



# NEHRU COLLEGE OF ENGINEERING AND RESEARCH CENTRE

(NAAC Accredited)



*(Approved by AICTE, Affiliated to APJ Abdul Kalam Technological University, Kerala)*

**Pampady, Thiruvilwamala (PO), Thrissur (DT), Kerala 680 588**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**



## LAB MANUAL



### ***CS334 NETWORK PROGRAMMING LAB***

#### **VISION OF THE INSTITUTION**

To mould true citizens who are millennium leaders and catalysts of change through excellence in education.

#### **MISSION OF THE INSTITUTION**

NCERC is committed to transform itself into a center of excellence in Learning and Research in Engineering and Frontier Technology and to impart quality education to mould technically competent citizens with moral integrity, social commitment and ethical values.

We intend to facilitate our students to assimilate the latest technological know-how and to imbibe discipline, culture and spiritually, and to mould them in to technological giants, dedicated research scientists and intellectual leaders of the country who can spread the beams of light and happiness among the poor and the underprivileged.

## **ABOUT DEPARTMENT**

- ◆ Established in: 2002
- ◆ Course offered: B. Tech in Computer Science and Engineering  
M.Tech in Computer Science and Engineering  
M.Tech in Cyber Security
- ◆ Approved by AICTE New Delhi and Accredited by NAAC
- ◆ Certified by ISO 9001:2015
- ◆ Affiliated to the A P J Abdul Kalam Technological University.

## **DEPARTMENT VISION**

Producing Highly Competent, Innovative and Ethical Computer Science and Engineering Professionals to facilitate continuous technological advancement.

## **DEPARTMENT MISSION**

1. To Impart Quality Education by creative Teaching Learning Process
2. To Promote cutting-edge Research and Development Process to solve real world problems with emerging technologies.
3. To Inculcate Entrepreneurship Skills among Students.
4. To cultivate Moral and Ethical Values in their Profession.

## **PROGRAMME EDUCATIONAL OBJECTIVES**

**PEO1:** Graduates will be able to Work and Contribute in the domains of Computer Science and Engineering through lifelong learning.

**PEO2:** Graduates will be able to Analyse, design and development of novel Software Packages, Web Services, System Tools and Components as per needs and specifications.

**PEO3:** Graduates will be able to demonstrate their ability to adapt to a rapidly changing environment by learning and applying new technologies.

**PEO4:** Graduates will be able to adopt ethical attitudes, exhibit effective communication skills, Team work and leadership qualities.

## PROGRAM OUTCOMES (POS)

### Engineering Graduates will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.
- 13.

## PROGRAM SPECIFIC OUTCOMES (PSO)

**PSO1:** Ability to Formulate and Simulate Innovative Ideas to provide software solutions for Real-time Problems and to investigate for its future scope.

**PSO2:** Ability to learn and apply various methodologies for facilitating development of high quality System Software Tools and Efficient Web Design Models with a focus on performance optimization.

**PSO3:** Ability to inculcate the Knowledge for developing Codes and integrating hardware/software products in the domains of Big Data Analytics, Web Applications and Mobile Apps to create innovative career path and for the socially relevant issues.

## COURSE OUTCOME

<b>CO 1</b>	Demonstrate network configuration tools
<b>CO 2</b>	Analyze the network programming skill
<b>CO 3</b>	Use network related commands and configuration files in Linux Operating System.
<b>CO 4</b>	Develop operating system and network application programs
<b>CO 5</b>	Analyze network traffic using network monitoring tools.
<b>CO 6</b>	Identify the network trouble shooting command

## CO VS PO'S AND PSO'S MAPPING

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO 1	3											
CO 2	3											
CO 3		3	3									
CO 4		3	3									
CO 5	3		2									
CO 6		2										

**Note: H-Highly correlated=3, M-Medium correlated=2, L-Less correlated=1**

	PSO1	PSO2	PSO3
CO1	3	2	
CO2	2	3	
CO3	2	3	
CO4			3
CO5			3
CO6			2

### **PREPARATION FOR THE LABORATORY SESSION**

#### **GENERAL INSTRUCTIONS TO STUDENTS**

1. Read carefully and understand the description of the experiment in the lab manual. You may go to the lab at an earlier date to look at the experimental facility and understand it better. Consult the appropriate references to be completely familiar with the concepts and hardware.
2. Make sure that your observation for previous week experiment is evaluated by the faculty member and you have transferred all the contents to your record before entering to the lab/workshop.
3. At the beginning of the class, if the faculty or the instructor finds that a student is not adequately prepared, they will be marked as absent and not be allowed to perform the experiment.
4. Bring necessary material needed (writing materials, record etc)
5. Please actively participate in class and don't hesitate to ask questions. Please utilize the teaching assistants fully. To encourage you to be prepared and to read the lab manual before coming to the laboratory, unannounced questions may be asked at any time during the lab.

6. Carelessness in personal conduct or in handling equipment may result in serious injury to the individual or the equipment.
7. Students must follow the proper dress code inside the laboratory. Long hair should be tied back.
8. Maintain silence, order and discipline inside the lab. Don't use cell phones inside the laboratory.
9. Any injury no matter how small must be reported to the instructor immediately.
10. Check with faculty members one week before the experiment to make sure that you have the handout for that experiment and all the apparatus.

#### AFTER THE LABORATORY SESSION

1. Clean up your work area.
2. Check with the technician before you leave.
3. Make sure you understand what kind of report is to be prepared and due submission of record is next lab class.

#### MAKE-UPS AND LATE WORK

Students must participate in all laboratory exercises as scheduled. They must obtain permission from the faculty member for absence, which would be granted only under justifiable circumstances. In such an event, a student must make arrangements for a make-up laboratory, which will be scheduled when the time is available after completing one cycle. Late submission will be awarded less mark for record and internals and zero in worst cases.

#### LABORATORY POLICIES

1. Food, beverages & mobile phones are not allowed in the laboratory at any time.
2. Do not sit or place anything on instrument benches.
3. Organizing laboratory experiments requires the help of laboratory technicians and staff. Be punctual.

## SYLLABUS

Course code	Course Name	L-T-P-Credits	Year of Introduction
CS334	Network Programming Lab	0-0-3-1	2016
<b>Pre-requisite:</b> CS307 Data Communication			
<b>Course Objectives</b> <ul style="list-style-type: none"> <li>To introduce Network related commands and configuration files in Linux Operating System.</li> <li>To introduce tools for Network Traffic Analysis and Network Monitoring.</li> <li>To practice Network Programming using Linux System Calls.</li> <li>To design and deploy Computer Networks.</li> </ul>			
<b>List of Exercises/ Experiments (12 Exercises/ Experiments are to be completed . Exercises/ Experiments marked with * are mandatory)</b> <ol style="list-style-type: none"> <li>Getting started with Basics of Network configurations files and Networking Commands in Linux.</li> <li>To familiarize and understand the use and functioning of System Calls used for Operating system and network programming in Linux.</li> <li><u>Familiarization and implementation of programs related to Process and thread.</u></li> <li><u>Implement the First Readers-Writers Problem.</u></li> <li><u>Implement the Second Readers-Writers problem.</u></li> <li><u>Implement programs for Inter Process Communication using PIPE, Message Queue and Shared Memory.</u></li> <li>Implement Client-Server communication using Socket Programming and TCP as transport layer protocol.*</li> <li>Implement Client-Server communication using Socket Programming and UDP as transport layer protocol.*</li> <li>Implement a multi user chat server using TCP as transport layer protocol.*</li> <li>Implement Concurrent Time Server application using UDP to execute the program at remoteserver. Client sends a time request to the server, server sends its system time back to the client. Client displays the result.*</li> <li>Implement and simulate algorithm for Distance vector routing protocol.</li> <li>Implement and simulate algorithm for Link state routing protocol.</li> <li>Implement Simple Mail Transfer Protocol.*</li> <li>Develop concurrent file server which will provide the file requested by client if it exists. If not server sends appropriate message to the client. Server should also send its process ID (PID) to clients for display along with file or the message.*</li> <li>Using Wireshark observe data transferred in client server communication using UDP and identify the UDP datagram.</li> <li>Using Wireshark observe Three Way Handshaking Connection Establishment, Data Transfer and Three Way Handshaking Connection Termination in client server communication using TCP.</li> <li>Develop a packet capturing and filtering application using raw sockets.</li> <li>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, Web server, File server, DHCP server and DNS server.*</li> <li>Install network simulator NS-2 in any of the Linux operating system and simulate wired and wireless scenarios.</li> </ol>			
<b>Expected Outcome</b> The students will be able to <ol style="list-style-type: none"> <li>Use network related commands and configuration files in Linux Operating System.</li> <li>Develop operating system and network application programs.</li> <li>Analyze network traffic using network monitoring tools.</li> </ol>			

## LIST OF EXPERIMENTS

<b>Exp. No</b>	<b>Experiment Name</b>	<b>Page No</b>
1	Basics Of Network Configurations Files And Networking Commands In Linux.	1
2	Familiarization Of System Calls	4
3	Implementation Of First Readers Writers Problem	7
4	Implementation Of Second Readers Writers Problem	15
5	Inter-Process Communication Using Pipes, Message Queues, And Shared Memory	26
6	Implementation Of Client-Server Communication Using Socket Programming and TCP as Transport Layer Protocol	32
7	Implementation of Client-Server Communication Using Socket Programming and UDP as Transport Layer Protocol	39
8	Implementation of A Multi User Chat Server Using TCP as Transport Layer Protocol	44
9	Implementation of Concurrent Time Server Using UDP	53
10	Implementation of Simple Mail Transfer Protocol	57
11	Implementation of Concurrent File Serve	63
12	Design And Configuration of Network And Services	67



Experiment no: 1

Date:

## **BASICS OF NETWORK CONFIGURATIONS FILES AND NETWORKING COMMANDS IN LINUX.**

### **Aim**

To Familiarize with different configuration files and commands

### **Description**

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

## 5. 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.

## 6. netstat

This command gives network status information.

```
Netstat -i
```

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.

## 7. 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
```

```
--- 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.

## Result & Discussion

Familiarized with different network commands and their functionalities.

### Viva Questions:

1. What is the functionality of `ifconfig` command
2. Ping command is used for \_\_\_\_\_
3. Network status information is given by \_\_\_\_\_

Experiment No: 2

Date:

## FAMILIARIZATION OF SYSTEM CALLS

### Aim

To familiarize various system calls used in network programming

### Description

Some system calls used in 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 regarding 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* immediately 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 system 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>`.

## Result & Discussion

Familiarized with the different system calls and their functionalities.

## Viva Questions:

1. What is the functionality of socket system call
2. Fork system call is used for \_\_\_\_\_
3. Open system call is used for \_\_\_\_\_

Experiment No:3

Date:

## IMPLEMENTATION OF FIRST READERS WRITERS PROBLEM

### Aim

To implement the 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 be based on *writeBlock* variable.

### Sample Code:

The sample code for the readers writers problem is given below

```

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);
}

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

**Output**

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

**Result & Discussion**

Program is executed successfully and output is obtained

**Viva Questions:**

1. Define process and thread
2. Join function in thread is used for \_\_\_\_\_
3. Mutex is used for \_\_\_\_\_

Experiment No: 4

Date:

## IMPLEMENTATION OF SECOND READERS WRITERS PROBLEM

### Aim

To implement the 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.

### Sample Code

```
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 mutex 1");
            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**



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 1 is READING  
READER 0 is READING

### **Result & Discussion**

Program is executed successfully and output is obtained

### **Viva Questions:**

1. Define Readers writer problem
2. What is the functionality of perror
3. Define Semaphore

Experiment No: 5

Date:

## **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.

### **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, MSGHDRSIZE + mesg.message_length);

    // read from pipe and write to the standard output

    while(1)
    {
        if(n = read(readfd, &mesg, MSGHDRSIZE)) == -1)
        {
            perror("read error");
            exit(1);
        }

        if( n!= MSGHDRSIZE)
        {
            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)

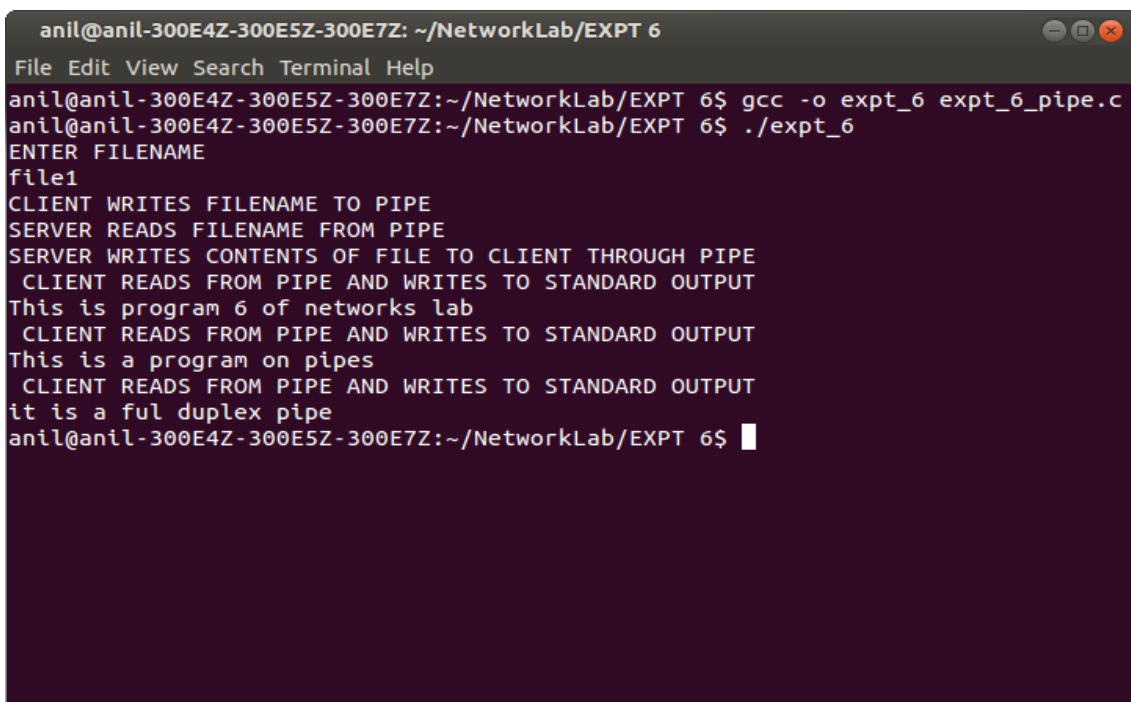
struct message {
    long message_length;
    long message_type;
    char message_data[MAXMESSAGEDATA];
};

#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 display it on the console.



```

anil@anil-300E4Z-300E5Z-300E7Z: ~/NetworkLab/EXPT 6
File Edit View Search Terminal Help
anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 6$ gcc -o expt_6 expt_6_pipe.c
anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 6$ ./expt_6
ENTER FILENAME
file1
CLIENT WRITES FILENAME TO PIPE
SERVER READS FILENAME FROM PIPE
SERVER WRITES CONTENTS OF FILE TO CLIENT THROUGH PIPE
CLIENT READS FROM PIPE AND WRITES TO STANDARD OUTPUT
This is program 6 of networks lab
CLIENT READS FROM PIPE AND WRITES TO STANDARD OUTPUT
This is a program on pipes
CLIENT READS FROM PIPE AND WRITES TO STANDARD OUTPUT
it is a full duplex pipe
anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 6$

```

**Result & Discussion**

Program is executed successfully and output is obtained

**Viva Questions:**

1. Define IPC
2. Explain the concept of shared memory
3. Pipe is used for \_\_\_\_

Experiment No: 6

Date:

## 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



*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 success 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**

```

import java.io.*;
import java.net.*;
class TCPCLIENT
{
public static void main(String args[])
{
try
{
Socket c=new Socket("127.0.0.1",9696);
BufferedReader br=new BufferedReader(new InputStreamReader(c.getInputStream()));
DataInputStream in=new DataInputStream(System.in);
while(true)
{
PrintWriter out=new PrintWriter(c.getOutputStream(),true);
System.out.println("\n enter the data to be send to server \n");
String str=in.readLine();
out.println(str);
String sw=br.readLine();
System.out.println("\n client received \n" +sw);
}}
catch(Exception e)
{}
}}

```

**Server Program**

```

import java.io.*;
import java.net.*;
class TCPSERVER
{
public static void main(String args[])
{
try
{
ServerSocket se=new ServerSocket(9696);
Socket c=se.accept();
BufferedReader br=new BufferedReader(new InputStreamReader(c.getInputStream()));
DataInputStream in=new DataInputStream(System.in);
while(true)
{
PrintWriter out=new PrintWriter(c.getOutputStream(),true);
String sw=br.readLine();
System.out.println(" \n senderreceived data \n" +sw);
System.out.println("\n enter the data to client \n");
String str=in.readLine();
out.println(str);
}}
catch(Exception e)
{}}

```

## Output

### Server

```
anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network_lab/expt1_tcp
File Edit View Search Terminal Help
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt1_tcp$ gcc -o server server.c
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt1_tcp$ ./server 5100
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt1_tcp$
```

### Client

```
anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network_lab/expt1_tcp
File Edit View Search Terminal Help
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt1_tcp$ gcc -o client client.c
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt1_tcp$ ./client 127.0.0.1 5100
Input string to be reversed:network lab
Reversed string: bal krowten
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt1_tcp$
```

**Result & Discussion**

Program is executed successfully and output is obtained

**Viva Questions:**

1. Explain TCP Protocol
2. What is the functionality of connect system call
3. Server Socket is used for \_\_

Experiment No :7

Date:

## 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 originating 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**

```
import java.io.*;
import java.net.*;
class UDPCLIENT
{
public static void main(String args[])
{
try
{
DatagramSocket cds=new DatagramSocket();
BufferedReader brc=new BufferedReader(new InputStreamReader(System.in));
while(true)
{
System.out.println("client send \n" );
String c=brc.readLine();
byte [] sd=new byte[1024];
```

```

        sd=c.getBytes();
        InetAddress addr=InetAddress.getLocalHost();
        DatagramPacket cdps=new DatagramPacket(sd,sd.length,addr,8585);
        cds.send(cdps);
        byte[] rd=new byte[1024];
        DatagramPacket cdpr=new DatagramPacket(rd,rd.length);
        cds.receive(cdpr);
        String st=new String(cdpr.getData());
        System.out.println("client received \n"+st);
    }
}
catch (Exception e)
{
}
}
}

```

### Server Program

```

import java.io.*;
import java.net.*;
class UDPSERVER

{
    public static void main(String args[])
    {
try
{
    DatagramSocket sds=new DatagramSocket(8585);
    BufferedReader brs=new BufferedReader(new InputStreamReader(System.in));
    byte[]rd=new byte[1024];
while(true)

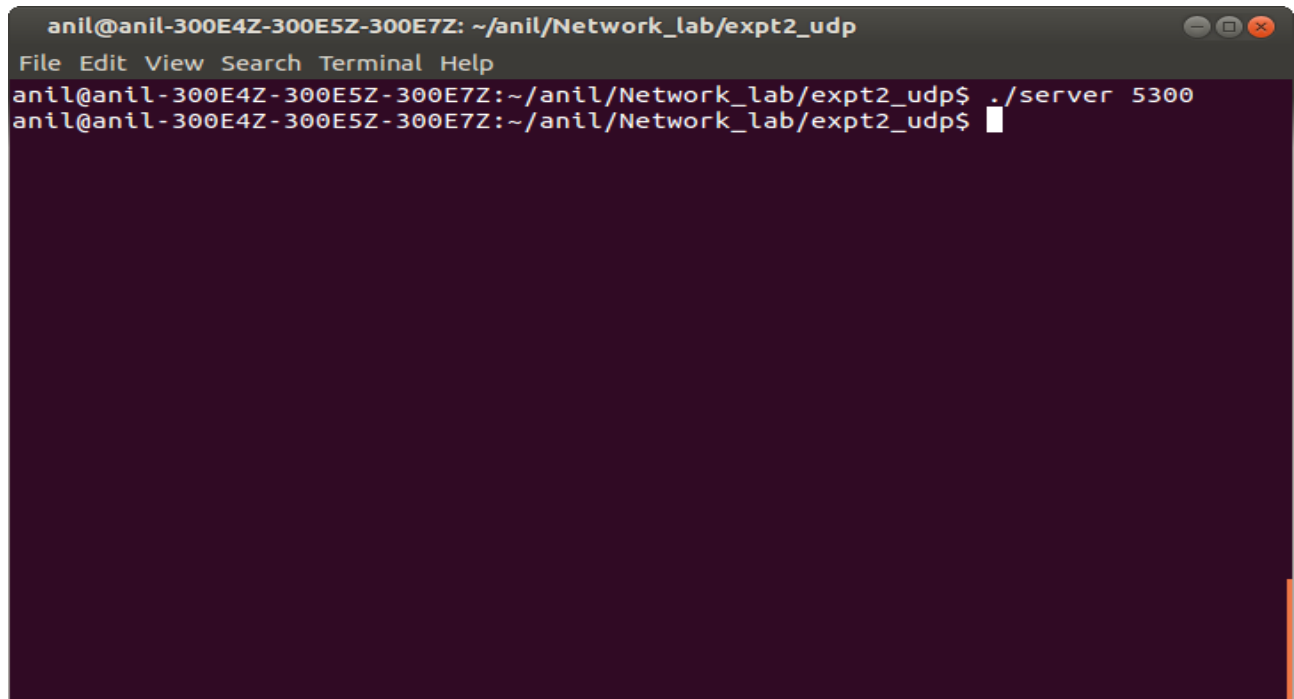
    {

        DatagramPacket sdpr=new DatagramPacket(rd,rd.length);
        sds.receive(sdpr);
        String str=new String(sdpr.getData());
        System.out.println("\n server received \n" +str);
        System.out.println("\n server send \n");
        InetAddress a=sdpr.getAddress();
        int port=sdpr.getPort();
        String s=brs.readLine();
        byte[] sd=new byte[1024];
        sd=s.getBytes();
        DatagramPacket sdps=new DatagramPacket(sd,sd.length,a,port);
        sds.send(sdps);
    }
}
catch(Exception e)
{
}
}
}

```

## Output

### Server

A terminal window with a dark purple background and light green text. The title bar at the top reads 'anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network\_lab/expt2\_udp'. Below the title bar is a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows the prompt 'anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network\_lab/expt2\_udp\$' followed by the command './server 5300'. The next line shows the prompt again with a cursor, indicating the command has been executed.

```
anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network_lab/expt2_udp
File Edit View Search Terminal Help
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt2_udp$ ./server 5300
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt2_udp$
```

### Result & Discussion

Program is executed successfully and output is obtained

### Viva Questions:

1. Explain UDP Protocol
2. What is the functionality of Buffered Reader?
3. Client Socket is used for \_\_

Experiment No: 8

Date:

## 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

fd0 fd1 fd2 fd3

0	0	0	1	
---	---	---	---	--

 $\text{Maxfd} + 1 = 4$ 

Client

-1
-1
-1
-1
.
.
.
-1

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

fd0 fd1 fd2 fd3 fd4

0	0	0	1	1	
---	---	---	---	---	--

 $\text{Maxfd} + 1 = 5$ 

Client

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

Now if a second client connects

rset

fd0 fd1 fd2 fd3 fd4 fd5

0	0	0	1	1	1	
---	---	---	---	---	---	--

Maxfd + 1 = 6



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

fd0 fd1 fd2 fd3 fd4 fd5

0	0	0	1	0	1	
---	---	---	---	---	---	--

Maxfd + 1 = 6



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**

```
import java.io.*;
import java.net.*;
import java.util.*;

public class TCPMultiClient {

    public static void main(String argv[]) throws Exception {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
            new BufferedReader(
                new InputStreamReader(System.in));

        Socket clientSocket = new Socket("127.0.0.1", 9595);

        while (true) {
            DataOutputStream outToServer =
                new DataOutputStream(
                    clientSocket.getOutputStream());

            BufferedReader inFromServer =
                new BufferedReader(
                    new InputStreamReader(
                        clientSocket.getInputStream()));

            sentence = inFromUser.readLine();

            outToServer.writeBytes(sentence + '\n');

            if (sentence.equals("EXIT")) {
                break;
            }

            modifiedSentence = inFromServer.readLine();

            System.out.println("FROM SERVER: " + modifiedSentence);
        }
        clientSocket.close();
    }
}
```

### **server**

```
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.logging.*;
```

```
public class TCPSMultiServer {

    public static void main(String argv[]) throws Exception {

        ServerSocket welcomeSocket = new ServerSocket(9595);

        Responder h = new Responder();
        // server runs for infinite time and
        // wait for clients to connect
        while (true) {
            // waiting..
            Socket connectionSocket = welcomeSocket.accept();

            // on connection establishment start a new thread for each client
            // each thread shares a common responder object
            // which will be used to respond every client request
            // need to synchronize method of common object not to have unexpected behaviour//
            Thread t = new Thread(new MyServer(h, connectionSocket));

            // start thread//
            t.start();

        }
    }

    class MyServer implements Runnable {

        Responder h;
        Socket connectionSocket;

        public MyServer(Responder h, Socket connectionSocket) {
            this.h = h;
            this.connectionSocket = connectionSocket;
        }

        @Override
        public void run() {

            while (h.responderMethod(connectionSocket)) {
                try {
                    // once an conversation with one client done,
                    // give chance to other threads
                    // so make this thread sleep//
                    Thread.sleep(5000);
                } catch (InterruptedException ex) {
                    ex.printStackTrace();
                }
            }

            try {
                connectionSocket.close();
            } catch (IOException ex) {
```



```

    Logger.getLogger(MyServer.class.getName()).log(Level.SEVERE, null, ex);
}

}

}

class Responder {

    String serverSentence;
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

    // on client process termination or
    // client sends EXIT then to return false to close connection
    // else return true to keep connection alive
    // and continue conversation//
    synchronized public boolean responderMethod(Socket connectionSocket) {
        try {

            BufferedReader inFromClient = new BufferedReader( new
            InputStreamReader(connectionSocket.getInputStream()));

            DataOutputStream outToClient = new DataOutputStream(connectionSocket.getOutputStream());

            String clientSentence = inFromClient.readLine();

            // if client process terminates it get null, so close connection//
            if (clientSentence == null || clientSentence.equals("EXIT")) {
                return false;
            }

            if (clientSentence != null) {
                System.out.println("client : " + clientSentence);
            }
            serverSentence = br.readLine() + "\n";

            outToClient.writeBytes(serverSentence);

            return true;

        } catch (SocketException e) {
            System.out.println("Disconnected");
            return false;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }
}

```

### **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

```

anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network_lab/expt3_chat
File Edit View Search Terminal Help
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt3_chat$ ./server 5600
^C
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt3_chat$

```

### Client 1

```

anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network_lab/expt3_chat
File Edit View Search Terminal Help
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt3_chat$ ./client 127.0.0.1
5600
Client 1 has joined chat
Client 2 has joined chat
2 From 1: Iam client 1, How r u
1 From 2: Iam client 2, Iam fine
Client 3 has joined chat
1 From 3: Iam client 3, Thankyou
3 From 1: Iam client 1, You are welcome
1 From2: Iam leaving
Client 2 has left chat
1 From 3: Hi client 1, client 2 has left
3 From 1: I am also leaving
^C
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt3_chat$

```

## Client 2

```

anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network_lab/expt3_chat
File Edit View Search Terminal Help
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt3_chat$
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt3_chat$ ./client 127.0.0.1
5600
Client 2 has joined chat

2 From 1: Iam client 1, How r u

1 From 2: Iam client 2, Iam fine
Client 3 has joined chat

3 From 2: Iam client 2, Welcome 3
2 From 3: Iam client 3, Thankyou 2

1 From2: Iam leaving
3 From 2: Iam leaving
^C
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt3_chat$

```

## Client 3

```

anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network_lab/expt3_chat
File Edit View Search Terminal Help
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt3_chat$ ./client 127.0.
5600
Client 3 has joined chat

3 From 2: Iam client 2, Welcome 3

1 From 3: Iam client 3, Thankyou
3 From 1: Iam client 1, You are welcome

2 From 3: Iam client 3, Thankyou 2
3 From 2: Iam leaving

Client 2 has left chat

1 From 3: Hi client 1, client 2 has left
3 From 1: I am also leaving

Client 1 has left chat

^C
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt3_chat$

```

**Result & Discussion**

Program is executed successfully and output is obtained

**Viva Questions:**

1. Explain the concept of multiple chat using TCP
2. What is the functionality of sleep option in thread
3. Server Socket is used for \_\_\_\_\_

Experiment No:9

Date:

## IMPLEMENTATION OF CONCURRENT TIME SERVER USING UDP

### Aim:

To implement a concurrent time server using UDP.

### 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 current time
      - 3.2.2 Send the time as a string to the client
      - 3.2.3 Exit
4. end of while

## **Program**

### **Client**

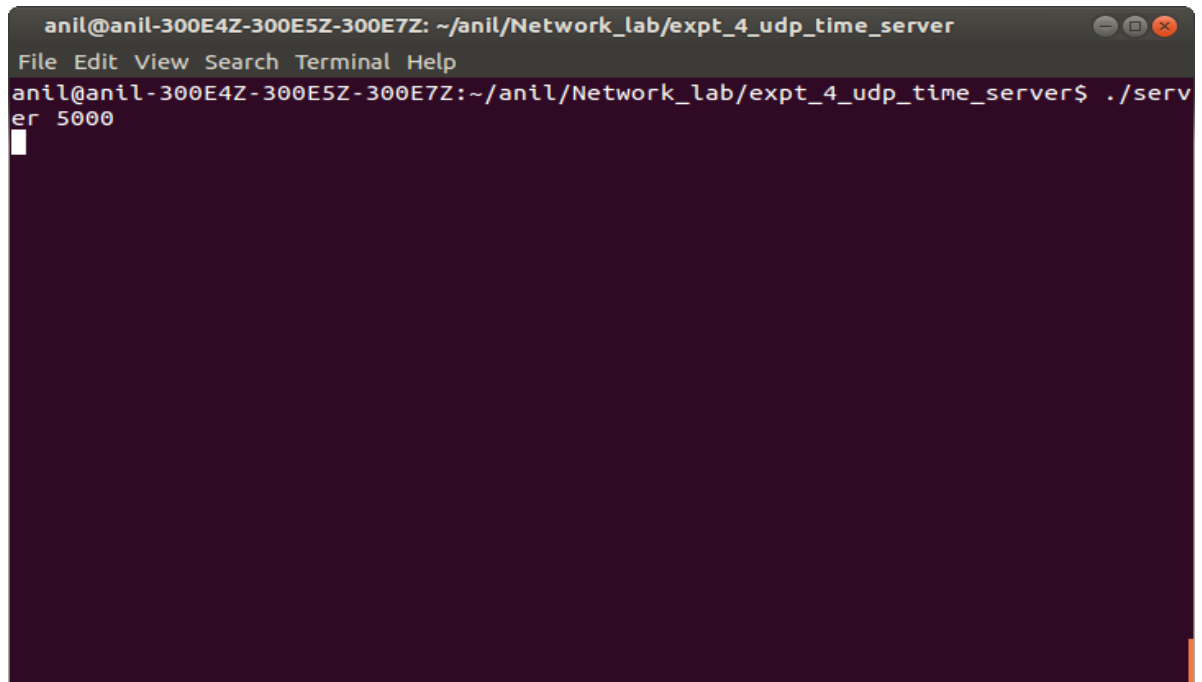
```
import socket
UDP_IP="127.0.0.1"
UDP_PORT=9797
MESSAGE="getTime"
sock=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
sock.sendto(MESSAGE,(UDP_IP,UDP_PORT))
data,addr=sock.recvfrom(1024)
print"Current time:",data
```

### **Server**

```
import socket
from time import gmtime, strftime
UDP_IP="127.0.0.1"
UDP_PORT=9797
sock=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
sock.bind((UDP_IP,UDP_PORT))
while True:
    data,addr=sock.recvfrom(1024)
    if data=='getTime':
        print"received request from:",addr
        time=strftime("%Y-%M-%D %H:%M:%S",gmtime())
        sock.sendto(time,addr)
```

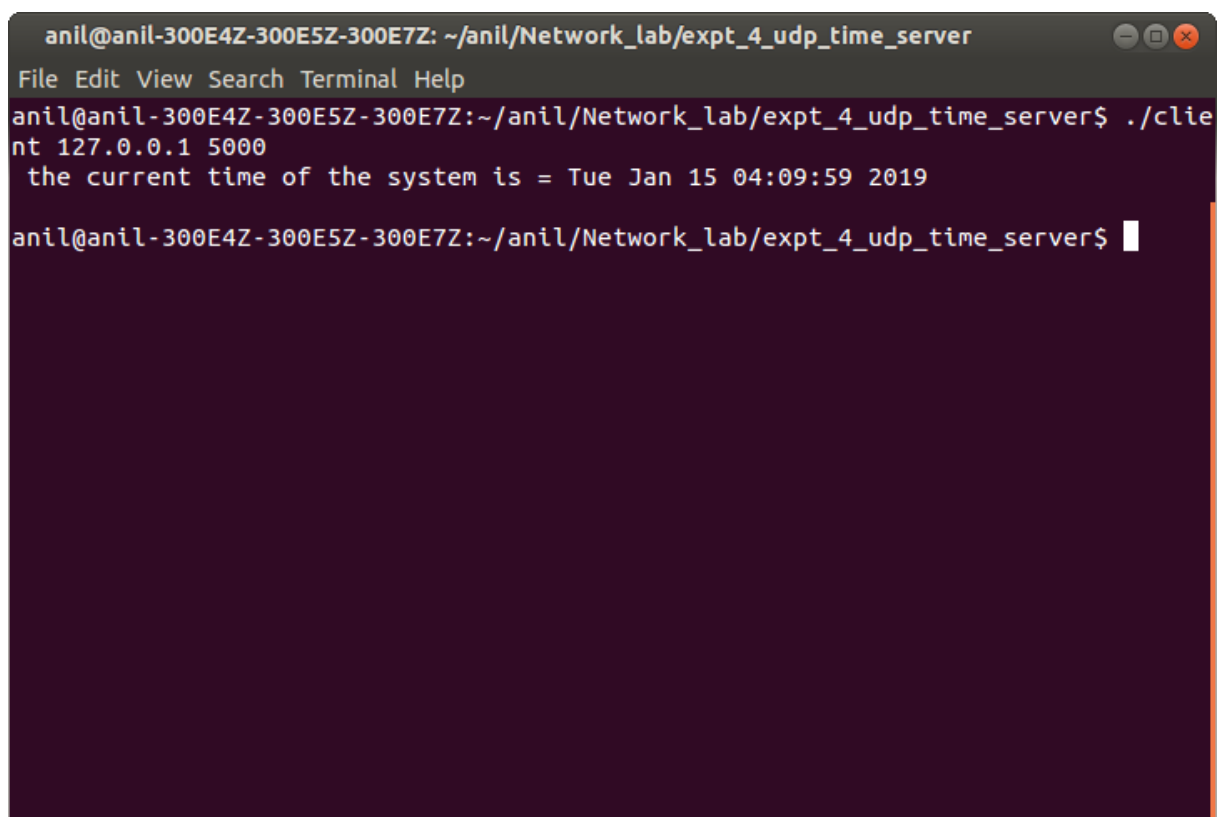
## Output

### Server



```
anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network_lab/expt_4_udp_time_server
File Edit View Search Terminal Help
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt_4_udp_time_server$ ./server 5000
█
```

### client



```
anil@anil-300E4Z-300E5Z-300E7Z: ~/anil/Network_lab/expt_4_udp_time_server
File Edit View Search Terminal Help
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt_4_udp_time_server$ ./client 127.0.0.1 5000
the current time of the system is = Tue Jan 15 04:09:59 2019
anil@anil-300E4Z-300E5Z-300E7Z:~/anil/Network_lab/expt_4_udp_time_server$ █
```

**Result & Discussion**

Program is executed successfully and output is obtained

**Viva Questions:**

1. Explain about concurrent time server?
2. What is the functionality of Port No?
3. IP Address is used for \_\_



Experiment No: 10

Date:

## 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 establishment, 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” .....Receipient 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**

```
import java.io.*;
import java.net.*;
class smtpclient
{
public static void main(String args[])
{
try
{

DatagramSocket cds=new DatagramSocket();
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
while(true)
{
System.out.println("\n FROM");
String c=br.readLine();
System.out.println("\n TO");
String d=br.readLine();
System.out.println("\n CONTENT");
String e=br.readLine();
byte[] sd=new byte[20];
sd=c.getBytes();
InetAddress addr=InetAddress.getLocalHost();
DatagramPacket cdps=new DatagramPacket(sd,sd.length,addr,8989);
cds.send(cdps);
byte [] sd1=new byte[20];
sd1=d.getBytes();
InetAddress addr1=InetAddress.getLocalHost();
DatagramPacket cdps1=new DatagramPacket(sd1,sd1.length,addr1,8989);
cds.send(cdps1);
byte [] sd2=new byte[20];
sd2=e.getBytes();
InetAddress addr2=InetAddress.getLocalHost();
DatagramPacket cdps2=new DatagramPacket(sd2,sd2.length,addr2,8989);
cds.send(cdps2);
}}
catch(Exception e){}
}
}
```

## Server

```
import java.io.*;
import java.net.*;
class SMTPSERVER
{
public static void main(String args[])
{
try
{
DatagramSocket ds=new DatagramSocket(8989);
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
byte[] rd=new byte[20];
byte[] rd1=new byte[20];
byte[] rd2=new byte[20];
while(true)
{
DatagramPacket sdpr=new DatagramPacket(rd,rd.length);
ds.receive(sdpr);
String str=new String(sdpr.getData());
System.out.println("\n");
System.out.println("\n FROM:");
System.out.println(str);
DatagramPacket sdpr1=new DatagramPacket(rd1,rd1.length);
ds.receive(sdpr1);
String str1=new String(sdpr1.getData());
System.out.println("\n");
System.out.println("\n TO");
System.out.println(str1);
DatagramPacket sdpr2=new DatagramPacket(rd2,rd2.length);
ds.receive(sdpr2);
String str2=new String(sdpr2.getData());
System.out.println("\n");
System.out.println("\n CONTENT:");
System.out.println(str2);
}
}
catch(Exception e){ }
}
}
```

## Output

## Client

```
File Edit View Search Terminal Help
anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 13$ ./client 127.0.0.1 5600
S:220 name_of_server_mail_server
S:250 Hello name_of_client_mail_server
please enter the email address of the sender:akanjama@hotmail.com
S:250 Hello <akanjama@hotmail.com>.....sender ok
please enter the email address of the receiver:akanjama@gmail.com
S:250 Hello <akanjama@gmail.com>.....Receipient ok
S:354 Enter mail,end with "." on a line by itself
Hi how r u
Iam fine
Thank you
.
S:250 messages accepted for delivery
S:221 servers mail server closing connection
anil@anil-300E4Z-300E5Z-300E7Z:~/NetworkLab/EXPT 13$
```

## **Server**

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

## **Result & Discussion**

Program is executed successfully and output is obtained

### **Viva Questions:**

1. Explain SMTP Protocol
2. What is the functionality of receive system call
3. SMTP is used for \_\_\_\_\_

Experiment No: 11

Date:

## IMPLEMENTATION OF CONCURRENT FILE SERVER

### AIM

To Develop concurrent file server which will provide the file requested by client if it exists. If not server sends appropriate message to the client. Server should also send its process ID (PID) to clients for display along with file or the message.

### ALGORITHM

Server Side:

- Step 1: Start.
- Step 2: Create Server Class.
- Step 3: Server is listening for requests from client.
- Step 4: Running infinite loop for getting client request.
- Step 5: Create input and out streams.
- Step 6: create a new thread object.
- Step 7: Create a ClientHandler class.
- Step 8: Create a constructor, and read the message from client.
- Step 9: write on output stream based on the answer from the client.
- Step 10: Close the Resources.
- Step 11: Stop.

Client Side:

- Step 1: Start.
- Step 2: Create Client Class.
- Step 3: Establish the connection with server.
- Step 4: Create input and out streams.
- Step 5: Create a loop performs the exchange of information between client and client handler.
- Step 6: printing date or time as requested by client
- Step 7: Close the resources.
- Step 8: Stop.

### PROGRAM

```
/ Java implementation of Server side
// It contains two classes: Server and ClientHandler
// Save file as Server.java

import java.io.*;
import java.text.*;
import java.util.*;
import java.net.*;
```

```

// Server class
public class Server
{
    public static void main(String[] args) throws IOException
    {
        // server is listening on port 5056
        ServerSocket ss = new ServerSocket(5056);

        // running infinite loop for getting
        // client request
        while (true)
        {
            Socket s = null;

            try
            {
                // socket object to receive incoming client requests
                s = ss.accept();

                System.out.println("A new client is connected : " + s);

                // obtaining input and out streams
                DataInputStream dis = new DataInputStream(s.getInputStream());
                DataOutputStream dos = new DataOutputStream(s.getOutputStream());

                System.out.println("Assigning new thread for this client");

                // create a new thread object
                Thread t = new ClientHandler(s, dis, dos);

                // Invoking the start() method
                t.start();

            }
            catch (Exception e){
                s.close();
                e.printStackTrace();
            }
        }
    }
}

// ClientHandler class
class ClientHandler extends Thread
{
    DateFormat fordate = new SimpleDateFormat("yyyy/MM/dd");
    DateFormat fortime = new SimpleDateFormat("hh:mm:ss");
    final DataInputStream dis;
    final DataOutputStream dos;
    final Socket s;

    // Constructor
    public ClientHandler(Socket s, DataInputStream dis, DataOutputStream dos)

```



```

{
    this.s = s;
    this.dis = dis;
    this.dos = dos;
}

@Override
public void run()
{
    String received;
    String toreturn;
    while (true)
    {
        try {

            // Ask user what he wants
            dos.writeUTF("What do you want?[Date | Time]..\n"+
                "Type Exit to terminate connection.");

            // receive the answer from client
            received = dis.readUTF();

            if(received.equals("Exit"))
            {
                System.out.println("Client " + this.s + " sends exit...");
                System.out.println("Closing this connection.");
                this.s.close();
                System.out.println("Connection closed");
                break;
            }

            // creating Date object
            Date date = new Date();

            // write on output stream based on the
            // answer from the client
            switch (received) {

                case "Date" :
                    toreturn = fordate.format(date);
                    dos.writeUTF(toreturn);
                    break;

                case "Time" :
                    toreturn = fortime.format(date);
                    dos.writeUTF(toreturn);
                    break;

                default:
                    dos.writeUTF("Invalid input");
                    break;
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

```

    }
}

try
{
    // closing resources
    this.dis.close();
    this.dos.close();

} catch(IOException e){
    e.printStackTrace();
}
}
}

```

### // Java implementation for a client

// Save file as Client.java

```

import java.io.*;
import java.net.*;
import java.util.Scanner;

// Client class
public class Client
{
    public static void main(String[] args) throws IOException
    {
        try
        {
            Scanner scn = new Scanner(System.in);

            // getting localhost ip
            InetAddress ip = InetAddress.getByName("localhost");

            // establish the connection with server port 5056
            Socket s = new Socket(ip, 5056);

            // obtaining input and out streams
            DataInputStream dis = new DataInputStream(s.getInputStream());
            DataOutputStream dos = new DataOutputStream(s.getOutputStream());

            // the following loop performs the exchange of
            // information between client and client handler
            while (true)
            {
                System.out.println(dis.readUTF());
                String tosend = scn.nextLine();
                dos.writeUTF(tosend);

                // loop performs the exchange of
                // information between client and client handler
                if(tosend.equals("Exit"))
                {

```

```

        System.out.println("Closing this connection : " + s);
        s.close();
        System.out.println("Connection closed");
        break;
    }

    // printing date or time as requested by client
    String received = dis.readUTF();
    System.out.println(received);
}

// closing resources
scn.close();
dis.close();
dos.close();
} catch (Exception e) {
    e.printStackTrace();
}
}
}

```

## Result & Discussion

Program is executed successfully and output is obtained

## Viva Questions:

1. Explain the functionality of scanner class?
2. What is the functionality of close system call
3. File Server is used for \_\_\_\_\_

Experiment No: 12

Date:

## 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, Web server, File server, DHCP server and DNS server.

### ALGORITHM

Step 1: Start.

Step2: Fixing the co-ordinate of simutaion area.

Step 3: set up topography object

Step 4: Tracing all the events and configuration

Step 5: general operational descriptor- storing the hop details in the network.

Step 6: Node Creation

Step 7: Location fixing for a single node

### PROGRAM

Wireless scenario:

#Filename: sample1.tcl

#TCL – Tool Command Language

# Simulator Instance Creation

set ns [new Simulator]

#Fixing the co-ordinate of simutaion area

set val(x) 500

set val(y) 500

# 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/PriQueue ;# 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) 2 ;# 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) 10.0 ;# time of simulation end

# set up topography object

set topo [new Topography]

\$topo load\_flatgrid \$val(x) \$val(y)

#Nam File Creation nam – network animator

set namfile [open sample1.nam w]

```

#Tracing all the events and cofiguration
$ns namtrace-all-wireless $namfile $val(x) $val(y)
#Trace File creation
set tracefile [open sample1.tr w]

#Tracing all the events and cofiguration
$ns trace-all $tracefile

# general operational descriptor- storing the hop details in the network
create-god $val(nn)

# 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

# Node Creation
set node1 [$ns node]

# Initial color of the node
$node1 color black

#Location fixing for a single node
$node1 set X_ 200
$node1 set Y_ 100
$node1 set Z_ 0
set node2 [$ns node]
$node2 color black
$node2 set X_ 200
$node2 set Y_ 300
$node2 set Z_ 0

# Label and coloring
$ns at 0.1 "$node1 color blue"
$ns at 0.1 "$node1 label Node1"
$ns at 0.1 "$node2 label Node2"

#Size of the node
$ns initial_node_pos $node1 30
$ns initial_node_pos $node2 30

# ending nam and the simulation
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"

```

```

$ns at $val(stop) "stop"

#Stopping the scheduler
$ns at 10.01 "puts \"end simulation\" ; $ns halt"

#$ns at 10.01 "$ns halt"
proc stop {} {
global namfile tracefile ns
$ns flush-trace
close $namfile
close $tracefile

#executing nam file
exec nam sample1.nam &
}

#Starting scheduler
$ns run

```

### Execution:

```
ns sample1.tcl
```

#### Wired Network:

```

# Filename: test1.tcl
#-----Event scheduler object creation-----#
set ns [new Simulator]
#-----creating trace objects-----#
set nt [open test1.tr w]
$ns trace-all $nt
#-----creating nam objects-----#
set nf [open test1.nam w]
$ns namtrace-all $nf
#-----Setting color ID-----#
$ns color 1 darkmagenta
$ns color 2 yellow
$ns color 3 blue
$ns color 4 green
$ns color 5 black

#----- Creating Network-----#
set totalNodes 3
for {set i 0} {$i < $totalNodes} {incr i} {
set node_($i) [$ns node]
}
set server 0
set router 1
set client 2
#----- Creating Duplex Link-----#
$ns duplex-link $node_($server) $node_($router) 2Mb 50ms DropTail
$ns duplex-link $node_($router) $node_($client) 2Mb 50ms DropTail
$ns duplex-link-op $node_($server) $node_($router) orient right
$ns duplex-link-op $node_($router) $node_($client) orient right
#-----Labelling-----#

```

```
$ns at 0.0 "$node_($server) label Server"
```

```
$ns at 0.0 "$node_($router) label Router"
```

```
$ns at 0.0 "$node_($client) label Client"
```

```
$ns at 0.0 "$node_($server) color blue"
```

```
$ns at 0.0 "$node_($client) color blue"
```

```
$node_($server) shape hexagon
```

```
$node_($client) shape hexagon
```

```
#—————finish procedure—————#
```

```
proc finish { } {
```

```
global ns nf nt
```

```
$ns flush-trace
```

```
close $nf
```

```
close $nt
```

```
puts "running nam..."
```

```
exec nam test1.nam &
```

```
exit 0
```

```
}
```

```
#Calling finish procedure
```

```
$ns at 10.0 "finish"
```

```
$ns run
```

```
#————— Execution —————#
```

```
ns test2.tcl
```

## Result & Discussion

Program is executed successfully and output is obtained

### Viva Questions:

1. NS2 stands for \_\_\_\_\_
2. What is the functionality of DHCP Server
3. SSH Stands for \_\_\_\_\_

