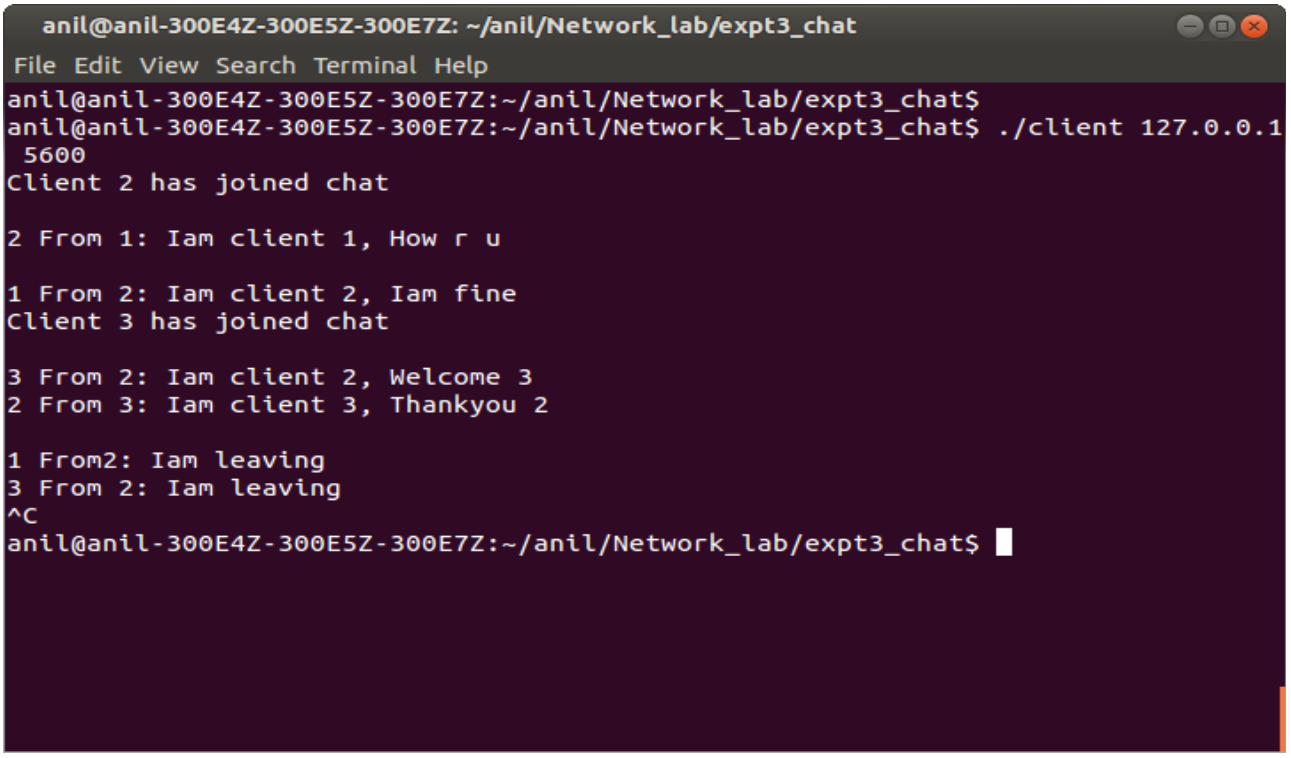
Server



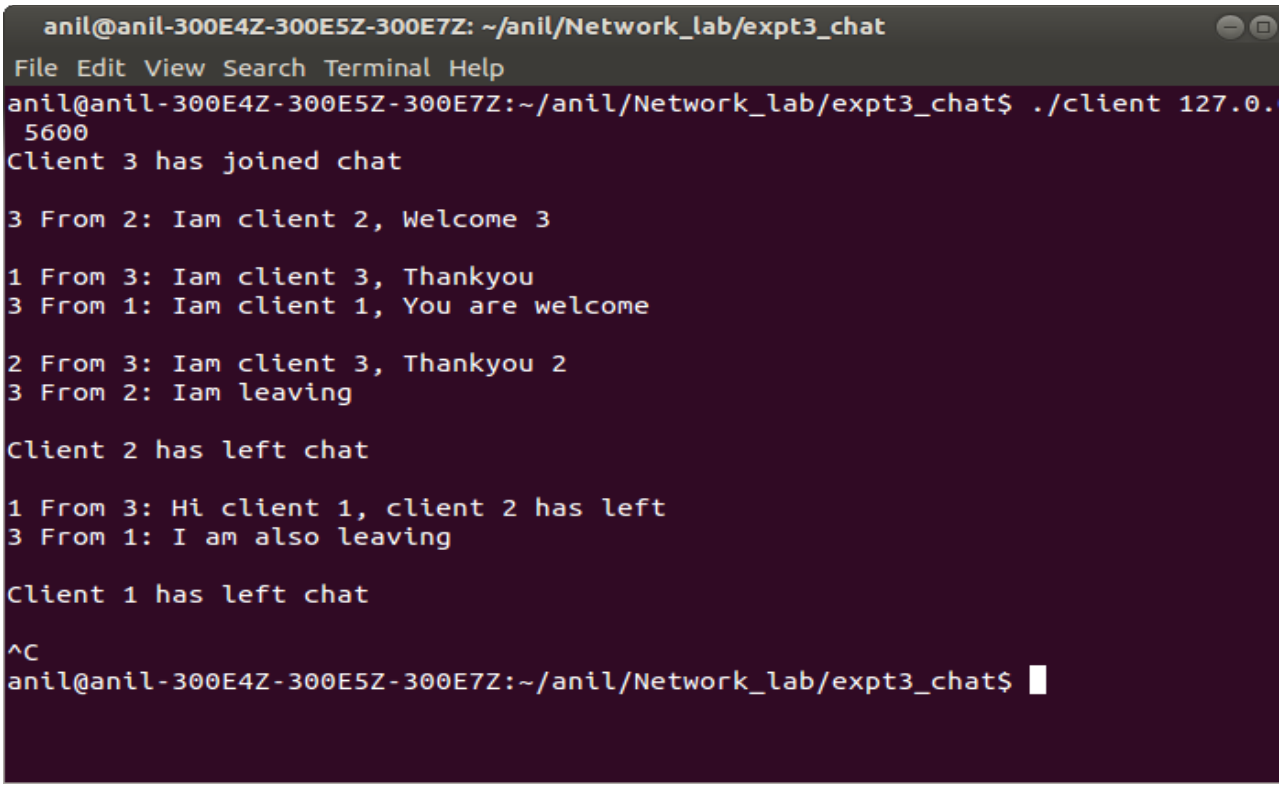
Client 1



Client 2



Client 3



Experiment 10

Implementation of concurrent time server using UDP

Aim: Client sends a time request to the server, server sends its system time back to the client.

Description:

A concurrent server handles multiple clients at the same time. The simplest technique for a concurrent server is to call the Unix fork function, i.e; creating one child process for each client.
The current time and date is obtained by the library function *time* which returns the number of seconds since the Unix Epoch: 00:00:00 January 1, 1970, UTC (Coordinated Universal Time). *ctime* is a function that converts this integer value into a human readable string.

Algorithm:

Client

1. Create UDP socket
2. Send a request for time to the server
3. Receive the time from the server
4. Display the result

Server

1. Create a UDP socket
2. bind the port and address to the socket
3. while (1)
    - 3.1 Receive time request from the client
    - 3.2 create a child process using fork If child process
        - 3.2.1 Use *time* and *ctime* functions to find out cuurent time
        - 3.2.2 Send the time as a string to the client
        - 3.2.3 Exit
4. end of while

Program

Client

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

main(int argc, char * argv[])
{
    int i,j,n;
    int sock_fd;
    struct sockaddr_in servaddr; char
    buff[100];

     if(argc != 3)
     {
         fprintf(stderr, "Usage: ./client IPaddress_of_server port\n"); exit(1);
    }

    if((sock_fd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
    {
         printf("Cannot create socket\n"); exit(1);
    }

  bzero((char*)&servaddr,  sizeof(servaddr));

  servaddr.sin_family = AF_INET;
  servaddr.sin_port = htons(atoi(argv[2]));

   inet_pton(AF_INET, argv[1], &servaddr.sin_addr);

  n = sendto(sock_fd, "", 1,0, (struct sockaddr*)&servaddr, sizeof(servaddr)); if( n < 0)
  {
       perror("error in sending"); exit(1);
  }

  if((n=recvfrom(sock_fd, buff, sizeof(buff),0, NULL, NULL)) == -1)
   {
```

```
        perror("read error from server:"); exit(1);
    }

    printf(" the current time of the system is %s\n", buff);
}
```

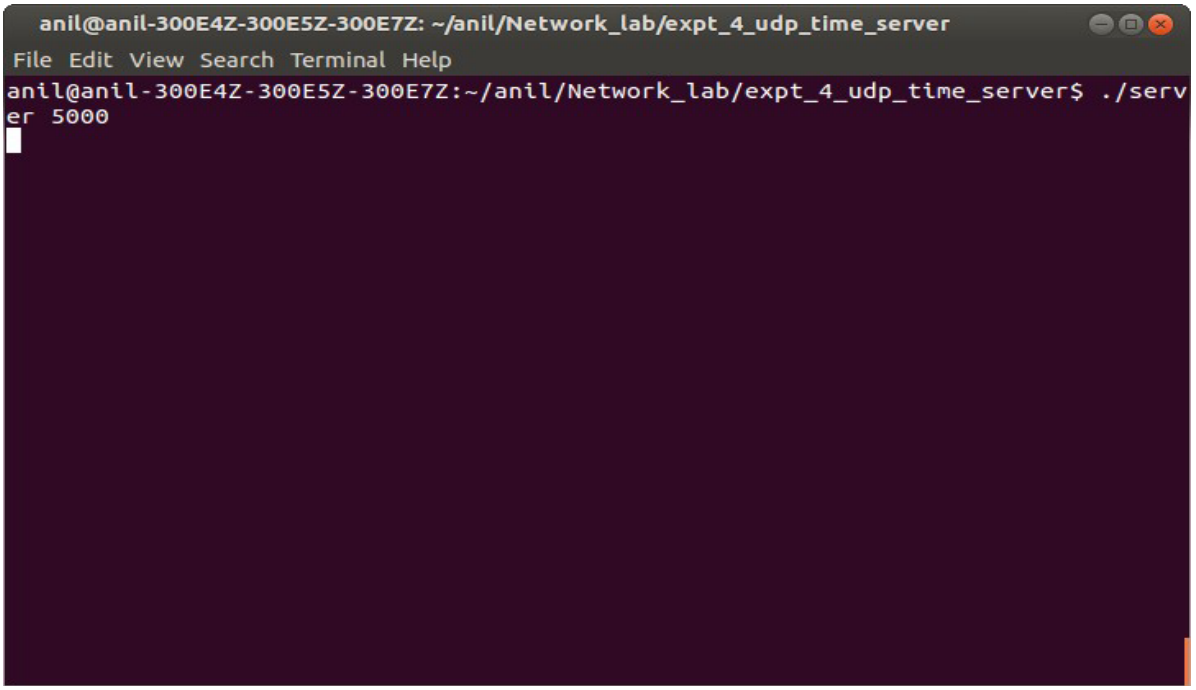
Server

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

main(int argc, char * argv[])
{
    int n;
    int sock_fd;
    int i,j,k;
    int childpid;
    char buffer[100];

    time_t curtime;
    struct sockaddr_in servaddr, cliaddr; int len =
    sizeof(cliaddr);

    if(argc != 2)
    {
       fprintf(stderr, "Usage: ./server  port\n"); exit(1);
    }

    if((sock_fd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
    {
        printf("Cannot create socket\n"); exit(1);
    }

   bzero((char*)&servaddr,   sizeof(servaddr));

  servaddr.sin_family = AF_INET;
```

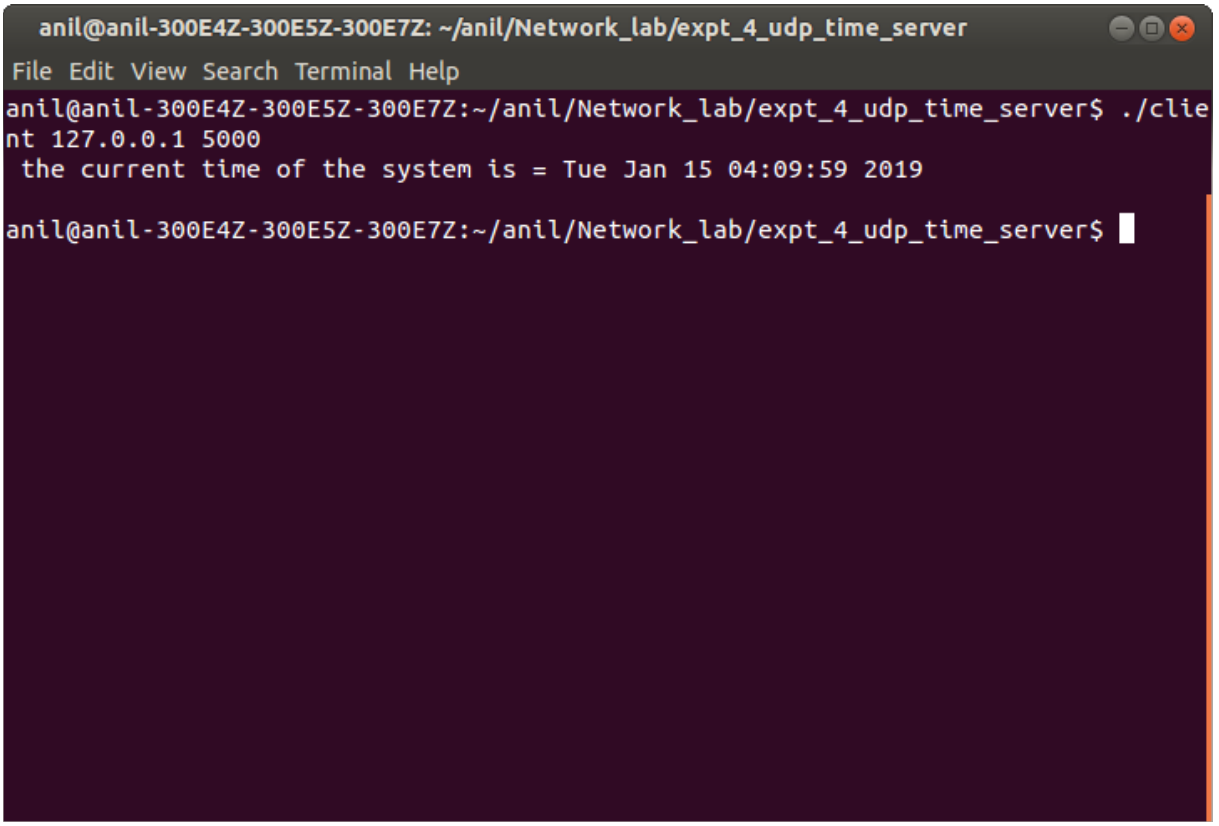
```
    servaddr.sin_port = htons(atoi(argv[1])); servaddr.sin_addr.s_addr =
  htonl(INADDR_ANY);

   if(bind(sock_fd, (struct sockaddr*)&servaddr, sizeof(servaddr)) < 0)
   {
        perror("bind failed:");
        exit(1);


  }

while(1)
{
   if((n = recvfrom(sock_fd,buffer , sizeof(buffer), 0, (struct sockaddr *)&cliaddr, &len)) == -1)
    {
         perror("size  not received:");
         exit(1);
    }

    childpid = fork();
    if(childpid == 0)
       {
          time(&curtime);
          sprintf( buffer, "= %s", ctime(&curtime));
       n = sendto(sock_fd, buffer, sizeof(buffer),0, (struct sockaddr*)&cliaddr, sizeof(cliaddr)); if( n < 0)
         {
           perror("error in  sending");
           exit(1);
         }
        exit(1);
    }
}

}
```

Output

Server



client



Experiment 11

Implemention and simulation of algorithm for Distance vector routing protocol. Aim: To implement and simulate algorithm for Distance vector routing protocol

Description:

This algorithm is iterative, and distributed. Each node receives information fromits directly attached neighbours, performs some calculations and returns the result the result to its neighbouring nodes. This process of updating the information goes on until there is no exchange of information between neighbours.

Algorithm:

(adapted from Computer Networking – A top down approach by Kurose and Rose) Bellman Ford

algoithm is applied.

Let $d_x(y)$ be the cost of the least cost path from node x to node y. Then Bellman Ford equation states that

$$d_x(y) = \min_v \{ c(x,v) + d_v(y) \}$$

where *v* is a neighbour of node *x.* $d_v(y)$ is the cost of the least cost path from *v* to *y*. c(x,v) is the cost from *x* to neighbour *v.* The least cost path has a value equal to minimum of $c(x,v) + d_v(y)$ over all its neighbours *v*. The solution of Bellman Ford equation provides entries in node x's forwarding table.

Distance vector (DV) algorithm At

each node *x*

Initialization:

for all destinations *y* in N:

$D_x(y)$ = c(x,y) /* if *y* is not a neighbour of *x,* then c(x,y) = ∞ */

for each neighbour *w*,

send distance vector $D_x = \{ D_x(y): y \text{ in } N\}$ to *w*

loop:

for each *y* in *N*:

$D_x(y) = \min \{ c(x,v) + D_v(y) \}$

v

If $D_x(y)$ changed for any destination *y* send distance vector $D_x = \{D_x(y) : y \text{ in } N\}$ to all neighbours.

forever

Program

The simulation is done on the principle of a chat server given in Expt 9. Each node is considered as a client that connects to a server. Each node reads the cost matrix and constructs the distance vector matrix which is a 3D matrix (RT), where the first element denotes the number of the node. After entering the cost matrix for each node, each node exchanges its distance vector matrix with its neighbours. This process goes on till there are no changes in the dsitance vector matrix table for each node. In the cost matrix infinite cost between two nodes is represented by 999.

client.c

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

struct message {
                int to_node;  int
                from_node;
                int RT[20][20][20];
            } ;

struct message mesg, mesg_from;
main(int argc, char * argv[])
{

   int i,j,n, h,k;
   int num_nodes;
   int sock_fd, max_fd, nready, fd[2]; char

   buffer[100], line[100];

   struct sockaddr_in servaddr; fd_set
   rset;

   int node_sel, temp1;
```

```
int node, p;

int from_node_it; int
to_node_it, t;

int prev[20];
int cost[10][10];

int N[20][20];


temp1 = 999;


h=0;
printf("Enter number of nodes:"); scanf("%d",
&num_nodes);


printf("Enter the cost matrix, For infinity enter 999\n"); for(i=0; i


<num_nodes; i++)
for(j=0; j <num_nodes; j++)
scanf("%d", &cost[i][j]);

// DISTANCE VECTOR MATRIX FOR EACH NODE

  for(i= 0; i < num_nodes; i++)
for(j=0; j < num_nodes; j++) for(k=0;
k < num_nodes; k++)
 {
   if(i==j)
     mesg.RT[i][j][k] = cost[j][k]; else
     mesg.RT[i][j][k] = 999;
 }

// NEIGHBOURS

for(i=0;  i  <  20;  i++)
for(j=0;  j  <  20;  j++)
N[i][j] = -1;

// COST MATRIX

for(i = 0; i < num_nodes; i++) for(j = 0;
 j < num_nodes; j++)
 {
    if(cost[i][j] < 999)
```

```
    {
        N[i][j] = j;
    }
}

if(argc != 3)
{
  fprintf(stderr, "Usage: ./client IPaddress_of_server port\n"); exit(1);
}

if((sock_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
 {
      printf("Cannot create socket\n"); exit(1);
 }

bzero((char*)&servaddr, sizeof(servaddr));
bzero(line, sizeof(line));

servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(atoi(argv[2]));
inet_pton(AF_INET,  argv[1],  &servaddr.sin_addr);


if(connect(sock_fd, (struct sockaddr *)&servaddr, sizeof(servaddr))< 0)
{
   perror("Connection failed:");
   exit(1);
}

fd[0] = 0;
fd[1] = sock_fd;


for(; ; )
{
   FD_ZERO(&rset); FD_SET(0,
   &rset); FD_SET(sock_fd,
   &rset); bzero(line,
   sizeof(line));

   max_fd = sock_fd;

   nready = select(max_fd + 1, &rset, NULL, NULL, NULL);
```

```
for ( j = 0; j <2 ; j++)
{
  if(FD_ISSET(fd[j],  &rset))
  {
    if(j==0)
     {  // reading from std input printf("Enter
        node number:"); scanf("%d", &node);
        printf("Node = %d\n", node);

        getchar();
        getchar();

        // send messages to its neighbours t=0;
        while(N[node][t] != -1)
        {
          if(N[node][t] != node)
           {
             mesg.from_node = node;
             mesg.to_node = N[node][t];
             n = write(fd[j+1],  &mesg, sizeof(mesg));
           }

           t++;
        }
     }
    else
     {
        read(fd[j], &mesg_from, sizeof(mesg_from));
        to_node_it = mesg_from.to_node; from_node_it =
        mesg_from.from_node;

        for(i =0; i<num_nodes; i++)
         mesg.RT[to_node_it][from_node_it][i]   =   mesg_from.RT[from_node_it][from_node_it][i];

        //DISTANCE VECTOR OF to_node_it


        mesg.RT[to_node_it][to_node_it][to_node_it] = 0; for(i =0;


        i<num_nodes  ; i++)
          prev[i] = mesg.RT[to_node_it][to_node_it][i]; i =

        from_node_it;

        while(N[to_node_it][h]  != -1)
```

```
                {
                  if(N[to_node_it][h] != 0)
                    {
                      if(N[to_node_it][h] == from_node_it)
                        {
                          node_sel = from_node_it;
                          if (temp1 > cost[to_node_it][N[to_node_it][h]] + mesg_from.RT[node_sel]
[N[to_node_it][h]][i])
                          temp1 = cost[to_node_it][N[to_node_it][h]] + mesg_from.RT[node_sel]
[N[to_node_it][h]][i];
                        }
                       else
                        {
                          node_sel = to_node_it;
                          if (temp1 > cost[to_node_it][N[to_node_it][h]] + mesg.RT[node_sel][N[to_node_it]
[h]][i])
                          temp1 = cost[to_node_it][N[to_node_it][h]] + mesg.RT[node_sel][N[to_node_it]
[h]][i];
                        }
                    } h+
                    +;
                }

                 mesg.RT[to_node_it][to_node_it][i] = temp1; h=0;

                for(i =0; i<num_nodes  ; i++)
                 {
                   if( prev[i] != mesg.RT[to_node_it][to_node_it][i])
                    {
                      p=0;
                      while(N[to_node_it][p] !=-1 )
                      {
                        if(p != to_node_it)
                         {
                           mesg.to_node =  N[to_node_it][p];
                           mesg.from_node =  to_node_it; write(fd[j],
                           &mesg, sizeof(mesg));
                         }
                         p++;
                      }
                        break;
                   }

                }
```

```
                temp1 =999;
              }

            if(--nready == 0)
            break;
        }
     }

  for(i = 0; i < num_nodes; i++)
  {
     for(j = 0; j < num_nodes; j++)
     {
        printf("%d ", mesg.RT[node][i][j]);
     }
     printf("\n");
  }
     printf("ITERATION RESULT FOR ROUTING TABLE\n");

   }

}
```

server.c

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

struct message {
                int to_node;  int
                from_node;
                int RT[20][20][20];
              } ;

struct message mesg;


main(int argc, char * argv[])
{
```

```
int n, i, maxi, max_fd, k;
int sock_fd, listen_fd, connfd, client_no;

int nready, num_q,  client[100], chat[100], conn[1000]; char

line[1000], buffer[1000];

fd_set rset, allset;
struct sockaddr_in servaddr, cliaddr; int len

= sizeof(cliaddr);

bzero(line, sizeof(line));

client_no = 0;
if(argc != 2)
{
  fprintf(stderr, "Usage: ./server  port\n"); exit(1);
}


if((listen_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
 {
       printf("Cannot create socket\n"); exit(1);
 }

bzero((char*)&servaddr,   sizeof(servaddr));

servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(atoi(argv[1]));
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);

if(bind(listen_fd, (struct sockaddr*)&servaddr, sizeof(servaddr)) < 0)
{
  perror("bind failed:");
  exit(1);
}

listen(listen_fd,  num_q);


max_fd = listen_fd;


maxi = -1;
```

```
for(i=0; i < 100; i++)
{
    client[i] = -1;
    chat[i] = -1;
}

FD_ZERO(&allset);
FD_SET(listen_fd, &allset);

for(; ;)
{
  rset = allset;
  nready = select(max_fd + 1, &rset, NULL, NULL, NULL);

  if(FD_ISSET(listen_fd,  &rset))
  {
     if((connfd = accept(listen_fd, (struct sockaddr *) &cliaddr,  &len))<0)
     {
          perror("accept failed");
          exit(1);
     }

     chat[++client_no] = connfd; conn[connfd] =
     client_no;



     k = client_no;
//     sprintf(buffer, "Client %d has joined chat\n", k);


     for( i = 0; i < 100; i++)
     {
       if( client[i] < 0)
        {
            client[i] = connfd; break;
        }
     }

     FD_SET(connfd, &allset);
     if(connfd > max_fd) max_fd =
     connfd;

     if( i > maxi)
```

```
        maxi = i;

        if(--nready <= 0)
        continue;

    }

   for( i =0; i <=maxi; i++)
   {
     bzero(line, sizeof(line)); if((sock_fd
     = client[i]) <0) continue;

        if( FD_ISSET(sock_fd, &rset))
        {
            n=   read(sock_fd, &mesg, sizeof(mesg));

          // line contains message from client

            // client closing if(
            n == 0)
            {
               close(sock_fd);
               FD_CLR(sock_fd, &allset);
               client[i] = -1;
            }
            else
            {
                    // writing to another client
                write(chat[mesg.to_node +1],  &mesg, sizeof(mesg));
            }

            if(--nready <= 0)
            break;
         }
    }

   }

}
```

Output

Figure for the network

server

anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 11$ gcc -o server server.c anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 11$ ./server 5080

client (Node 0)

anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 11$ ./client 127.0.0.1 5080 Enter number of nodes:3
Enter the cost matrix, For infinity enter 999 0 2 7
2 0 1
7 1 0

Enter node number:0 Node = 0

0 2 7
999 999 999
999 999 999
ITERATION RESULT FOR ROUTING TABLE 0 2 7
2 0 1
999 999 999
ITERATION RESULT FOR ROUTING TABLE 0 2 3
2 0 1
7 1 0
ITERATION RESULT FOR ROUTING TABLE 0 2 3
2 0 1
3 1 0
ITERATION RESULT FOR ROUTING TABLE

client (Node 1)

anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 11$ ./client 127.0.0.1 5080 Enter number of nodes:3
Enter the cost matrix, For infinity enter 999 0 2 7
2 0 1
7 1 0

Enter node number:1 Node = 1

999 999 999
2 0 1
999 999 999
ITERATION RESULT FOR ROUTING TABLE 0 2 7
2 0 1
999 999 999
ITERATION RESULT FOR ROUTING TABLE 0 2 7
2 0 1
7 1 0
ITERATION RESULT FOR ROUTING TABLE 0 2 3
2 0 1
7 1 0
ITERATION RESULT FOR ROUTING TABLE 0 2 3
2 0 1
3 1 0
ITERATION RESULT FOR ROUTING TABLE

client (Node 2)

anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 11$ ./client 127.0.0.1 5080 Enter number of nodes:3
Enter the cost matrix, For infinity enter 999 0 2 7
2 0 1
7 1 0

Enter node number:2 Node = 2

999 999 999
999 999 999
7 1 0
ITERATION RESULT FOR ROUTING TABLE 0 2 7
999 999 999
7 1 0
ITERATION RESULT FOR ROUTING TABLE 0 2 7

2 0 1
7 1 0
ITERATION RESULT FOR ROUTING TABLE 0 2 3
2 0 1
3 1 0
ITERATION RESULT FOR ROUTING TABLE

Experiment 12

Implementation and simulation of link state protocol Aim: To implement and

simulate link state protocol

Description:

Routing algorithms can be classified as global or centralized. A global routing algorithm computes the least cost path between a source and destination using knowledge about the entire network. Global algorithm has complete information about connectivity and link costs. Such algorithms are called link state algorithms.

Algorithm:

(adapted from

It is the Dijkstra algorithm. This algorithm finds the shortest (least cost paths) from the source *u* to every other node in the network.

D(v) : cost of the least cost path fromthe source node to the destination node *v* as of this iteration of the algorithm

p(v): previous node of *v* along the current least cost path from the source to *v*

N' : subset of nodes. *v* is in N' if the least cost path from the source to *v* is definitely known Iniltialization :

N' = {u}

for all nodes *v*
    if v is a neighbour of u then
        D(v) = c(u,v)
    else D(v)
= ∞

do {

    find *w* not in N' such that D(w) is a minimum add *w* to
    N'
    update D(v) for each neighbour *v* of *w* and not in N'

D(v) = min{D(v) , D(w) + c(w,v)}

} while (N' != N)

Program

```
#include<stdio.h> int
cost[10][10]; int
dist[10];
int arr[20];

void djikstra(int); int
search(int);
int length_of( int * );
void   print_route(int, int);

int prev[20];
int order_arr[20]; int

src;

main()
{
   int num_nodes,i, j, d;

   printf("Enter number of nodes:");
   scanf("%d", &num_nodes); printf("Enter
   the source node:"); scanf("%d", &src);
   printf("Enter the cost matrix, For infinity enter 999\n");

   for(i=0;    i    <num_nodes;    i++)
   for(j=0;    j    <num_nodes;    j++)
   scanf("%d", &cost[i][j]);

   djikstra( num_nodes);

}
void djikstra(int _num_nodes )
{
  int i, min_val, l; int
  k =0;
  int m =0;
  int min =999;
  int last;

  int neighbour[20];
```

```
for(i = 0; i <20; i++)
 {
   arr[i] = -1;
   neighbour[i] = -1;
   order_arr[i] = -1;
   prev[i] = -1;
 }

 arr[0] =  src;
 last =0;

 for(i = 0; i <_num_nodes; i++)
 {
   if(i != src)
   {
        if( cost[src][i]  < 999)
         {
           dist[i] = cost[src][i]; prev[i] =
           src;
         }
        else
         dist[i] = 999;
   }
   else
   {
       dist[i] = 0;
   }
 }
 do {
      for(i=0; i < _num_nodes; i++)
      {
       if(search(i) == 0)
        {
          if(dist[i] < min)
           {
             min = dist[i];
             min_val = i;
           }
        }
      }

      last++;
      arr[last] =  min_val;

      for(i=0; i<_num_nodes; i++)
```

```
        {
          if(search(i) == 0)
          {
            if(cost[min_val][i] < 999)
              {
                neighbour[m] = i; m++;
              }
          }
        }

       m =0;

       while(neighbour[m] != -1)
       {
         if(dist[min_val]  + cost[min_val][neighbour[m]] < dist[neighbour[m]])
            {
              dist[neighbour[m]] = dist[min_val] + cost[min_val][neighbour[m]]; prev[neighbour[m]] =
              min_val;
            }
           m++;
       }

       m=0;
       for(i = 0; i <_num_nodes; i++) neighbour[i] =
       -1;

       min =999;
       min_val = -1;

   }while( length_of(arr) != _num_nodes); i =1;
   l=1;
   while( i < _num_nodes)
   {
      print_route(i, l);
      printf("[ distance = %d]", dist[i]); printf("\n");
      i++;
      l++;
      for(k = 0; k <20; k++)
       order_arr[k] = -1;
   }
}
```

```
void print_route( int _i, int _l)
{
   int begin;
   int  * ptr;
   int h, len, temp;
   static int inc[20];

   if( _i == src)
   {
      ptr = order_arr; while(*ptr
      != -1) ptr++;

      len = ptr-order_arr;  for(h

      =0; h < len/2; h++)
      {
        temp = order_arr[h];
        order_arr[h] = order_arr[len - h -1];
        order_arr[len-h-1] = temp;
      }
      ptr = order_arr;

      printf("%d", src);

      while(*ptr != -1)
      {
        printf("->%d ", *ptr); ptr+
        +;
      }
      return;
   }
   else
   {
     order_arr[inc[_l]] = _i;
     inc[_l]++;
     print_route(prev[_i], _l);
   }
}

int search(_i)
{
  int i =0;

  while (arr[i] != -1)
  {
```

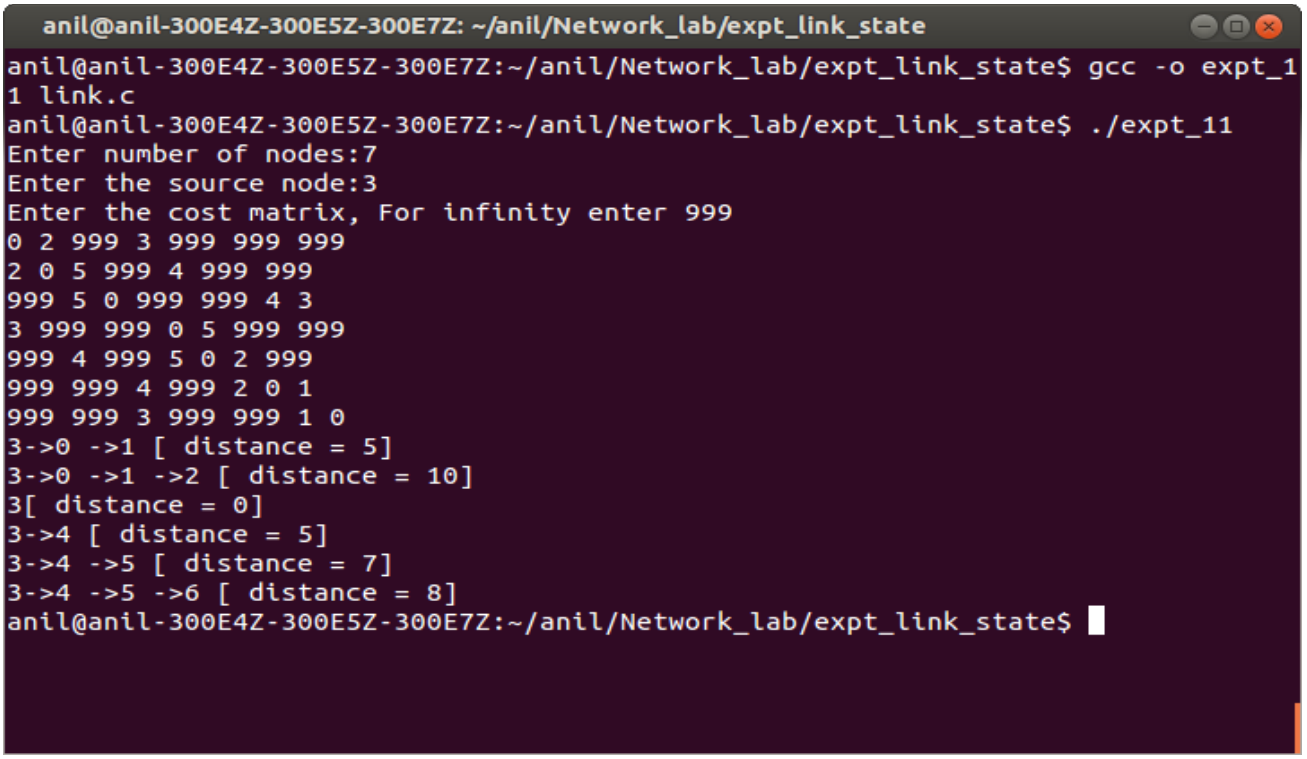
```
      if(_i == arr[i])
      break;
      else
        i++;
  }

  if(arr[i] == -1)
    return 0;
  else
    return 1;
}

int length_of( int _arr[])
{
   int i=0; while(_arr[i] !
   = -1)
       i++;
   return i;
}
```

Output

Figure shows a network where the link state algorithm is applied.

FIGURE

When finding out the cost matrix use 999 in the place of ∞.



Experiment 13

Implementation of Simple Mail Transfer Protocol

Aim: To implement a subset of simple mail transfer protocol (SMTP) using UDP

Description:

SMTP provides for mail exchanges between users on the same or different computers. The SMTP client and server can be divided into two components: user agent (UA) and mail transfer agent (MTA). The user agent is a program used to send and receive mail. The actual mail transfer is done through mail transfer agents. To send mail, a system must have client MTA, and to receive mail, a system must have a server MTA. SMTP uses commands and responses to transfer messages between an MTA client and MTA server. Commands are sent from the client to the server. It consists of a keyword followed by zero or more arguments. Examples: HELO, MAIL FROM, RCPT TO etc. Responses are sent from the server to the client. It is a three-digit code that may be followed by additional textual information. The process of transferring a mail message occurs in three phases: connection establishmnet, mail transfer, and connection termination.

Although the transport protocol specified for SMTP is TCP, in this experiment, UDP protocol will be used.

Algorithm:

SMTP Client

1. Create the client UDP socket.
2. Send the message “SMTP REQUEST FROM CLIENT” to the server. This is done so that the server understands the address of the client.
3. Read the first message from the server using client socket and print it.
4. The first command HELO<”Client's mail server address”> is sent by the client
5. Read the second message from the server and print it.
6. The second command MAIL FROM:<”email address of the sender”> is sent by the client.
7. Read the third message from the server and print it.
8. The third command RCPT TO:<”email address of the receiver”> is sent by the client
9. Read the fourth message from the server and print it.

10. The fourth command DATA is sent by the client.

11. Read the fifth message from the server and print it.

12. Write the messages to the server and end with "."

13. Read the sixth message from the server and print it.

14. The fifth command QUIT is sent by the client.

15. Read the seventh message from the server and print it.

Server

1. Create the server UDP socket

2. Read the message from the client and gets the client's address

3. Send the first command to the client.
   220 "server name"

4. Read the first message from the client and print it.

5. Send the second command to the client.
   250 Hello "client name"

6. Read the second message from client and print it.

7. Send the third command to the client.
   250 "client email address ".....Sender ok

8. Read the third message from client and print it

9. Send the fourth command to the client
   250 "server email address".....Recepient ok

10. Read the fourth message from client and print it

11. Send the fifth command to the client
    354 Enter mail, end with "." on a line by itself

12. Read the email text from the client till a "." is reached

13. Send the sixth command to the client
    250 Message accepted for delivery

14. Read the fifth message from the client and print it.

15. Send the seventh command to the client
221 "server name" closing connection

Program

Client

```
#include<stdio.h>

#include<string.h>

#include<sys/socket.h>

#include<sys/types.h>

#include<netinet/in.h>

#include<arpa/inet.h>

#include<fcntl.h>

#include<stdlib.h>


#define MAXLINE 100 main(int

argc,char * argv[])

{

        int n;

        int sock_fd;

        int i=0;


        struct sockaddr_in servaddr; char

        buf[MAXLINE+1];
```

```
char address_buf[MAXLINE],message_buf[MAXLINE]; char *
str_ptr, *buf_ptr, *str;
if(argc!=3)
{
     fprintf(stderr,"Command is :./client address port\n"); exit(1);
}
if((sock_fd = socket(AF_INET, SOCK_DGRAM, 0))<0)
{
        printf("Cannot create socket\n");
        exit(1);
}
bzero((char *) & servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_port=htons(atoi(argv[2]));
inet_pton(AF_INET,argv[1],&servaddr.sin_addr);
sprintf(buf,"SMTP REQUEST FROM CLIENT\n");

n=sendto(sock_fd,buf,strlen(buf),0,(struct sockaddr*)&servaddr,sizeof(servaddr)); if(n<0)
{
    perror("ERROR"); exit(1);
```

```
}
if((n=recvfrom(sock_fd,buf,MAXLINE,0,NULL,NULL))==-1)
{
        perror("UDP read error"); exit(1);
}
buf[n]='\0';
printf("S:%s",buf);
sprintf(buf,"HELLO   name_of_client_mail_server\n");

n=sendto(sock_fd,buf,strlen(buf),0,(struct    sockaddr*)&servaddr,sizeof(servaddr));

if((n=recvfrom(sock_fd,buf,MAXLINE,0,NULL,NULL))==-1)
{
        perror("UDP read error"); exit(1);
}
buf[n]='\0';
printf("S:%s",buf);
printf("please enter the email address of the sender:");
fgets(address_buf,sizeof(address_buf),stdin);
address_buf[strlen(address_buf)-1]='\0';

sprintf(buf,"MAIL FROM :<%s>\n",address_buf);
```

```
sendto(sock_fd,buf,sizeof(buf),0,(struct sockaddr*)&servaddr,sizeof(servaddr));
if((n=recvfrom(sock_fd,buf,MAXLINE,0,NULL,NULL))==-1)
{
        perror("UDP read error"); exit(1);
}
buf[n]='\0';
printf("S:%s",buf);
printf("please enter the email address of the receiver:");
fgets(address_buf,sizeof(address_buf),stdin);
address_buf[strlen(address_buf)-1]='\0'; sprintf(buf,"RCPT TO : <
%s>\n",address_buf);
sendto(sock_fd,buf,strlen(buf),0, (struct sockaddr *) &servaddr,sizeof(servaddr));
if((n=recvfrom(sock_fd,buf,MAXLINE,0,NULL,NULL))==-1)
{
        perror("UDP read error"); exit(1);
}
buf[n]='\0';
printf("S:%s",buf); sprintf(buf,"DATA\n");
sendto(sock_fd,buf,strlen(buf),0,(struct sockaddr*)&servaddr,sizeof(servaddr));
if((n=recvfrom(sock_fd,buf,MAXLINE,0,NULL,NULL))==-1)
```

```
    {
        perror("UDP read error"); exit(1);
    }
    buf[n]='\0';
    printf("S:%s",buf);
do
 {
        fgets(message_buf,sizeof(message_buf),stdin); sprintf(buf,"%s",message_buf);
        sendto(sock_fd,buf,strlen(buf),0,(struct sockaddr*)&servaddr,sizeof(servaddr));
        message_buf[strlen(message_buf)-1]='\0';
        str=message_buf;
        while(isspace(*str++));
        if(strcmp(--str,".")==0)
        break;
 } while(1);

if((n=recvfrom(sock_fd,buf,MAXLINE,0,NULL,NULL))==-1)
 {
       perror("UDP read error"); exit(1);
  }
```

```
        buf[n]='\0'; sprintf(buf,"QUIT\n");
        printf("S:%s",buf);
        sendto(sock_fd,buf,strlen(buf),0,(struct sockaddr*)&servaddr,sizeof(servaddr));
        if((n=recvfrom(sock_fd,buf,MAXLINE,0,NULL,NULL))==-1)
         {
                perror("UDP read error"); exit(1);
          }

           buf[n]='\0';
           printf("S:%s",buf);
}
```

Server

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

```
#define MAXLINE 100 main(int

argc, char * argv[])

{
        int n,sock_fd;

        struct sockaddr_in servaddr,cliaddr; char

        mesg[MAXLINE+1]; socklen_t len;

        char * str_ptr, *buf_ptr, *str;

        len=sizeof(cliaddr);

        if((sock_fd=socket(AF_INET,SOCK_DGRAM,0))<0)

        {

            printf("cannot create socket\n");

           exit(1);

       }

        bzero((char*)&servaddr,sizeof(servaddr)); servaddr.sin_family=AF_INET;

        servaddr.sin_port=htons(atoi(argv[1]));

        servaddr.sin_addr.s_addr=htonl(INADDR_ANY); if(bind(sock_fd,(struct

        sockaddr*)&servaddr,sizeof(servaddr))<0)

        {

           perror("bind failed:");

           exit(1);

        }
```

```
if((n=recvfrom(sock_fd,mesg,MAXLINE,0,(struct      sockaddr*)&cliaddr,&len))==-1)
{
     perror("size not received:"); exit(1);
}
mesg[n]='\0'; printf("mesg:%s\n",mesg);
sprintf(mesg,"220 name_of_server_mail_server\n"); sendto(sock_fd,mesg,MAXLINE,0,(struct
sockaddr*)&cliaddr,sizeof(cliaddr)); n=recvfrom(sock_fd,mesg,MAXLINE,0,(struct
sockaddr*)&cliaddr,&len); mesg[n]='\0';
printf("C:%s\n",mesg); str_ptr=strdup(mesg);
buf_ptr=strsep(&str_ptr," ");
sprintf(mesg,"250 Hello %s",str_ptr);
free(buf_ptr);
sendto(sock_fd, mesg, MAXLINE, 0, (struct sockaddr *)&cliaddr, sizeof(cliaddr));
n=recvfrom(sock_fd,mesg,MAXLINE,0,(struct sockaddr*)&cliaddr,&len); mesg[n]='\0';
printf("C:%s",mesg);
str_ptr=strdup(mesg);
buf_ptr=strsep(&str_ptr,":");
str_ptr[strlen(str_ptr)-1]='\0';
```

```
sprintf(mesg,"250 Hello %s.....sender ok\n",str_ptr);
free(buf_ptr);
sendto(sock_fd,mesg,MAXLINE,0,(struct sockaddr*)&cliaddr,sizeof(cliaddr));
n=recvfrom(sock_fd,mesg,MAXLINE,0,(struct sockaddr*)&cliaddr,&len); mesg[n]='\0';
printf("C:%s",mesg);
str_ptr=strdup(mesg);
buf_ptr=strsep(&str_ptr,":");
str_ptr[strlen(str_ptr)-1]='\0';
sprintf(mesg,"250 Hello %s.....Recepient ok\n",str_ptr);
free(buf_ptr);
sendto(sock_fd,mesg,MAXLINE,0,(struct sockaddr*)&cliaddr,sizeof(cliaddr));
n=recvfrom(sock_fd,mesg,MAXLINE,0,(struct sockaddr*)&cliaddr,&len); mesg[n]='\0';
printf("C:%s\n",mesg);
sprintf(mesg,"354 Enter mail,end with \".\" on a line by itself \n");
sendto(sock_fd,mesg,MAXLINE,0,(struct sockaddr*)&cliaddr,sizeof(cliaddr)); while(1)
{
    n=recvfrom(sock_fd,mesg,MAXLINE,0,(struct sockaddr*)&cliaddr,&len); mesg[n]='\0';
    printf("C:%s\n",mesg);
   mesg[strlen(mesg)-1]='\0';
```

```
            str=mesg;
            while(isspace(*str++));
            if(strcmp(--str,".")==0)
            break;
           sprintf(mesg,"250 messages accepted for delivery \n"); sendto(sock_fd,mesg,MAXLINE,0,
           (struct sockaddr*)&cliaddr,sizeof(cliaddr)); n=recvfrom(sock_fd,mesg,MAXLINE,0,(struct
           sockaddr*)&cliaddr,&len); mesg[n]='\0';
            printf("C:%s\n",mesg);
           sprintf(mesg,"221 servers mail server closing connection\n");
           sendto(sock_fd,mesg,MAXLINE,0,(struct sockaddr*)&cliaddr,sizeof(cliaddr));
        }

}
```

Server program

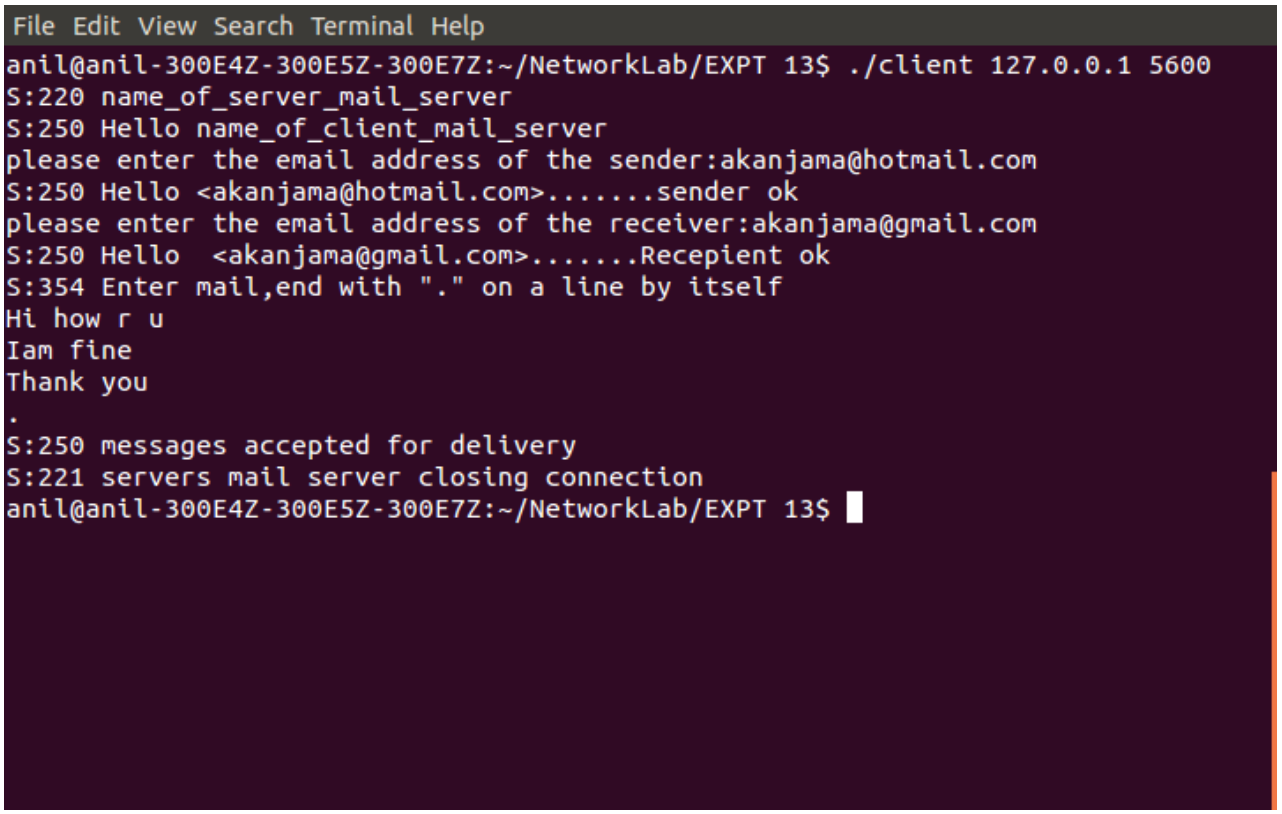
```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<fcntl.h>
#include<stdlib.h>

#define MAXLINE 100 main(int

argc, char * argv[])
{
   int n, sock_fd;
   struct sockaddr_in servaddr, cliaddr;
```

```
char mesg[MAXLINE + 1]; socklen_t
len;

char * str_ptr, *buf_ptr, *str; len =
sizeof(cliaddr);


if((sock_fd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
 {
      printf("Cannot create socket\n"); exit(1);
 }

bzero((char*)&servaddr,  sizeof(servaddr));

servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(atoi(argv[1]));
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);


if(bind(sock_fd, (struct sockaddr*)&servaddr, sizeof(servaddr)) < 0)
{
  perror("bind failed:");
  exit(1);
}

if((n = recvfrom(sock_fd, mesg, MAXLINE, 0, (struct sockaddr *)&cliaddr, &len)) == -1)
{
   perror("size  not received:"); exit(1);
}

mesg[n] = '\0';

printf("mesg:%s\n",  mesg);

sprintf(mesg, "220 name_of_server_mail_server\n");
sendto(sock_fd, mesg, MAXLINE,0, (struct sockaddr*)&cliaddr, sizeof(cliaddr));

n = recvfrom(sock_fd, mesg, MAXLINE, 0, (struct sockaddr *)&cliaddr, &len); mesg[n] = '\0';
printf("C:%s\n",mesg);

str_ptr = strdup(mesg);
buf_ptr = strsep(&str_ptr, " "); sprintf(mesg, "250
Hello %s", str_ptr);
```

```
free(buf_ptr);
sendto(sock_fd, mesg, MAXLINE,0, (struct sockaddr*)&cliaddr, sizeof(cliaddr));

n = recvfrom(sock_fd, mesg, MAXLINE, 0, (struct sockaddr *)&cliaddr, &len); mesg[n] = '\0';
printf("C:%s\n",mesg);

str_ptr = strdup(mesg);  buf_ptr =
strsep(&str_ptr, ":");
str_ptr[strlen(str_ptr)-1] = '\0';
sprintf(mesg, "250 Hello %s.....Sender ok\n", str_ptr);
free(buf_ptr);
sendto(sock_fd, mesg, MAXLINE,0, (struct sockaddr*)&cliaddr, sizeof(cliaddr));

n = recvfrom(sock_fd, mesg, MAXLINE, 0, (struct sockaddr *)&cliaddr, &len); mesg[n] = '\0';
printf("C:%s\n",mesg)

sprintf(mesg, "354 Enter mail, end with \".\" on a line by itself\n");
sendto(sock_fd, mesg, MAXLINE,0, (struct sockaddr*)&cliaddr, sizeof(cliaddr));

while(1)
 {
    n = recvfrom(sock_fd, mesg, MAXLINE, 0, (struct sockaddr *)&cliaddr, &len); mesg[n] = '\0';
    printf("C:%s\n",mesg);

    mesg[strlen(mesg) – 1] = '\0'; str =
    mesg; while(isspace(*str++));
    if(strcmp(--str, ".") == 0) break;

    sprintf(mesg, "250 Message accepted for delivery\n");
    sendto(sock_fd, mesg, MAXLINE,0, (struct sockaddr*)&cliaddr, sizeof(cliaddr));

    n = recvfrom(sock_fd, mesg, MAXLINE, 0, (struct sockaddr *)&cliaddr, &len); mesg[n] = '\0';
    printf("C:%s\n",mesg);

    sprintf(mesg, "221 Server's mail server closing connection\n");
    sendto(sock_fd, mesg, MAXLINE,0, (struct sockaddr*)&cliaddr, sizeof(cliaddr));

}
```

Output

Client



Server

```
File  Edit  View  Search  Terminal  Help
anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 13$ ./server 5600
mesg:SMTP REQUEST FROM CLIENT

C:HELLO name_of_client_mail_server

C:MAIL FROM :<akanjama@hotmail.com>
C:RCPT TO : <akanjama@gmail.com>
C:DATA

C:Hi how r u

C:Iam fine

C:Thank you

C:.

C:QUIT

^C
anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 13$
```

Experiment 14

Implementation of simple file server

Aim: To develop a concurrent file server which will provide the file requested by client if it exists. If not, server sends appropriate message to the client. Server sends its process ID (PID) to clients for display along with file or the message.

Description:

The file server creates listening sockets on two ports that have consecutive port numbers. One is the listening socket for control connection and the other is the listening socket for data connection. These sockets have descriptors *listen_control* and *listen_data* respectively. The client creates a socket and connects to the server using TCP connection. The client socket descriptor is stored in *sock_ctrl.* The server creates a connection socket when the client makes a TCP connection with the server. This connection socket is stored in *sock_ctrl.* This connection is for control information to pass between client and server. Next, the server forks a child process. Since the child is a perfect image of its parent, the child process will also have descriptors *listen_control*, *listen_data* and *sock_ctrl*. The control connection will also be duplicated between the client and the child process. The child process closes its listening socket for control connection, i.e; *listen_control*. The parent server process closes its connection socket for the control connection i.e; *sock_ctrl*. The client now makes a TCP connection with the server for transferring data is stored in *sock_data.* The connection socket for the data connection is stored in *sock_data* in the server child process.

Algorithm

Client

1. Create the client TCP control connection to a port (port_num) of the server with the client socket descriptor *sock_ctrl.*

2. `while(1)

{

2.1 Create client socket descriptor *sock_data* for the data connection with server on another port (portnum + 1). For each file transfer a new data connection is required.

2.2 Read the command from the terminal given by the user

2.3 Send the command to the server using control socket *sock_ctrl.*

2.4 If command == “close”

close *sock_data* and *sock_ctrl*
break

2.5 Enter the filename using the keyboard

2.6 Write the filename using control socket *sock_ctrl* to the server.

2.7 Use *while(connect(...) < 0)* to wait for the data connection.

2.8 read data using *sock_data* (file contents) from the server and write to the file

2.9 If file does not exist print the process id of server

}

Server

1. Initialize variable *file_present* to 1

2. Create listening socket for the control connection on port (port_num) and store it in *listen_control.*

3. Create a listening socket for the data connection on port (port_num +1) and store it in *listen_data*

4. while (1)

{

4.1 accepts the client control connection and returns the connect socket descriptor *sock_ctrl*

4.2 fork a child process

4.2.1 if(childprocess)

{

close listening socket *listen_control*

while(1)

{

read command from the client on the control connection

if command == "close"

break

else

read the filename from the client using control connection open the file

if (no file)

form a string "@FILE NOT FOUND PROCESS ID = getprocess id"

and store it in variable *buffer*

file_present = 0;

accept the data connection and return the socket descriptor *sock_data*

if(file_present)

read file contents and write to *sock_data*

form a string "FILE *filename* RECEIVED FROM SERVER WITH PROCESS ID =

getprocess id" and store it in variable *buffer*

else

file_present = 1;

write *buffer* using *sock_data* to the client

}

close *sock_data*

close*(sock_ctrl)*

close*(listen_data)* child

process exits

```
        }
      close( sock_ctrl)

    }
```

5. close(*listen_control*);

6. close(*listen_data*);

Program

Client

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<fcntl.h>
#include<stdlib.h>
main(int argc, char * argv[])
{
    int n, fd, i;
    int sock_ctrl, sock_data;
```

```
    char buffer[100], line[100], cmd[100]; char
    name[100];
    char * p;
  struct sockaddr_in servaddr; if(argc !=
   3)
   {
        fprintf(stderr, "Usage: ./client IPaddress_of_server port\n"); exit(1);
 }
 if((sock_ctrl = socket(AF_INET, SOCK_STREAM, 0)) < 0)
  {
        printf("Cannot create control socket\n"); exit(1);
  }
 bzero((char*)&servaddr, sizeof(servaddr));
 bzero(line, sizeof(line));
 bzero(buffer, sizeof(buffer));
servaddr.sin_family = AF_INET;
 servaddr.sin_port = htons(atoi(argv[2]));
 inet_pton(AF_INET,  argv[1],  &servaddr.sin_addr);
```

```
if(connect(sock_ctrl, (struct sockaddr *)&servaddr, sizeof(servaddr))< 0)
{
    perror("Connection failed, control socket:"); exit(1);
}
printf(" Enter \"get\" for receiving file from server\n"); printf("
Enter \"close\" for closing connection\n");  while (1)
{
  i=0;
  if(( sock_data = socket(AF_INET, SOCK_STREAM, 0)) < 0)
   {
      perror("Creation of client data socket failed");
      exit(1);
   }
   printf("Enter command:");
   scanf("%s", cmd);
   n =  write( sock_ctrl, cmd, sizeof(cmd));
  if(strcmp(cmd, "close") == 0)
  {
     close(sock_data);
     close(sock_ctrl);
     break;
```

```
    }
     printf("Enter filename:");
     scanf("%s", name);
    printf("name of file = %s\n", name);
    if((fd = open(name, O_WRONLY | O_CREAT)) < 0)
    {
        perror("Error in opening file "); exit(1);
    }
    write(sock_ctrl, name, sizeof(name)); servaddr.sin_port=
htons(atoi(argv[2]) + 1);
  n =  connect(sock_data, (struct sockaddr *)&servaddr, sizeof(servaddr)); while(n == -1)
 {
   n = connect(sock_data, (struct sockaddr *)&servaddr, sizeof(servaddr)); perror("cannot connect");
 }
 do
 {
        n = read(sock_data, buffer, sizeof(buffer)); write(fd, buffer,
        n);
         i += n;
    } while(n>0);
```

```
   if(buffer[0] =='@')
   {
      p = buffer; printf("%s\
      n", p++);
      remove(name);
   }

 }

}
```

Server

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

main(int argc, char * argv[])
{
   int n,m, fd, i ;
```

```
int sock_ctrl, sock_data, listen_control, listen_data; int
file_present =1;
char name[100], buffer[100], cmd[100]; struct
sockaddr_in servaddr, cliaddr;
int cli_len = sizeof(cliaddr); if(argc !=
2)
{
  fprintf(stderr, "Usage: ./server  port\n"); exit(1);
}
 if((listen_control = socket(AF_INET, SOCK_STREAM, 0)) < 0)
  {
      perror("cannot create listening socket for control connection"); exit(1);
  }
  bzero(&servaddr, sizeof(servaddr)); servaddr.sin_family =
  AF_INET; servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
  servaddr.sin_port = htons(atoi(argv[1]));
  if(bind(listen_control, (struct sockaddr*)&servaddr, sizeof(servaddr)) < 0)
   {
          perror("server bind failed for control listening socket"); exit(1);
```

```
    }
    listen(listen_control,  5);
  if((listen_data = socket(AF_INET, SOCK_STREAM, 0)) < 0)
   {
       perror("cannot create listening socket for data connection"); exit(1);
   }
   servaddr.sin_port = htons(atoi(argv[1]) +1);
   if(bind(listen_data, (struct sockaddr*)&servaddr, sizeof(servaddr)) < 0)
    {
          perror("server bind failed for data listening socket"); exit(1);
    }
    listen(listen_control,  5);
    listen(listen_data, 5);
 for(; ;)
 {
     if((sock_ctrl = accept(listen_control, (struct sockaddr*)&servaddr, & cli_len)) <0)
      {
          perror("accept failed");
          exit(1);
      }
     servaddr.sin_port = htons(atoi(argv[1] + 1));
```

```
if(fork() == 0)
  {
    close(listen_control); while(1)
       {
          i=0;
          n = read(sock_ctrl, cmd, 100);

          if(strcmp(cmd, "close") == 0)
          {
             break;
          }
          else
          {
             read(sock_ctrl, name, 100);
             if((fd = open(name, O_RDONLY)) <0)
             {
                  sprintf(buffer, "@FILE NOT FOUND PROCESS ID =  %d", getpid()); perror("error in
                  opening file");
                  file_present = 0;
             }

            if((sock_data = accept(listen_data, (struct sockaddr*)&servaddr, & cli_len)) <0)
```

```
        {
            perror("accept failed");
            exit(1);
        }
        if(file_present == 1)
        {
          do
           {
              n = read(fd, buffer, sizeof(buffer)); write(sock_data,
              buffer, n);
              i = i+n;
           } while(n>0);
           close(fd);
           sprintf(buffer, "FILE %s RECEIVED FROM SERVER WITH PROCESS ID = %d",
name, getpid());
              m = write(sock_data, buffer, strlen(buffer));
        }
         else
        {
              m = write(sock_data, buffer, sizeof(buffer));
              file_present =1;
              bzero(buffer, sizeof(buffer));
        }
```

```
                close(sock_data);
              }
            }
          close(sock_ctrl);
          close(listen_data); exit(0);
    }
    close(sock_ctrl);
}
    close(listen_control); close(listen_data);
      }
```

Output

The files stored in the server are *a1, b1,* and *c1.* Client 1 connects to the server and gets files a1 and b1. Client 2 connects to server and receives *c1.* The server process id is stored in the files received.

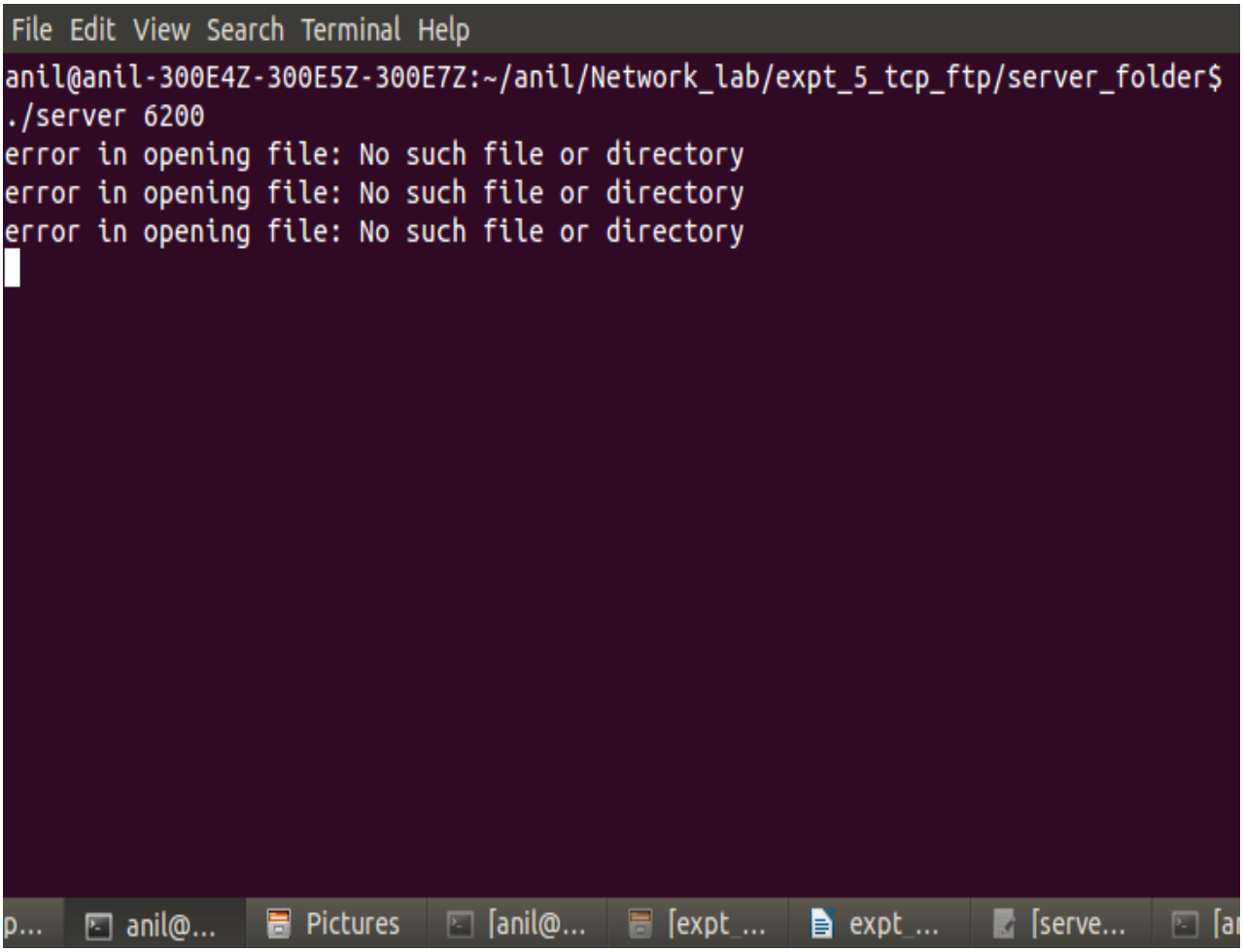
Client 1

```
File  Edit  View  Search  Terminal  Help
$ ./client 127.0.0.1 6200
 Enter "get" for receiving file from server
 Enter "close" for closing connection
Enter command:get
Enter filename:a1
name of file = a1
Enter command:get
Enter filename:b1
name of file = b1
Enter command:get
Enter filename:e1
name of file = e1
@FILE NOT FOUND PROCESS ID =  10902
Enter command:close
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt_5_tcp_ftp/client_1_folder
$ cat a1
Iam file a1
Iam in server
FILE a1 RECEIVED FROM SERVER WITH PROCESS ID = 10902anil@anil-300E4Z-300E5Z-300E
$ cat b1
Iam file b2
Iam in server
FILE b1 RECEIVED FROM SERVER WITH PROCESS ID = 10902anil@anil-300E4Z-300E5Z-300E
$ :~/anil/Network_lab/expt_5_tcp_ftp/client_1_folder$
```

Client 2

```
File  Edit  View  Search  Terminal  Help
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt_5_tcp_ftp/client_2_folder
$ ./client 127.0.0.1 6200
 Enter "get" for receiving file from server
 Enter "close" for closing connection
Enter command:get
Enter filename:c1
name of file = c1
Enter command:get
Enter filename:d1
name of file = d1
@FILE NOT FOUND PROCESS ID =  10899
Enter command:close
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt_5_tcp_ftp/client_2_folder
$ cat c1
Iam file b3
Iam in server
FILE c1 RECEIVED FROM SERVER WITH PROCESS ID = 10899anil@anil-300E4Z-300E5Z-300E
$ :~/anil/Network_lab/expt_5_tcp_ftp/client_2_folder$
```

Server


File Edit View Search Terminal Help
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt_5_tcp_ftp/server_folder$
./server 6200
error in opening file: No such file or directory
error in opening file: No such file or directory
error in opening file: No such file or directory
p...
anil@...
Pictures
[anil@...
[expt ...
expt ...
[serve...

Experiment 15

UDP datagram in client server communication using Wireshark

Aim: To observe data transferred using UDP in client server communication and to study UDP datagram.

Description:

Wireshark is a free network protocol analyzer that runs on Windows, Mac, and Linux/Unix computer. It operates in computers using Ethernet, serial (PPP and SLIP), 802.11 wireless LANs, and many other link-layer technologies. It is also a packet sniffer since it can be used to observe messages exchanged between protocol entities. It will also typically store and/or display the contents of the various protocol fields in these captured messages. A packet sniffer receives a copy of packets that are sent/received from/by application and protocols executing on your machine. Packet sniffer software consists of two components. One is packet capture library that receives a copy of every link-layer frame that is sent from or received by your computer. The second component of a packet sniffer is the packet analyzer, which displays the contents of all fields within a protocol message. Wireshark has a rich functionality that includes the capability to analyze hundreds of protocols, and a well-designed user interface.

Wireshark is installed in the computer. The wireshark interface has 5 major components.

1. Command menus
The two noteworthy menus are File menu that allows you to save captured packet data and exit the Wireshark application. The Capture menu allows you to begin packet capture and stop capturing when needed.

2. Packet listing window
Displays a one-line summary for each packet captured, including the packet number assigned by Wireshark, the time at which the packet was captured, the packet's source and destination addresses, the protocol type, and protocol-specific information contained in the packet.

3. Packet-header details window
Provides details about the packet selected in the packet-listing window.

4. Packet-contents window
Displays the entire contents of the captured frame in both ASCII and hexadecimal format.

Output

UDP datagrams

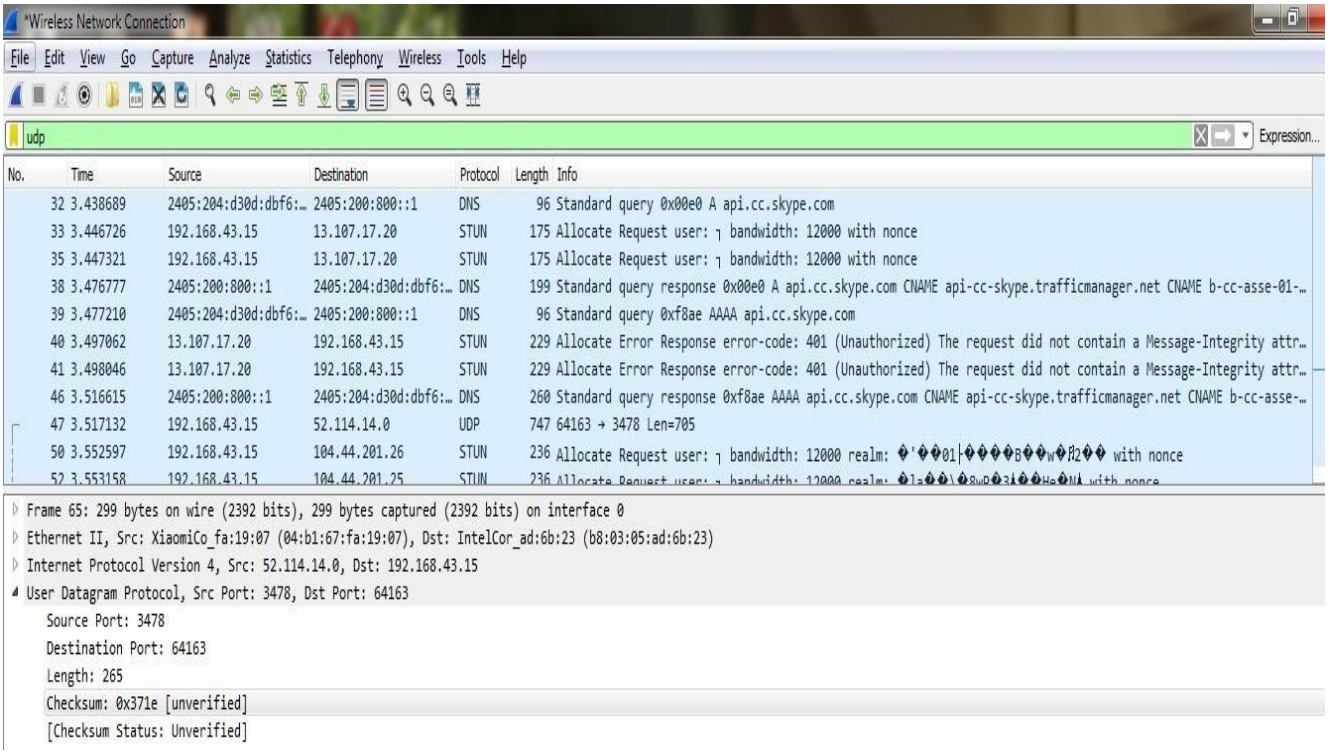
Packet capturing is started. Skype application is also started. This generates UDP datagrams.

From the trace, the details of frame 47 can be seen. It is an UDP datagram as specified by column 5.
UDP header has the following fields

source port = 64163
destination port = 3478
Length field = Length of UDP header + Length of data field = 713 bytes checksum =0x9beb

UDP header has a length of 8bytes. The payload of the UDP datagram is therefore 705 bytes whose hexadecimal values are given in the trace.

Output



Experiment 16

3 way handshake during TCP connection establishment using Wireshark

Aim: To observe 3 way handshake during TCP connection establishment using Wireshark

Description:

In this experiment, we upload a file to a server.

The steps are

1. Create a file named input.txt
2. Start Wireshark and begin packet capturing
3. Upload input.txt to a server with IP address 128.119.245.12 using a web page.
4. Stop packet capturing when the file is completely uploaded.

Output:

IP address of the client computer is 192.168.43.15 and the source port is 53001. The destination has IP address 128.119.245.12 with port number 80.

Connection Establishment

The TCP SYN segment is used to initiate the TCP connection between the client computer and server. Frame 24 in the trace shows the transfer of SYN segment from the source to destination computer. It has sequence number 0. The SYN flag is set to 1 and it indicates that this segment is a SYN segment.

The sequence number of the SYN/ACK segment from server to the client computer in reply to the SYN has the value of 0 in the trace and is given by frame 27. The value of the acknowledgement field in the SYN/ACK segment is 1. The value of the acknowledgement field in the SYN/ACK segment is determined by the server by adding 1 to the initial sequence number of SYN segment

from the client computer. The SYN flag and Acknowledgement flag in the segment are set to 1 and they indicate that this segment is a SYNACK segment. Finally an acknowledgement is sent by the client computer to the server with ack number 1 and sequence number 1 as shown in frame 28. This completes TCP connection establishment.

Data Transfer

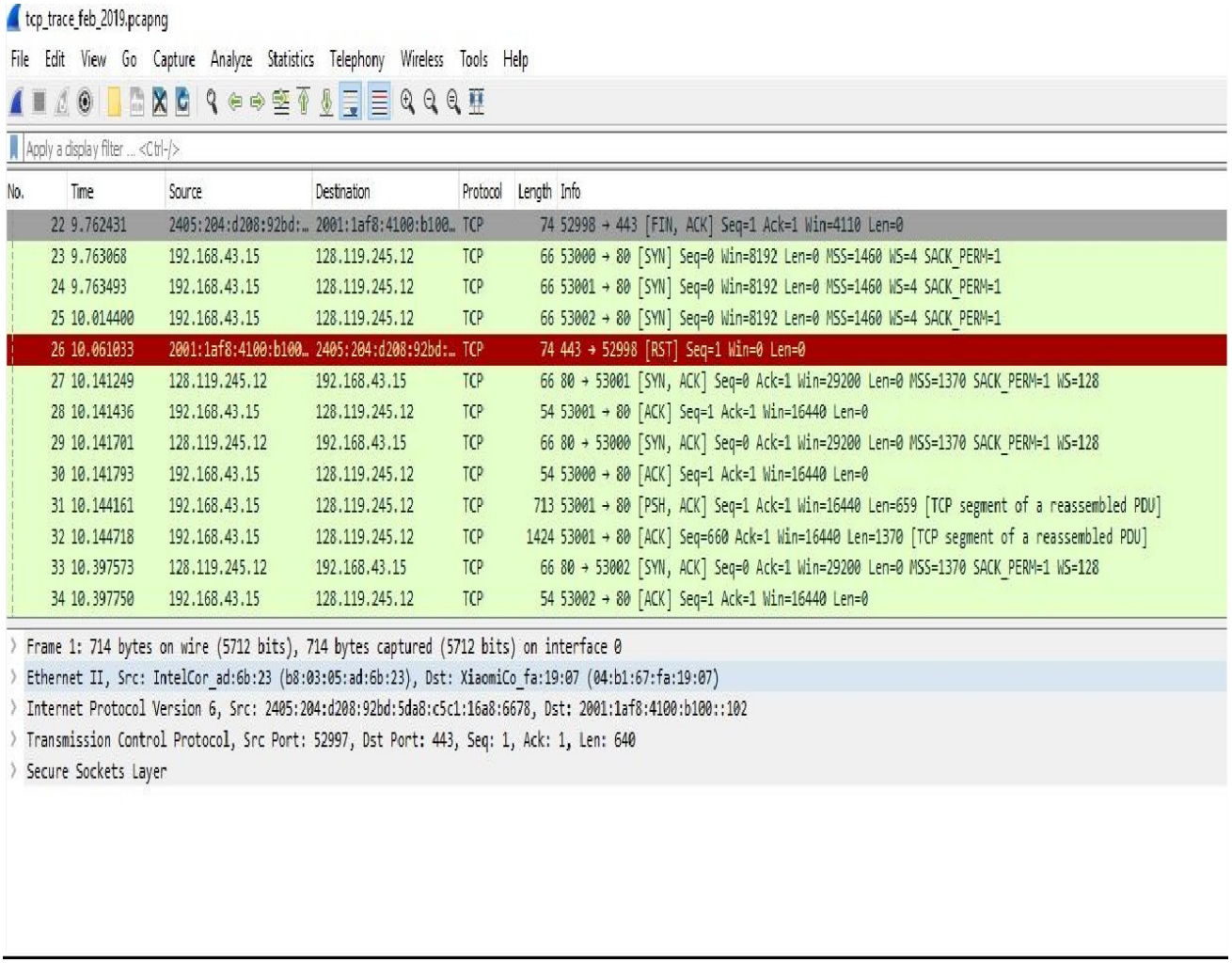
Let us look at packet numbers 42 and 43. They are TCP segments carrying data from the source to destination. The sequence number of former segment = 7510, while that of the latter = 8880. The

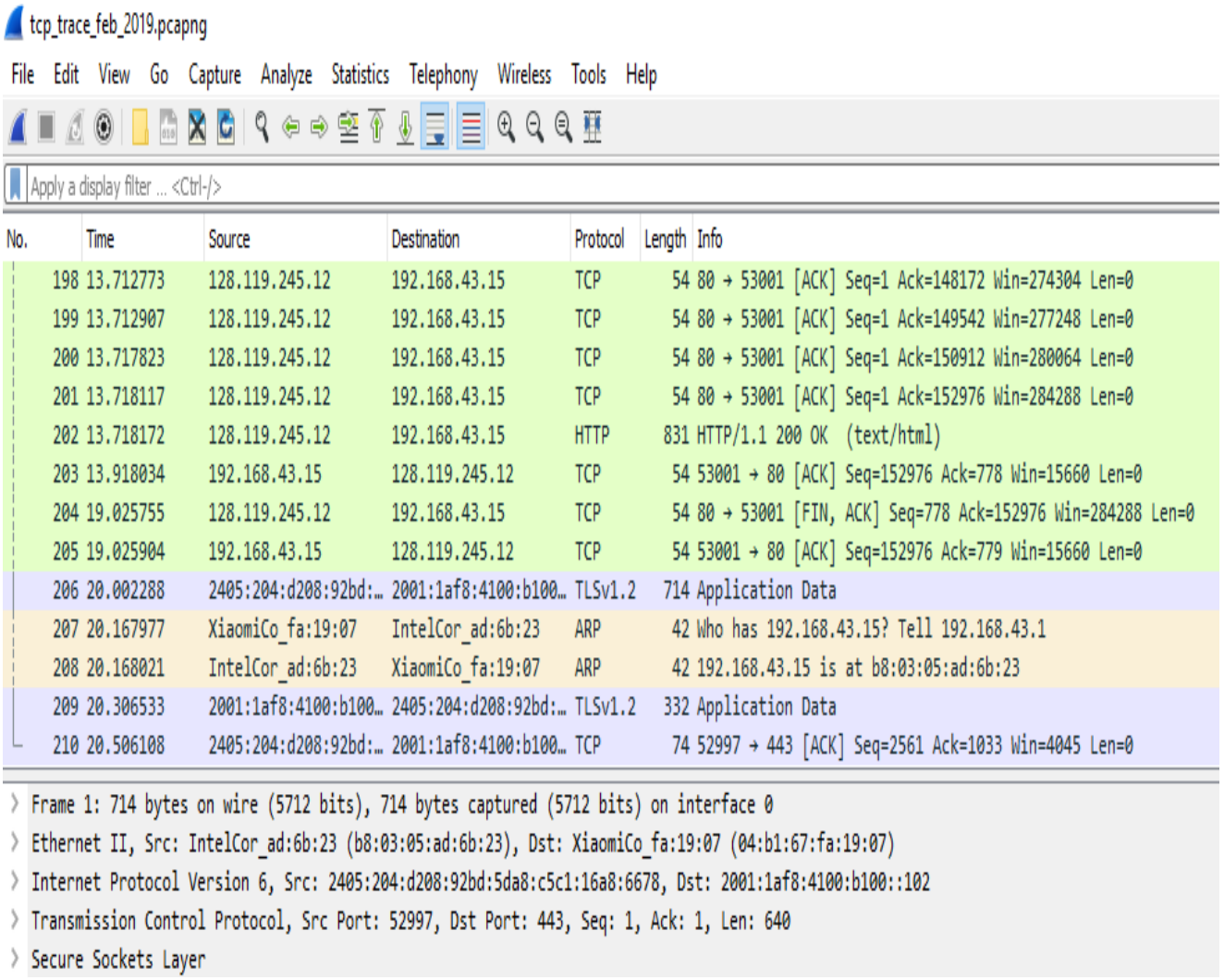
difference between the two sequence numbers = 8880 – 7510 = 1370 bytes is the length of the TCP segment.

Connection Termination

Packet 204 is the TCP segment with FIN and ACK fields set from the destination to source. The source then sends an ACK given by packet 2015

Output




tcp_trace_feb_2019.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
198 13.712773 128.119.245.12 192.168.43.15 TCP 54 80 → 53001 [ACK] Seq=1 Ack=148172 Win=274304 Len=0
199 13.712907 128.119.245.12 192.168.43.15 TCP 54 80 → 53001 [ACK] Seq=1 Ack=149542 Win=277248 Len=0
200 13.717823 128.119.245.12 192.168.43.15 TCP 54 80 → 53001 [ACK] Seq=1 Ack=150912 Win=280064 Len=0
201 13.718117 128.119.245.12 192.168.43.15 TCP 54 80 → 53001 [ACK] Seq=1 Ack=152976 Win=284288 Len=0
202 13.718172 128.119.245.12 192.168.43.15 HTTP 831 HTTP/1.1 200 OK (text/html)
203 13.918034 192.168.43.15 128.119.245.12 TCP 54 53001 → 80 [ACK] Seq=152976 Ack=778 Win=15660 Len=0
204 19.025755 128.119.245.12 192.168.43.15 TCP 54 80 → 53001 [FIN, ACK] Seq=778 Ack=152976 Win=284288 Len=0
205 19.025904 192.168.43.15 128.119.245.12 TCP 54 53001 → 80 [ACK] Seq=152976 Ack=779 Win=15660 Len=0
206 20.002288 2405:204:d208:92bd:… 2001:1af8:4100:b100… TLSv1.2 714 Application Data
207 20.167977 XiaomiCo_fa:19:07 IntelCor_ad:6b:23 ARP 42 Who has 192.168.43.15? Tell 192.168.43.1
208 20.168021 IntelCor_ad:6b:23 XiaomiCo_fa:19:07 ARP 42 192.168.43.15 is at b8:03:05:ad:6b:23
209 20.306533 2001:1af8:4100:b100… 2405:204:d208:92bd:… TLSv1.2 332 Application Data
210 20.506108 2405:204:d208:92bd:… 2001:1af8:4100:b100… TCP 74 52997 → 443 [ACK] Seq=2561 Ack=1033 Win=4045 Len=0
Frame 1: 714 bytes on wire (5712 bits), 714 bytes captured (5712 bits) on interface 0
Ethernet II, Src: IntelCor_ad:6b:23 (b8:03:05:ad:6b:23), Dst: XiaomiCo_fa:19:07 (04:b1:67:fa:19:07)
Internet Protocol Version 6, Src: 2405:204:d208:92bd:5da8:c5c1:16a8:6678, Dst: 2001:1af8:4100:b100::102
Transmission Control Protocol, Src Port: 52997, Dst Port: 443, Seq: 1, Ack: 1, Len: 640
Secure Sockets Layer

Experiment 17

Packet capturing and filtering application using raw sockets.

Aim: To develop a packet capturing and filtering application using raw sockets

Description:

Normally, an application accesses the network layer using the transport layer. However, if a raw socket is used an application can bypass the transport layer and access the network layer directly.

The raw socket is created using the socket function

int sockfd;
sockfd = socket(AF_INET, SOCK_RAW, protocol);

where protocol is one of the the constants IPPROTO_xxx defined in <netinet/in.h> header. The socket created above is an IPv4 raw socket. Only the superuser can create a raw socket. There is no concept of a port number in a raw socket. It is possible to read and write any type of datalink frame also, if, in the above socket creation function, the first parameter is AF_PACKET and the third parameter has value *htons* (ETH_P_ALL).

The program described here receives frames from an application that sends UDP datagrams. Using the raw socket, we read the frame using the raw socket and filter out the UDP datagrams based on the protocol field in the IP header. The data link frames are read in by the raw socket using *recvfrom.*

Program

The program for filtering the UDP packets is given in filter_new.c

filter_new.c

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>

#include<linux/if_packet.h>
#include<netinet/in.h>
#include<netinet/if_ether.h>
#include<netinet/ip.h>
#include<netinet/udp.h>
#include<netinet/tcp.h>
#include<arpa/inet.h>
#include<arpa/inet.h>

void print_headers(unsigned char * , int ); void
udp_header(unsigned char*,int );
```

```
int main()
{
    int sock_r_fd, len, n;
    struct sockaddr_in servaddr; len =
    sizeof(servaddr);
    unsigned char * buffer;
    buffer  = (unsigned char *)malloc(65536); memset(buffer,0,
    65536);
    sock_r_fd =  socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL)); if(sock_r_fd < 0)
     {
         perror("Error in socket creation"); exit(1);
     }
    while(1)
    {
        n = recvfrom(sock_r_fd, buffer, 65536, 0,  (struct sockaddr*)&servaddr,   &len);
     print_headers(buffer, n);
     }
}
void print_headers(unsigned char * _buffer, int _n)
{
    static int tcp =0;
    static int udp =0;
    static int other_headers =0;
    struct iphdr * ip = (struct iphdr *) (_buffer + sizeof(struct ethhdr)); if(ip-
    >protocol == 6)
     {
        ++tcp;
     }
    else if(ip->protocol == 17)
     {
```

```
        ++udp; udp_header(_buffer,
        _n);
     }
    else
     {
        other_headers++;
     }

        printf("\nTCP = %d, UDP = %d, others = %d\n", tcp, udp, other_headers);
}

void udp_header(unsigned char* buffer,int buflen)
{

    int i=0;
    int remaining_data, iphdrlen; struct
    sockaddr_in src, dest; unsigned char *
    data;
    struct iphdr *ip; struct
    udphdr *udp;

    struct ethhdr *eth = (struct ethhdr *)(buffer); printf("Ethernet
    Header\n");

    printf("Source address: %.2x.%.2x.%.2x.%.2x.%2x.%2x\n", eth->h_source[0], eth->h_source[1], eth->h_source[2],
eth->h_source[3], eth->h_source[4], eth->h_source[5]);

    printf("Dest address: %.2x.%.2x.%.2x.%.2x.%2x.%2x\n", eth->h_source[0], eth->h_source[1], eth-
>h_source[2], eth->h_source[3], eth->h_source[4], eth->h_source[5]);

    printf("Protocol: = %d\n", eth->h_proto);

    ip = (struct iphdr*)(buffer + sizeof(struct ethhdr)); iphdrlen
    =ip->ihl*4;
    memset(&src, 0, sizeof(src));
    src.sin_addr.s_addr = ip->saddr; memset(&dest,
    0, sizeof(dest)); dest.sin_addr.s_addr = ip-
    >daddr;

    printf(" Source IP = %s\n", inet_ntoa(src.sin_addr));

    udp = (struct udphdr*)(buffer + iphdrlen + sizeof(struct ethhdr));

    printf("Source Port: %d\n" , ntohs(udp->source)); printf("Destination
    Port: %d\n" , ntohs(udp->dest)); printf("UDP Length: %d\n" , ntohs(udp-
    >len));
```

```
    printf("UDP Checksum: %d\n" , ntohs(udp->check));

}
```

Output

```
anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 17$ sudo ./filter akanjama
Ethernet Header
Source address: 00.00.00.00. 0. 0
Dest address: 00.00.00.00. 0. 0
Protocol: = 8
Source IP = 127.0.0.1
Source Port: 53562
Destination Port: 5016
UDP Length: 24
UDP Checksum: 65067

TCP = 0, UDP = 1, others = 0 Ethernet
Header
Source address: 00.00.00.00. 0. 0
Dest address: 00.00.00.00. 0. 0
Protocol: = 8
 Source IP = 127.0.0.1
Source Port: 53562
Destination Port: 5016
UDP Length: 24
UDP Checksum: 65067

TCP = 0, UDP = 2, others = 0

TCP = 0, UDP = 2, others = 1

TCP = 0, UDP = 2, others = 2 Ethernet
Header
Source address: 00.00.00.00. 0. 0
Dest address: 00.00.00.00. 0. 0
Protocol: = 8
 Source IP = 127.0.0.1
Source Port: 53562
Destination Port: 5016
UDP Length: 408
UDP Checksum: 65451

TCP = 0, UDP = 3, others = 2 Ethernet
Header
Source address: 00.00.00.00. 0. 0
```

Dest address: 00.00.00.00. 0. 0
Protocol: = 8
Source IP = 127.0.0.1
Source Port: 53562
Destination Port: 5016
UDP Length: 408
UDP Checksum: 65451

TCP = 0, UDP = 4, others = 2

TCP = 0, UDP = 4, others = 3

TCP = 0, UDP = 4, others = 4 Ethernet Header
Source address: 00.00.00.00. 0. 0
Dest address: 00.00.00.00. 0. 0
Protocol: = 8
Source IP = 127.0.0.1
Source Port: 53562
Destination Port: 5016
UDP Length: 408
UDP Checksum: 65451

TCP = 0, UDP = 5, others = 4 Ethernet Header
Source address: 00.00.00.00. 0. 0
Dest address: 00.00.00.00. 0. 0
Protocol: = 8
Source IP = 127.0.0.1
Source Port: 53562
Destination Port: 5016
UDP Length: 408
UDP Checksum: 65451

TCP = 0, UDP = 6, others = 4

TCP = 0, UDP = 6, others = 5

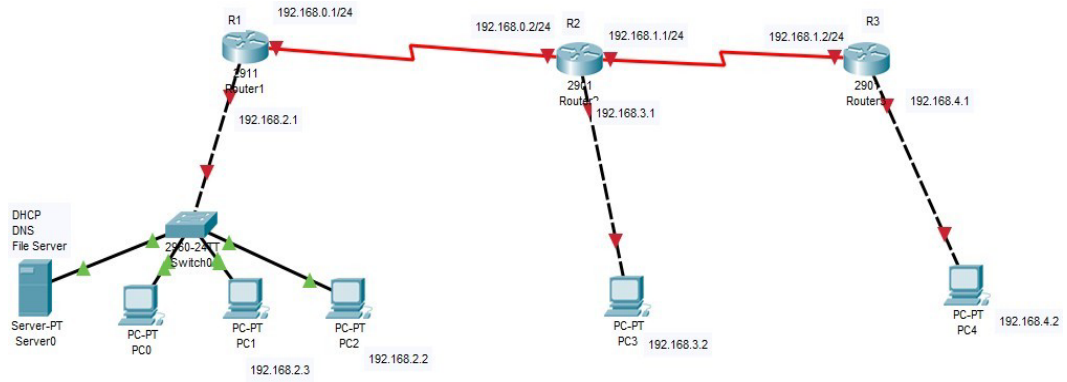
TCP = 0, UDP = 6, others = 6
^Canil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 17$

Experiment 18

Design and Configuration of network and services

Aim: To design and configure a network with multiple subnets with wired and wireless LANs using required network devices. Configure the following services in the network- TELNET, SSH, FTP server, Webserver, File server, DHCP server and DNS server.*

Description:



Design

The above topology shows 3 subnets .The network is designed with server, hosts, switch and Routers.

Configuration

Subnet 1 contains 3 hosts.

Hosts are configured with IP addresses 192.168.2.2, 192.168.2.3, 192.168.2.4 and the server with 192.168.2.100 . These hosts are connected to the fast Ethernet port of the router through the switch

Server is configured with DHCP, DNS etc. Subnet 2 contains one host with IP address 192.168.3.2 Subnet 3 contains one host with IP address 192.168.4.2. Routers R1, R2, R3 are connected through serial cable (R1,R2,R3)

We need to configure static routing in the routers to enable communication of hosts in different subnets

Router 1

Configure the router serial interface with IP address 192.168.0.1 255.255.255.0 Configure the fast Ethernet port with IP address 192.168.2.1

Configure the static route -ip route 192.168.1.0 255.255.255.0 s0/0 ip route 192.168.3.0 255.255.255.0 s0/0

ip route 192.168.4.0 255.255.255.0 s0/0

Router 2

Configure the router serial interface 0 with IP address 192.168.0.2 255.255.255.0 Configure the router serial interface 1 with IP address 192.168.1.1 255.255.255.0 Configure the fast Ethernet port with IP address 192.168.3.1

Configure the static route -ip route 192.168.2.0 255.255.255.0 s0/0 ip route 192.168.4.0 255.255.255.0 s0/1

Router 3

Configure the router serial interface with IP address 192.168.1.2 255.255.255.0 Configure the fast Ethernet port with IP address 192.168.4.1

Configure the static route -ip route 192.168.0.0 255.255.255.0 s0/0 ip route 192.168.1.0 255.255.255.0 s0/0

ip route 192.168.2.0 255.255.255.0 s0/0

Output

Files and services can be accessed across subnets.

Experiment 19

Simulation using NS2 simulator

Aim: To Install network simulator NS-2 in any of the Linux operating system and simulate wired and wireless scenarios.

Description

Basics of Computer Network Simulation:

A simulation can be thought of as a flow process of network entities (e.g., nodes, packets). As these entities move through the system, they interact with other entities, join certain activities, trigger events, cause some changes to the state of the system, and leave the process. From time to time, they contend or wait for some type of resources. This implies that there must be a logical execution sequence to cause all these actions to happen in a comprehensible and manageable way.

Introduction to Network Simulator 2 (NS2)

Network Simulator (Version 2), widely known as NS2, is simply an event driven simulation tool that has proved useful in studying the dynamic nature of communication networks. Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2. In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviors.

Basic Architecture

NS2 provides users with an executable command *ns* which takes on input argument, the name of a Tcl simulation scripting file. In most cases, a simulation trace file is created, and is used to plot graph and/or to create animation. NS2 consists of two key languages: C++ and Object-oriented Tool Command Language (OTcl). While the C++ defines the internal mechanism (i.e., a backend) of the simulation objects, the OTcl sets up simulation by assembling and configuring the objects as well as scheduling discrete events (i.e., a frontend).

Ns2 Installation On Ubuntu 16.04

1) Download **'ns-allinone-2.35'** from :
http://sourceforge.net/projects/nsnam/files/allinone/ns-allinone-2.35/ns-allinone-2.35.tar.gz/download

2) Extract the downloaded zip file 'ns-allinone-2.35.tar.gz file' to desktop.

3) Now you need to download some essential packages for ns2.Type the below lines one by one on the terminal window

"sudo apt-get update" "sudo apt-
get dist-upgrade" "sudo apt-get
update" "sudo apt-get gcc"
"sudo apt-get install build-essential autoconf automake" "sudo apt-
get install tcl8.5-dev tk8.5-dev"
"sudo apt-get install perl xgraph libxt-dev libx11-dev libxmu-dev"

4) Now change your directory(here i have already extracted the downloaded files to desktop,so my location is desktop) type the following codes in the command window to install NS2.

cd Desktop
cd ns-allinone-2.35
./install

The installation procedure will take a few minutes.....

5) After compleating the installation type the following command in the command window gedit ~/.bashrc

6) Now an editor window appears,please copy and paste the follwing codes in the end of the text file (note that '/home/abhiram/Desktop/ns-allinone-2.35/octl-1.14' in each line in the below code should be replaced with your location where the 'ns-allinone-2.35.tar.gz'file is extracted)

```
# LD_LIBRARY_PATH
OTCL_LIB=/home/abhiram/Desktop/ns-allinone-2.35/otcl-1.14
NS2_LIB=/home/abhiram/Desktop/ns-allinone-2.35/lib
X11_LIB=/usr/X11R6/lib
USR_LOCAL_LIB=/usr/local/lib export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$OTCL_LIB:$NS2_LIB:$X11_LIB:$USR_LOCAL
_LIB

# TCL_LIBRARY
TCL_LIB=/home/abhiram/Desktop/ns-allinone-2.35/tcl8.5.10/library
```

USR_LIB=/usr/lib
export TCL_LIBRARY=$TCL_LIB:$USR_LIB

# PATH
XGRAPH=/home/abhiram/Desktop/ns-allinone-2.35/bin:/home/abhiram/Desktop/ns-allinone-2.35/tcl8.5.10/unix:/home/abhiram/Desktop/ns-allinone-2.35/tk8.5.10/unix NS=/home/abhiram/Desktop/ns-allinone-2.35/ns-2.35/
NAM=/home/abhiram/Desktop/ns-allinone-2.35/nam-1.15/ PATH=$PATH:$XGRAPH:$NS:$NAM

7) Save and close the text editor and then type the following command on the terminal source

~/.bashrc

8) Close the terminal window and start a new terminal window and now change the directory to ns-2.35 and validate ns-2.35 by exicuting the following command (it takes 30 to 45 minutes)

cd ns-2.35
./validate

9) If the installation is successful, then you will be able to see % at the command prompt while typing the following command

ns

10) Now type

exit

Ns2 Commands

Creating wired node

Simulator class has a procedure node which return a node.command

[simulator instance] node

Changing shape of Node

Shape of node can be changed using shape procedure of node class. Shapes in ns2 available are box, hexagon and circle. Default shape is circle. Once a shape is defined then it can't be changed after simulation starts.

Syntax:

```
[node-instance] shape <circle|hexagon|box>
```

Reset all agents of Node

reset command is used to reset all agents attached to node.

Syntax:

Fetch id of Node

```
[node-instance] reset
```

Node id can be fetched in ns2. Sytanx:

```
[node-instance] id
```

Attach agent to node on a specific port

Agent can be attached to node on a specified port number.

Sytanx:

Attach Label to node

```
[node-instance] attach-agent [agent-instance] optional:<Port_no
```

Label can be attached to node in NAM for better understanding. Sytanx:

```
[node-instance] label [label]
```

Fetch next available port number

alloc-port is used to access available port number on a node for new agents. Sytanx:

```
[node-instance] alloc_port null_agent
```

Fetch list of neighbors

Every node maintains list of neighbors which are connected to that node. Syntax:
[*node-instance] neighbor*

Link Commands

In *ns2*, nodes can be connected in two ways, simplex and duplex. Simplex connection allows one-way communication and duplex connection allows two-way communication. Each type requires bandwidth, delay and type of queue for configuration.

Bandwidth is specified in Mbps(Mb) and delay is specified in milli seconds (ms).
Type of Queue available in ns2: DropTail, RED, CBQ, FQ, SFQ, DRR.

Syntax:
$ns simplex-link/duplex-link [node-instance1] [node-instance2] bandwidth delay Q-Type

Setting orientation of links in NAM

Position of links can be defined for better representation of network in NAM(Network AniMator).
Syntax : *$ns simplex-link-op/duplex-link-op [node-instance1] [node-instance2] orient [orientation]*
Orientation can be right, left, up, down, up-right, up-left etc.

Agent Class

For every node transport mechanism need to be defined to send data. These transport mechanism in ns2 defined using agent. For example FTP application requires TCP transport protocol, that's why TCP agent need to be associated with sending node. All agents are subclass of Agent class

Attach agent to node

Before attaching a agent to node, instance of agent need to created. Instance of above given agent can be instantiated as

$ [simulator-instance] attach-agent [node-instance] [agent-instance]

Fetch port number to which agent is attached

Every agent store port number assigned to it. This port number can be accessed by using following command
Syntax:
$ [agent-instance] port Connect
agent to another agent

For communication agent need to be connected to another agent to which it wants to send data.
Syntax : $ [simulator-instance] connect [agent-instance] [agent-instance]

Analysing trace file

In *ns2* simulation result stored in trace files. These file formats need to be understood for analysing result. Here we will discuss general trace file and nam trace file.

$set trace [open result.tr w]

$ns trace-all $trace

LAN simulation on NS2(CSMA)

- Carrier Sense Multiple Access (CSMA) is a network protocol that listens to or senses network signals on the carrier/medium before transmitting any data.
- CSMA is implemented in Ethernet networks with more than one computer or network device attached to it.

Type of CSMA Protocols

- p-persistent CSMA
- I-persistent CSMA
- Non- Persistent CSMA

Features of CSMA routing protocol:

- Low packet delays
- To provide QoS
- Low packet loss
- Collision avoidance etc.

A research topic in CSMA is:

- For channel sharing resource management.
- For collision avoidance.
- Collision detection problem.
- Discuss energy related issues.
- To manage medium access.
- Achieve scalability.
- Considered mobility .

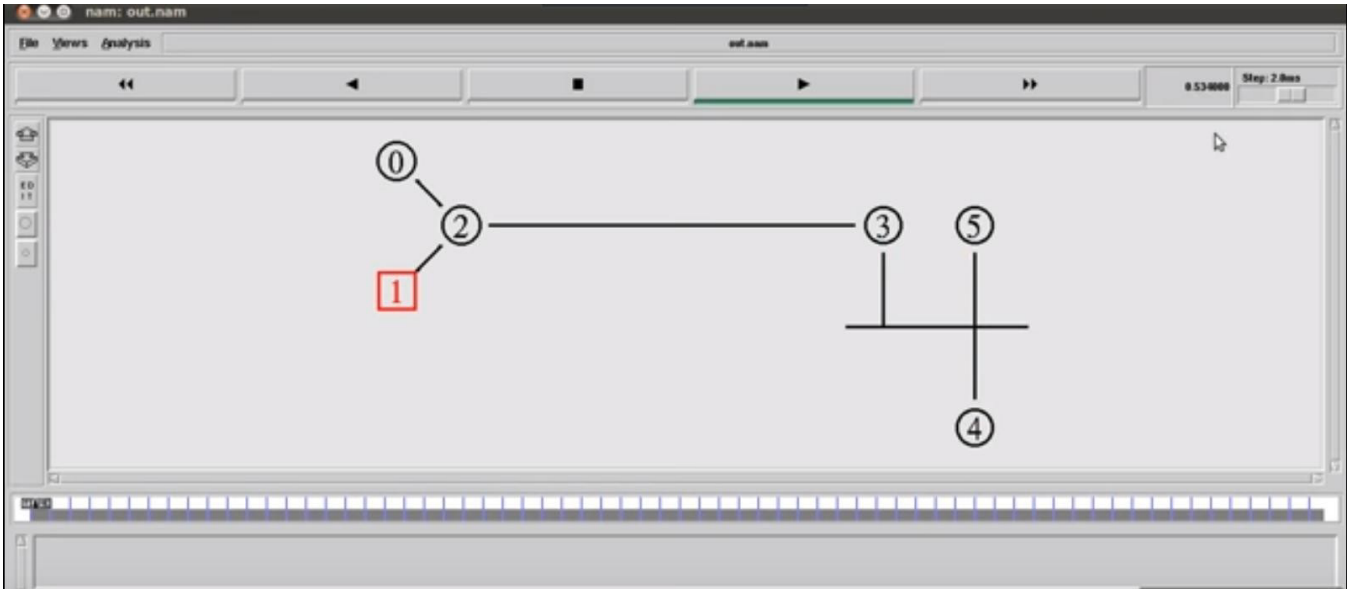
Simulation of CSMA

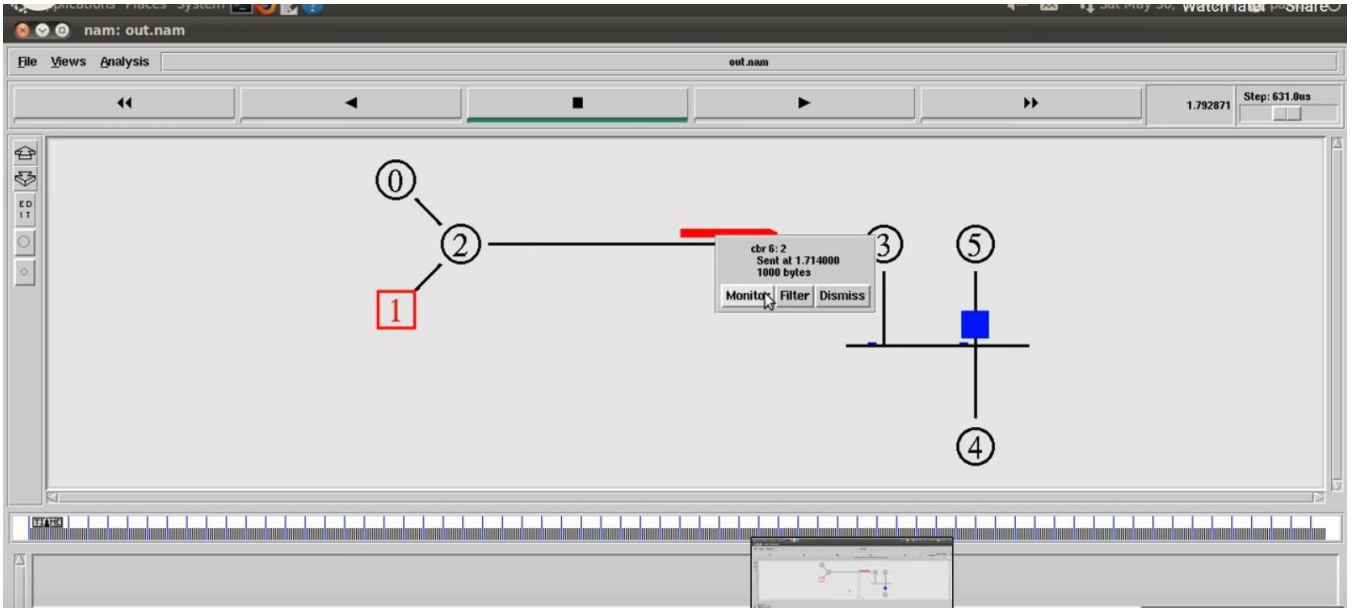
```
set ns [new Simulator] #define
color for data flows
$ns color 1 Blue
$ns color 2 Red #open
 tracefiles
```

```
set tracefile1 [open out.tr w] set
winfile [open winfile w]
$ns trace-all $tracefile1 #open
nam file
set namfile [open out.nam w]
$ns namtrace-all $namfile #define
the finish procedure proc finish {} {
        global ns tracefile1 namfile
        $ns flush-trace
        close $tracefile1
        close $namfile
        exec nam out.nam & exit
        0
        }
#create six nodes set
n0 [$ns node] set n1
[$ns node] set n2
[$ns node] set n3
[$ns node] set n4
[$ns node] set n5
[$ns node]
$n1 color Red
$n1 shape box

#create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns simplex-link $n2 $n3 0.3Mb 100ms DropTail
$ns simplex-link $n3 $n2 0.3Mb 100ms DropTail
set lan [$ns newLan "$n3 $n4 $n5" 0.5Mb 40ms LL Queue/DropTail MAC/Csma/Cd Channel]

#Give node position
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns simplex-link-op $n2 $n3 orient right
$ns simplex-link-op $n3 $n2 orient left

#set queue size of link(n2-n3) to 20
$ns queue-limit $n2 $n3 20

#setup TCP connection
set tcp [new Agent/TCP/Newreno]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink/DelAck]
$ns attach-agent $n4 $sink
```

```
$ns connect $tcp $sink
$tcp set fid_ 1
$tcp set packet_size_ 552

#set ftp over tcp connection set ftp
[new Application/FTP]
$ftp attach-agent $tcp

#setup a UDP connection set
udp [new Agent/UDP]
$ns attach-agent $n1 $udp set
null [new Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_ 2

#setup a CBR over UDP connection set cbr
[new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 0.01Mb
$cbr set random_ false
#scheduling the events
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 124.0 "$ftp stop"
$ns at 125.5 "$cbr stop"
proc plotWindow {tcpSource file} { global ns
set time 0.1
set now [$ns now]
set cwnd [$tcpSource set cwnd_] puts
$file "$now $cwnd"
 $ns at [expr $now+$time] "plotWindow $tcpSource $file"
}
$ns at 0.1 "plotWindow $tcp $winfile"
 $ns at 125.0 "finish"
 $ns run
```

Output





AODV

An Ad Hoc On-Demand Distance Vector (AODV) is a routing protocol designed for wireless and mobile ad hoc networks. This protocol establishes routes to destinations on demand and supports both unicast and multicast routing.The AODV protocol builds routes between nodes only if they are requested by source nodes. AODV is therefore considered an on-demand algorithm and does not create any extra traffic for communication along links. The routes are maintained as long as they are required by the sources.

In AODV, networks are silent until connections are established. Network nodes that need connections

broadcast a request for connection. The remaining AODV nodes forward the message and record the node that requested a connection. Thus, they create a series of temporary routes back to the requesting node.
A node that receives such messages and holds a route to a desired node sends a backward message through temporary routes to the requesting node. The node that initiated the request uses the route containing the least number of hops through other nodes. The entries that are not used in routing tables are recycled after some time. If a link fails, the routing error is passed back to the transmitting node and the process is repeated.

Simulation of AODV

A 3-node example for ad-hoc simulation with DSDV # Define

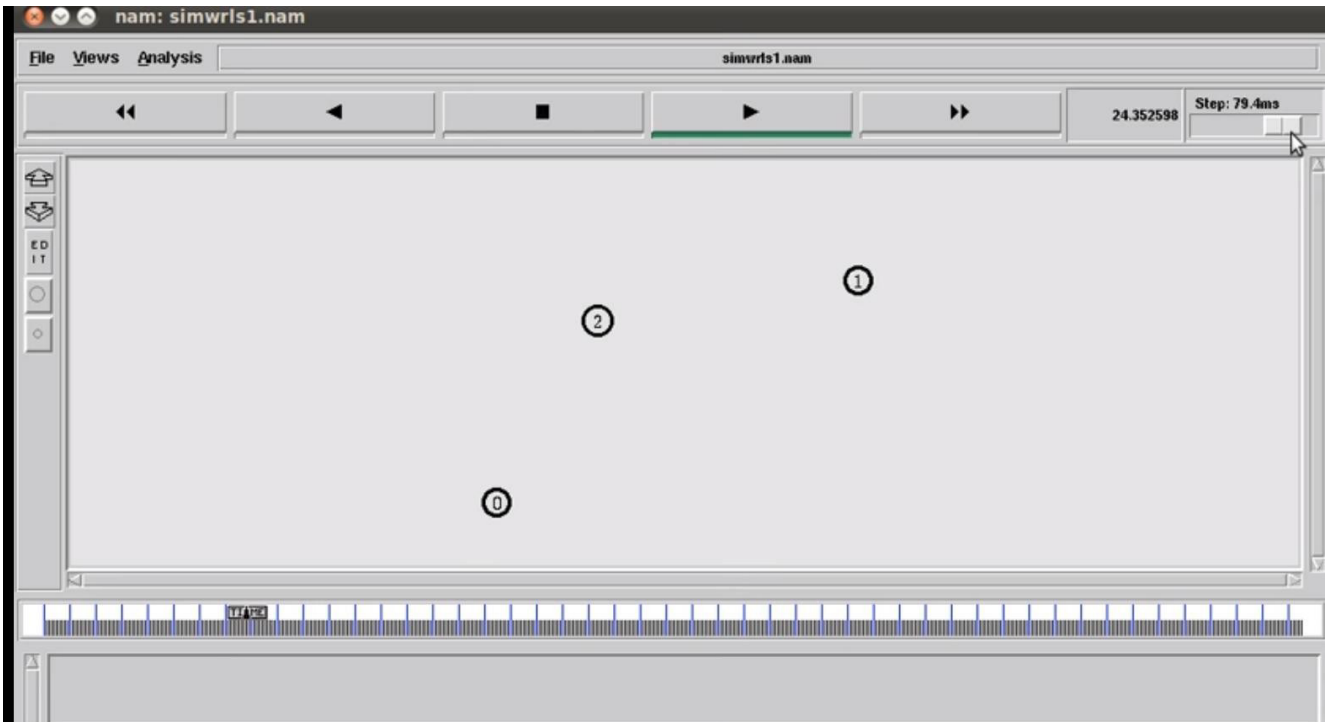
options

```
set val(chan)           Channel/WirelessChannel          ;# channel type
set val(prop)           Propagation/TwoRayGround  ;# radio-propagation model set
val(netif)              Phy/WirelessPhy                  ;# network interface type
set val(mac)            Mac/802_11                       ;# MAC type
set val(ifq)            Queue/DropTail          ;# interface queue type set
val(ll)                 LL                               ;# link layer type
set val(ant)            Antenna/OmniAntenna              ;# antenna model set
val(ifqlen)             50                        ;# max packet in ifq
set val(nn)             3                         ;# number of mobilenodes set
val(rp)                 AODV                        ;# routing protocol
set val(x)              500                                ;# X dimension of topography
set val(y)              400                                ;# Y dimension of topography
set val(stop)                150                           ;# time of simulation end

set ns                  [new Simulator]
set tracefd          [open simple-dsdv.tr w] set
windowVsTime2 [open win.tr w]  set namtrace
                        [open simwrls1.nam w]

$ns trace-all $tracefd
$ns use-newtrace
$ns namtrace-all-wireless $namtrace $val(x) $val(y)

# set up topography object
set topo          [new Topography]

$topo load_flatgrid $val(x) $val(y)

create-god $val(nn)
```

```
#
#  Create nn mobilenodes [$val(nn)] and attach them to the channel. #

# configure the nodes
        $ns node-config -adhocRouting $val(rp) \
                                -llType $val(ll) \
                                -macType $val(mac) \
                                -ifqType $val(ifq) \
                                -ifqLen $val(ifqlen) \
                                -antType $val(ant) \
                                -propType $val(prop) \
                                -phyType $val(netif) \
                                -channelType $val(chan) \
                                -topoInstance $topo \
                                -agentTrace ON \
                                -routerTrace ON \
                                -macTrace OFF \
                                -movementTrace  ON

            for {set i 0} {$i < $val(nn) } { incr i } { set
                    node_($i) [$ns node]
            }

# Provide initial location of mobilenodes
$node_(0) set X_ 5.0
$node_(0) set Y_ 5.0
$node_(0) set Z_ 0.0

$node_(1) set X_ 490.0
$node_(1) set Y_ 285.0
$node_(1) set Z_ 0.0

$node_(2) set X_ 150.0
$node_(2) set Y_ 240.0
$node_(2) set Z_ 0.0

# Generation of movements
$ns at 10.0 "$node_(0) setdest 250.0 250.0 3.0"
$ns at 15.0 "$node_(1) setdest 45.0 285.0 5.0"
$ns at 110.0 "$node_(0) setdest 480.0 300.0 5.0"

# Set a TCP connection between node_(0) and node_(1) set tcp [new
Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
```

```
$ns attach-agent $node_(0) $tcp
$ns attach-agent $node_(1) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 10.0 "$ftp start"



# Define node initial position in nam for {set i
0} {$i < $val(nn)} { incr i } { # 30 defines the
node size for nam
$ns initial_node_pos $node_($i) 30
}

# Telling nodes when the simulation ends for {set i
0} {$i < $val(nn) } { incr i } {
    $ns at $val(stop) "$node_($i) reset";
}

# ending nam and the simulation
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "stop"
$ns at 150.01 "puts \"end simulation\" ; $ns halt" proc stop {}
{
    global ns tracefd namtrace
    $ns flush-trace
    close $tracefd
    close $namtrace
    exec nam simwrls1.nam &
}

$ns run
```

Output


nam: simwrls1.nam
File Views Analysis
simwrls1.nam
24.352598
Step: 79.4ms
EDIT
1
2
0


145.964251
Step: 79.4ms
EDIT
1
2
0

END