REVA UNIVERSITY
Bengaluru, India

Vision of the University

''REVA University aspires to become an innovative university by developing excellent human resources with leadership qualities, ethical and moral values, research culture and innovative skills through higher education of global standards"

Mission of the University

- To create excellent infrastructure facilities and state-of-the-art laboratories and incubation centres
- To provide student-centric learning environment through innovative pedagogy and education reforms
- To encourage research and entrepreneurship through collaborations and extension activities
- To promote industry-institute partnerships and share knowledge for innovation and development
- To organize society development programs for knowledge enhancement in thrust areas
- To enhance leadership qualities among the youth and enrich personality traits, promote patriotism and moral values.

Vision of the School

The School of Electronics and Communication Engineering is envisioned to be a leading centre of higher learning with academic excellence in the field of electronics and communication engineering blended by research and innovation in tune with changing technological and cultural challenges supported with leadership qualities, ethical and moral values.

Mission of the School

- Establish a unique learning environment to enable the students to face the challenges in the field of Electronics and Communication Engineering and explore multidisciplinary which serve the societal requirements.
- Create state-of-the-art laboratories, resources, and exposure to the current industrial trends to enable students to develop skills for solving complex technological problems of current times and provide a framework for promoting collaborative and multidisciplinary activities.
- Promote the establishment of Centres of Excellence in niche technology areas to nurture the spirit of innovation and creativity among faculty and students.
- Offer ethical and moral value-based education by promoting activities which inculcate the leadership qualities, patriotism and set high benchmarks to serve the society

Program Educational Objectives (PEOs)

The Program Educational Objectives of B. Tech in Electronics and Computer Engineering are as follows:

PEO-1: Have successful professional career in industry, government, and software organization as innovative engineers
PEO-2: Successfully solve engineering problems related to Electronics and Computer Engineering by communicating effectively either as a team or as a team member and lead the team
PEO-3: Pursue higher studies and have an attitude of lifelong learning through cultural, technical and outreach activities
PEO-4: Serve the society regionally, globally and will take up entrepreneurship for the growth of the economy and generate employment

Program Outcomes (POs)

On successful completion of the program, the graduates of B. Tech. (Electronics and Computer Engineering) program will be able to

- **PO-1: Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals for the solution of complex problems in Electronics and computer Engineering.
- **PO-2: Problem analysis:** Identify, formulate, research literature, and analyze engineering problems to arrive at substantiated conclusions using first principles of mathematics, natural, and engineering sciences.
- **PO-3: Design/development of solutions:** Design solutions for complex engineering problems and design system components, processes to meet the specifications with consideration for the public health and safety, and the cultural, societal, and environmental considerations.
- **PO-4: Conduct investigations of complex problems:** Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
- **PO-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.
- **PO-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.
- **PO-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.
- **PO-8: Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice
- **PO-9: Individual and teamwork:** Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings.
- **PO-10: Communication:** Communicate effectively with the engineering community and with society at large. Be able to comprehend and write effective reports documentation. Make effective presentations and give and receive clear instructions.
- **PO-11: Project management and finance:** Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work, as a member and leader in a team. Manage projects in multidisciplinary environments.
- **PO-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.

Programme Specific Outcomes (PSOs)

On successful completion of the program, the graduates of B. Tech. (Electronics and Computer Engineering) program will be able to

- **PSO-1:** Isolate and solve complex problems in the domains of Electronics and Computer Engineering using latest hardware and software tools and technologies, along with analytical and managerial skills to arrive at cost effective and optimum solutions either independently or as a team.

- **PSO-2:** Implant the capacity to apply the concepts of electronics, data analytics, computer networks, cloud computing, artificial intelligence, and machine learning, etc., in the design, development of hardware and software for the application engineering lifecycle systems.

- **PSO-3:** Design, develop and build electronics and software systems to solve real life industry problems using modern tools and techniques.

Contents

SYLLABUS

Course Title	Embedded Systems Lab				Course Type		HC	
Course Code	B20EP0505	Credits	1		Class		V Semester	
Embedded Systems	TLP	Credits	Contact Hours	Work Load	Total Numberof Classes Per Semester		Assessment in Weightage	
	Theory	0	0	0				
	Practice	1	2	2	Theory	Practical	IA	SEE
	-	-	-	-				
	Total	1	2	2	0	28	50%	50%

COURSE OVERVIEW:

Embedded systems have become the next inevitable wave of technology, finding application in diverse fields of engineering. Microprocessors, together with sensors and actuators, have become embeddable in almost everything. The purpose of the course is to provide the students hands-on on Real Time operating Systems (RTOS) widely applied in Embedded Systems.

COURSE OBJECTIVES:

The objectives of this course are:

1. To provide insights into the basics of unix OS based GCC compiler
2. To teach hardware software co-design and firmware design approaches
3. To introduce the concepts POSIX thread libraries
4. Illustrate the different scheduling algorithms and synchronization techniques.

COURSE OUTCOMES(COs)

On successful completion of this course; the student shall be able to:

CO#	Course Outcomes	POs	PSOs
CO1	Implement required thread creations using Unix OS based GCC Compiler	1,2,3,5,9, 10	1,2,3
CO2	Acquire knowledge and understand fundamental embedded system paradigms, characteristics, and attributes.	1,2,3,5,9, 10	1,2,3
CO3	Design and execute a program using POSIX thread libraries	1,2,3,5,9, 10	1,2,3
CO4	Understand about the RTOS based ES concepts and the goal of ES in real time applications.	1,2,3,5,9, 10	1,2,3
CO5	Conduct the experiment for the given design parameters individually (and in a team) within the stipulated time	5,9,10,12	
CO6	Analyze the results, make relevant observations and measurements, and document the results in a form of report/journal.	5,9,10,12	

BLOOM'S LEVEL OF THE COURSE OUTCOMES

CO/ PO	Bloom's Level					
	Remember (L1)	Understand (L2)	Apply (L3)	Analyze (L4)	Evaluate (L5)	Create (L6)
CO1	✓	✓	✓	✓		
CO2	✓	✓	✓	✓		
CO3		✓	✓			
CO4	✓	✓	✓			
CO5	✓	✓	✓	✓		✓
CO6	✓	✓	✓			

COURSE ARTICULATION MATRIX

CO/ POs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	2	1		2				3	3		1	1	2	1
CO2	3	2	1		2				3	3		1	1	2	1
CO3	3	2	1		2				3	3		1	1	2	1
CO4	3	2	1		2				3	3		1	1	2	1
CO5					3				3	2		1			
CO6					3				3	2		1			

Note:1-Low,2-Medium,3-High

COURSE CONTENT

Sl. No.	Name of the Experiment	Tools and Techniques	Expected Skill /Ability
1	Write a program for singly Thread Creation and Termination using POSIX threads.	Unix OS based GCC Compiler	C programming language.
2	Write a program for creating independent threads each of which will execute some random function and use concept of Mutual Exclusion (Task Synchronization). Write a program to create N number of threads and to count how many threads are being executed. Use concept of Mutual Exclusion.	Unix OS based GCC Compiler	C programming language.
3	Write a program to create independent threads each of which will execute some function and wait till threadsare complete before main continues. Unless we wait run the risk of executing an exit which will terminate the process and all threads before the threads have completed.	Unix OS based GCC Compiler	C programming language.

4	Write a program to create the N number of threads andfind the how many threads are executed. Use conceptof Mutual Execution.	Unix OS based GCC Compiler	C programming language.
5	Write a program to create two threads T1 and T2. Thread T1 should count numbers between 1-3 and 8- 10 by calling the function FunctionCount1 and threadT2 should count numbers between 4-7 by calling the function FunctionCount2. Program should print the final count value.	Unix OS based GCC Compiler	C programming language.
6	Design and execute a program using POSIX thread library to create the number of threads specified by the user, each thread independently generates a random integer as an upper limit and then computes and prints the number of primes less than or equal to that upper limit, along with the upper limit.	Unix OS based GCC Compiler	C programming language.
7	Write a program to implement a process with a producer thread and a consumer thread which make use of a bounded buffer (Size can be prefixed atsuitable value) for communication. Use any suitable synchronization construct.	Unix OS based GCC Compiler	C programming language.
8	Write a program to implement the usage of an Anonymous Pipe with size of 512bytes for data sharing between parent and child process using Inheritance Handling mechanism.	Unix OS based GCC Compiler	C programming language.

TEXTBOOKS:

1. K. V. Shibu, "Introduction to embedded systems", TMH education Pvt. Ltd. 2009.

REFERENCE BOOKS

1. Frank Vahid, Tony D. Givargis, Embedded System Design – A Unified Hardware/Software Introduction, John Wiley, 2002.
2. Jonathan W. Valvano, Embedded Microcomputer Systems, 3rd. edition, Cengage Learning, 2011.
3. David E. Simon, An Embedded Software Primer, Pearson Ed., 2005.
4. Raj Kamal, Introduction to Embedded Systems, TMH, 2002.
5. KVKK Prasad, Embedded / Real Time Systems, Dreamtech Press, 2005.
6. Peter M, Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and Internet of Things, Springer, 3rd Edition, 2018

Theory :-

Introduction to Threads & Process:

A process is a program in execution and contains one or more threads.

A thread of execution is often regarded as the smallest unit of processing that a Scheduler works on. A thread is a path of execution within a process.

Multithreading:

A thread is also known as lightweight process. The idea is to achieve parallelism by dividing a process into multiple threads. For example, in a browser, multiple tabs can be different threads. MS Word uses multiple threads: one thread to format the text, another thread to process inputs, etc.

Process vs Thread:

The primary difference is that threads within the same process run in a shared memory space, while processes run in separate memory spaces. Threads are not independent of one another like processes are, and as a result threads share with other threads their code section, data section, and OS resources (like open files and signals). But, like process, a thread has its own program counter (PC), register set, and stack space.

Advantages of Thread over Process:

1. Responsiveness: If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.

2. Faster context switch: Context switch time between threads is lower compared to process context switch. Process context switching requires more overhead from the CPU.

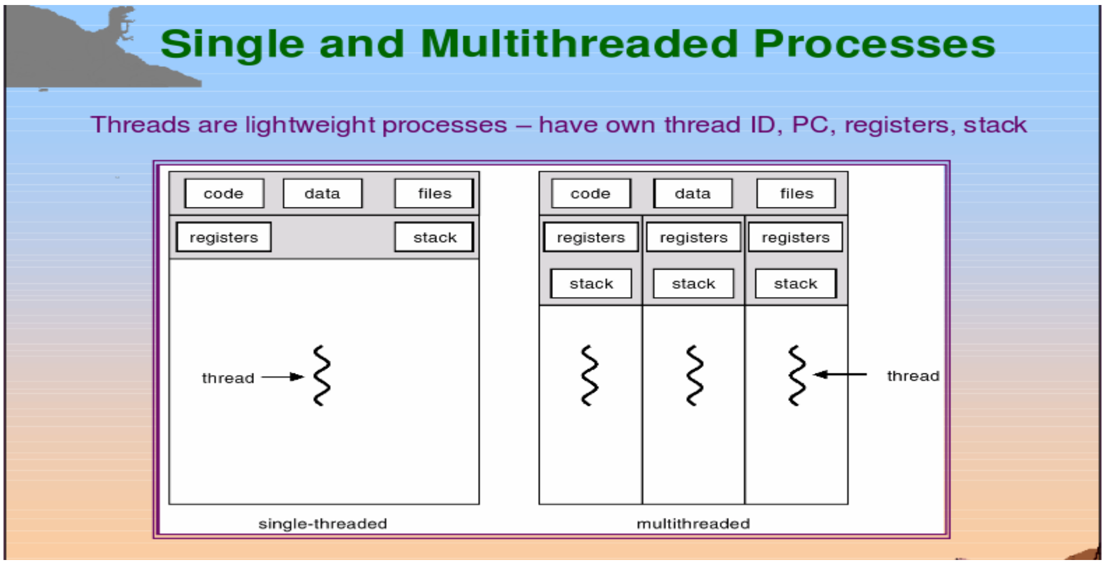
3. Effective utilization of multiprocessor system: If we have multiple threads in a single process, then we can schedule multiple threads on multiple processor. This will make process execution faster.

4. Resource sharing: Resources like code, data, and files can be shared among all threads within a process.

Note: stack and registers can't be shared among the threads. Each thread has its own stack and registers.

5. Communication: Communication between multiple threads is easier, as the threads shares common address space. while in process we have to follow some specific communication technique for communication between two process.

6. Enhanced throughput of the system: If a process is divided into multiple threads, and each thread function is considered as one job, then the number of jobs completed per unit of time is increased, thus increasing the throughput of the system.



Types of Threads

There are two types of threads.

User Level Thread

Kernel Level Thread

User Level Thread:

Threads management is done by user level threads library.

Three primary IEEE Standard thread libraries are present:

1) POSIX Threads (pthreads).

2) Win32 Threads

3) Java Threads.

Three primary IEEE Standard thread libraries are present:

1) POSIX Threads (pthreads).

2) Win32 Threads

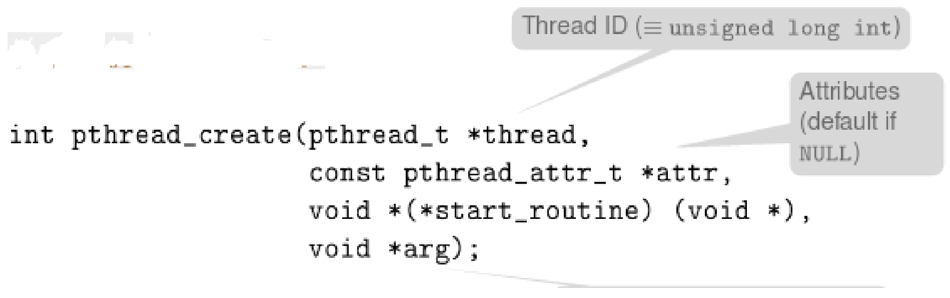
3) Java Threads.

POSIX Threads (pthreads): API for thread creation, synchronization and termination.

Thread Creation and Termination:

#include <pthread.h> // Library for thread creation, synchronization and termination

pthread_ t th_id; // Thread Identification



```
int pthread_create(pthread_t *thread,
                   const pthread_attr_t *attr,
                   void *(*start_routine) (void *),
                   void *arg);
```

// Thread Creation

pthread_join (th_id, status); // Thread Management – Wait for termination of other thread and join // current thread for execution

General Procedure:

The operating system used is RED HAT ENTERPRISE LINUX (RHEL).

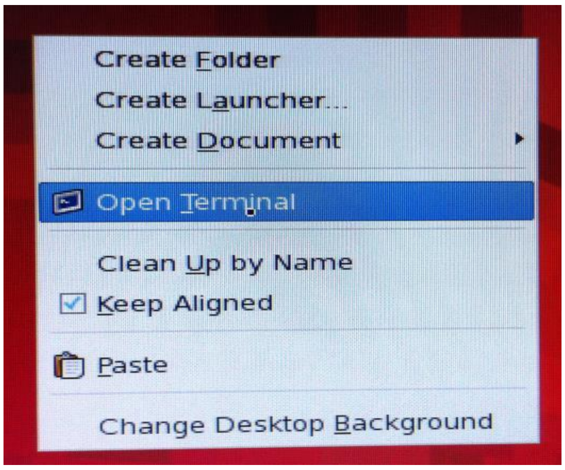
User name: **root**

Password: **root123**

(The Username and Password is common for all RHEL Systems.)

1.Right click on the desktop select open terminal.

2.Click on Open Terminal



A window named as **[root@cad ~]#** will open.

3.To create your own directory type **mkdir <directoryname>**<press enter button>

4.Change (get into) the directory type **cd <directoryname>**<press enter button>

5.Then type **pwd** – present working directory – to check the path of the directory located. To type the C program here we are using editor called **gedit**

6.Type **gedit <filename.c>**(file name followed with the extension **.c(dot c)**<press enter button .The editor window will open there type the C program, save and exit.

7.To compile the program type **cc <filename.c>**<press enter button>

8.If you are using thread in the program then to compile type

<cc–lpthread <filename.c>

<press enter button>

9.If any error exits it will show the error. To make corrections again type

gedit <filename.c>

<press enter button>

10.Correct the errors save and exit. If there are no errors then you can continue to check output.

11.To check output **type ./a.out** the output is visible in the same window.

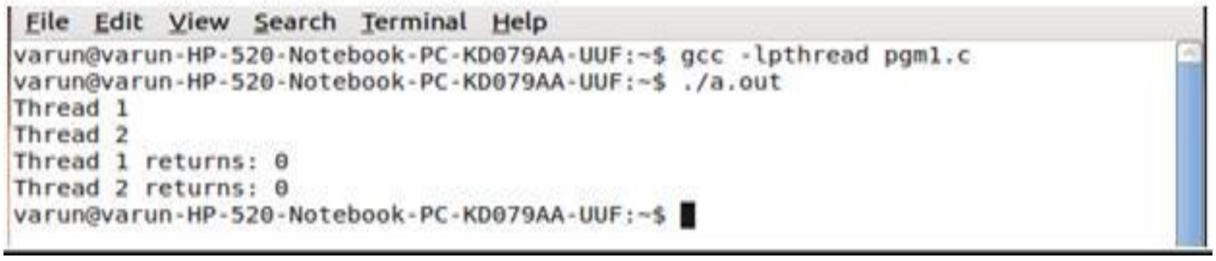
<press enter button>

Program 1

Aim: Write a program in C for single Thread Creation and Termination using POSIX thread library.

```
#include <stdio.h>
#include <pthread.h>
void *print_message_function( void *ptr );
int main()
{
        pthread_t thread1, thread2;
        char *message1 = "Thread 1";
        char *message2 = "Thread 2";
        int iret1, iret2;
/* Create independent threads each of which will execute function */
        iret1 = pthread_create( &thread1, NULL, print_message_function, message1);
        iret2= pthread_create( &thread2, NULL,
        print_message_function, message2);
        pthread_join( thread1, NULL);
        pthread_join( thread2, NULL);
        printf("Thread 1 returns: %d\n",iret1);
        printf("Thread 2 returns: %d\n",iret2);
        return 0;
}
void *print_message_function( void *ptr )
{
        char *message;
        message = (char *) ptr;
        printf("%s \n", message);
}
```

Output:-



```
varun@varun-HP-520-Notebook-PC-KD079AA-UUF:~$ gcc -lpthread pgm1.c
varun@varun-HP-520-Notebook-PC-KD079AA-UUF:~$ ./a.out
Thread 1
Thread 2
Thread 1 returns: 0
Thread 2 returns: 0
varun@varun-HP-520-Notebook-PC-KD079AA-UUF:~$
```

Program 2

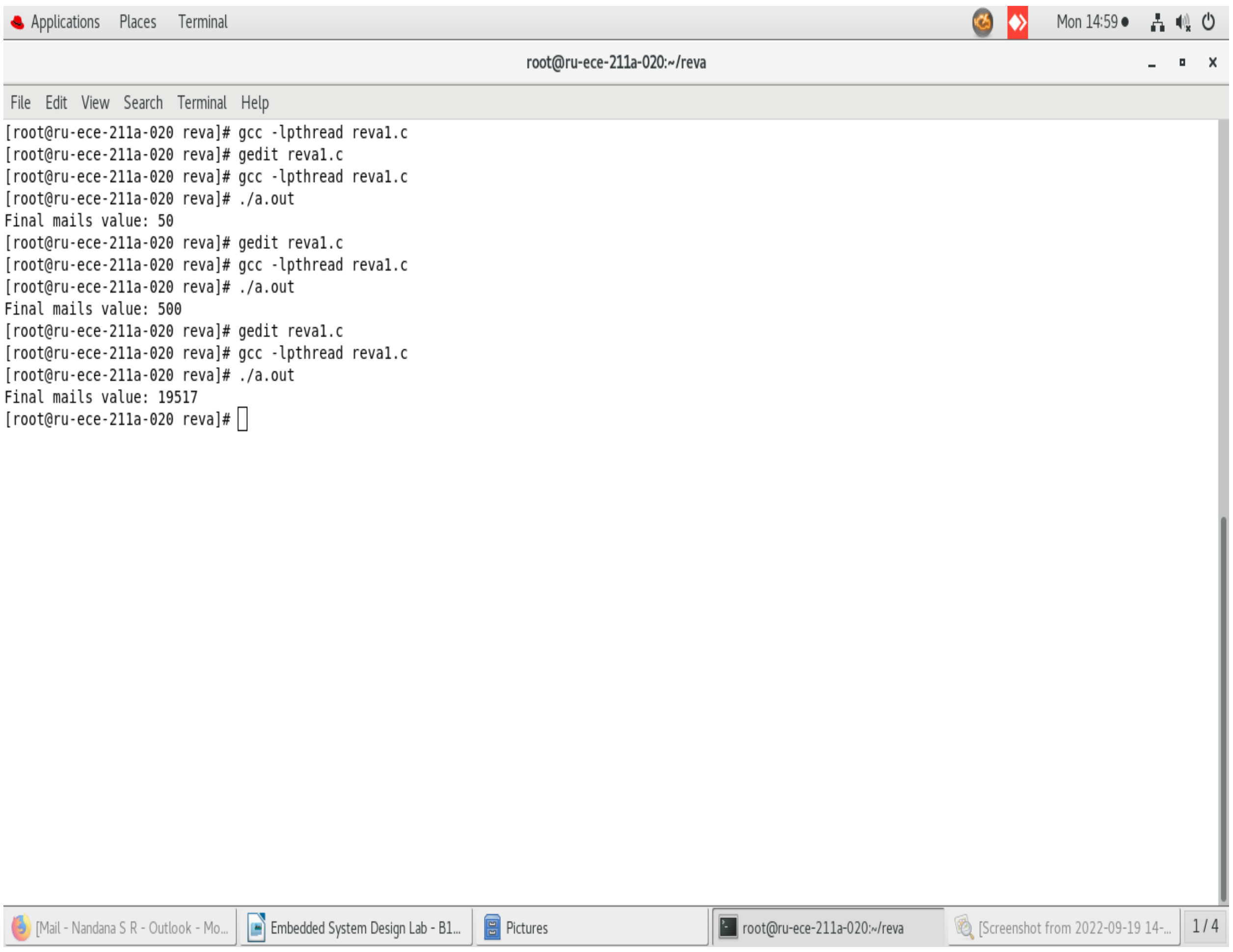
Write a program in C using POSIX thread library to create N number of threads USING shared variable without MUTEX (RACING).

```
#include <stdio.h>
#include <pthread.h>
#define NTHREADS 5
int mails=0;
void *task()
{
        int k;
        for(k=0; k<10000; k++)
                z`mails++;
}
int main()
{
        pthread_t thread_id[NTHREADS];
        int i, j;
        for(i=0; i < NTHREADS; i++)
        {
                pthread_create( &thread_id[i], NULL, &task, NULL );
        }
        for(j=0; j < NTHREADS; j++)
        {
                pthread_join( thread_id[j], NULL);
        }
printf("Final mails value: %d\n", mails);
return 0;
}
```

Note: change the value of k=100, k=1000, k=1000, k=10000 note down the value of mails.

k value	Mails count
10	
100	
1000	
10000	
100000	

Output:-


Applications Places Terminal
Mon 14:59
root@ru-ece-211a-020:~/reva
File Edit View Search Terminal Help
[Mail - Nandana S R - Outlook - Mo...
Embedded System Design Lab - B1...
Pictures
root@ru-ece-211a-020:~/reva
[Screenshot from 2022-09-19 14-...
1 / 4

```
[root@ru-ece-211a-020 reva]# gcc -lpthread reva1.c
[root@ru-ece-211a-020 reva]# gedit reva1.c
[root@ru-ece-211a-020 reva]# gcc -lpthread reva1.c
[root@ru-ece-211a-020 reva]# ./a.out
Final mails value: 50
[root@ru-ece-211a-020 reva]# gedit reva1.c
[root@ru-ece-211a-020 reva]# gcc -lpthread reva1.c
[root@ru-ece-211a-020 reva]# ./a.out
Final mails value: 500
[root@ru-ece-211a-020 reva]# gedit reva1.c
[root@ru-ece-211a-020 reva]# gcc -lpthread reva1.c
[root@ru-ece-211a-020 reva]# ./a.out
Final mails value: 19517
[root@ru-ece-211a-020 reva]# 
```

Program 3

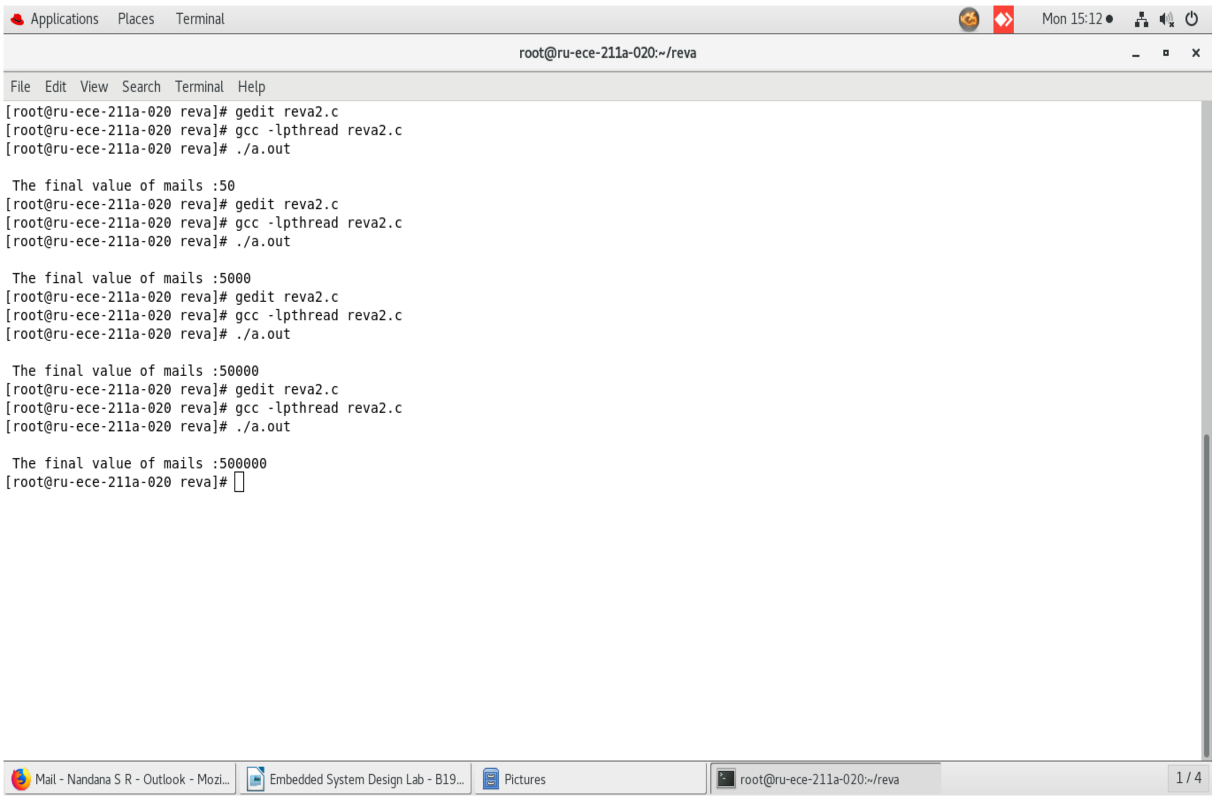
Aim : Write a program to create the N number of threads and find the how many threads areexecuted. Use concept of Mutual Execution.

```
#include<stdio.h>
#include<pthread.h>
#define N 5
pthread_mutex_t m1 = PTHREAD_MUTEX_INITIALIZER;
int mails =0;
void *task()
{
int k;
for(k=0; k<10;k++)
{
        pthread_mutex_lock(&m1);
        mails++;
        pthread_mutex_unlock(&m1);
}
}
int main()
{
        pthread_t th[N];
int i,j;
for(i=0;i<N;i++)
{
        pthread_create(&th[i],NULL, &task,NULL);
}
for(j=0;j<N;j++)
{
        pthread_join(th[j], NULL);
}
printf("\n The final value of mails :%d\n", mails);
}
```

Note: change the value of k=100, k=1000, k=1000, k=10000 note down the value of mails.

k value	Mails count
10	
100	
1000	
10000	
100000	

Output:-


Applications
Places
Terminal
Mon 15:12 ●
root@ru-ece-211a-020:~/reva
File
Edit
View
Search
Terminal
Help
Mail - Nandana S R - Outlook - Mozi...
Embedded System Design Lab - B19...
Pictures
root@ru-ece-211a-020:~/reva
1 / 4

```
[root@ru-ece-211a-020 reva]# gedit reva2.c
[root@ru-ece-211a-020 reva]# gcc -lpthread reva2.c
[root@ru-ece-211a-020 reva]# ./a.out

 The final value of mails :50
[root@ru-ece-211a-020 reva]# gedit reva2.c
[root@ru-ece-211a-020 reva]# gcc -lpthread reva2.c
[root@ru-ece-211a-020 reva]# ./a.out

 The final value of mails :5000
[root@ru-ece-211a-020 reva]# gedit reva2.c
[root@ru-ece-211a-020 reva]# gcc -lpthread reva2.c
[root@ru-ece-211a-020 reva]# ./a.out

 The final value of mails :50000
[root@ru-ece-211a-020 reva]# gedit reva2.c
[root@ru-ece-211a-020 reva]# gcc -lpthread reva2.c
[root@ru-ece-211a-020 reva]# ./a.out

 The final value of mails :500000
[root@ru-ece-211a-020 reva]# 
```

Program 4

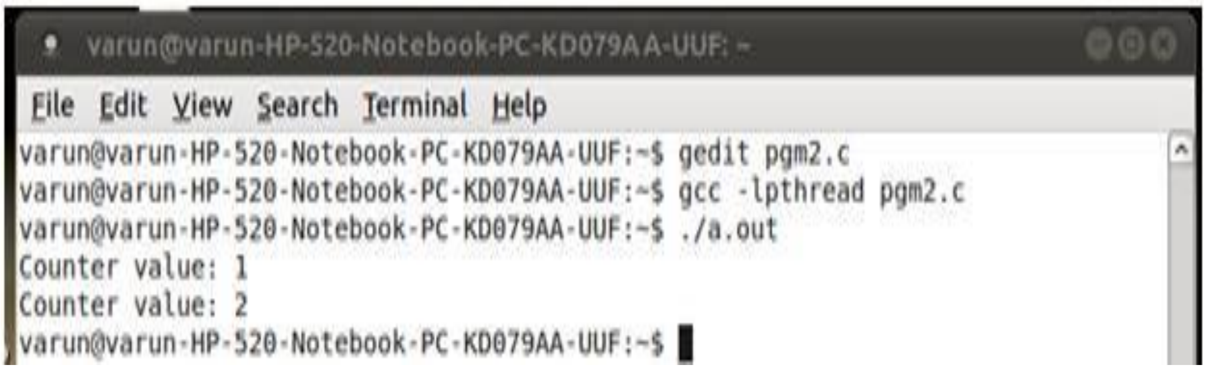
Aim: Write a program in C using POSIX thread library for creating independent threads each of which will execute some random function and use concept of Mutual Exclusion (Task Synchronization).

Theory:

Mutual Exclusion (Mutex) is one of the Inter process communication mechanisms, which is used to provide tasks with synchronized access to shared resources.In the case of mutex, only the thread that locked or acquired the mutex can unlock it. As it is created and initialised as a global variable it need not be passed as a argument to the function.

```
#include <stdio.h>
#include <pthread.h>
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
int counter = 0;
void *ThreadFn()
{
pthread_mutex_lock( &mutex1 );
counter++;
printf("Counter value: %d\n",counter);
pthread_mutex_unlock( &mutex1 );
}
int main()
{
pthread_t thread1, thread2;
pthread_create( &thread1, NULL, &ThreadFn, NULL);
pthread_create( &thread2, NULL, &ThreadFn, NULL);
 pthread_join( thread1, NULL);
pthread_join( thread2, NULL);
return 0;
}
```

Output:-


varun@varun-HP-520-Notebook-PC-KD079AA-UUF: ~
File Edit View Search Terminal Help

```
varun@varun-HP-520-Notebook-PC-KD079AA-UUF:~$ gedit pgm2.c
varun@varun-HP-520-Notebook-PC-KD079AA-UUF:~$ gcc -lpthread pgm2.c
varun@varun-HP-520-Notebook-PC-KD079AA-UUF:~$ ./a.out
Counter value: 1
Counter value: 2
varun@varun-HP-520-Notebook-PC-KD079AA-UUF:~$
```

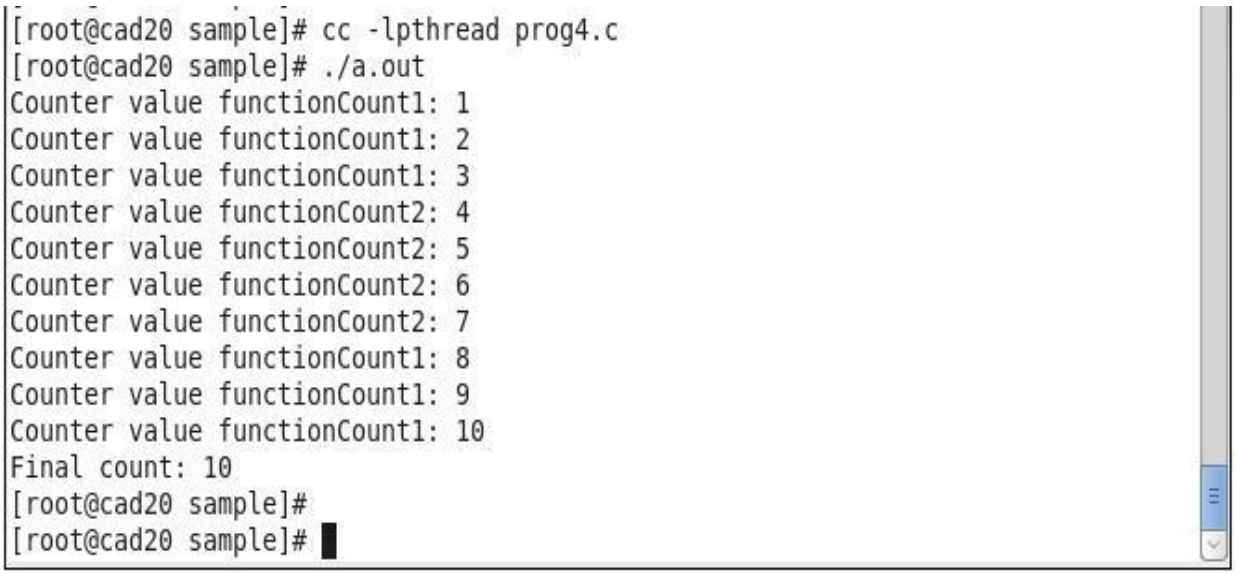
Program 5

Aim : Write a program in C using POSIX thread library to create two threads T1 and T2. Thread T1 should count numbers between 1-3 and 8-10 and Thread T2 should count numbers between 4-7. Threads should print final count value.

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

pthread_mutex_t count_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t condition_var= PTHREAD_COND_INITIALIZER;

void *functionCount1();
void *functionCount2();
int count = 0;

#define COUNT_DONE 10
#define COUNT_HALT1 3
#define COUNT_HALT2 6

int main()
{
        pthread_t thread1, thread2;
        pthread_create( &thread1, NULL, &functionCount1, NULL);
        pthread_create( &thread2, NULL, &functionCount2, NULL);
        pthread_join( thread1, NULL);
        pthread_join( thread2, NULL);
        printf("Final count: %d\n",count);
return 0;
}

//Write numbers 1-3 and 8-10 as permitted by functionCount2()
void *functionCount1()
{
```

```
while (1)
{
//Lock mutex and then wait for signal to relase mutex
        pthread_mutex_lock( &count_mutex );
//Wait while functionCount2() operates on count
//mutex unlocked if conditionn varialbe in functionCount2() signaled.
        pthread_cond_wait( &condition_var, &count_mutex );
        count++;
        printf("Counter value functionCount1: %d\n",count);
        pthread_mutex_unlock( &count_mutex );
if(count >= COUNT_DONE)
        return(NULL);
}
return 0;
}
//Write numbers 4-7
        void *functionCount2()
        {
                while(1)
                {
                        pthread_mutex_lock( &count_mutex );
                if( count < COUNT_HALT1 || count > COUNT_HALT2 )

                {
//Condition of if statement has been met.
//Signal to free waiting thread by freeing the mutex.
//Note: functionCount1() is now permitted to modify "count".
                pthread_cond_signal( &condition_var );
                }
        else
                {
                count++;
                printf("Counter value functionCount2: %d\n",count);
                }
        pthread_mutex_unlock( &count_mutex );
        if(count >= COUNT_DONE)
                return(NULL);
        }
}
```

<u>**Output:-**</u>



```
[root@cad20 sample]# cc -lpthread prog4.c
[root@cad20 sample]# ./a.out
Counter value functionCount1: 1
Counter value functionCount1: 2
Counter value functionCount1: 3
Counter value functionCount2: 4
Counter value functionCount2: 5
Counter value functionCount2: 6
Counter value functionCount2: 7
Counter value functionCount1: 8
Counter value functionCount1: 9
Counter value functionCount1: 10
Final count: 10
[root@cad20 sample]#
[root@cad20 sample]#
```

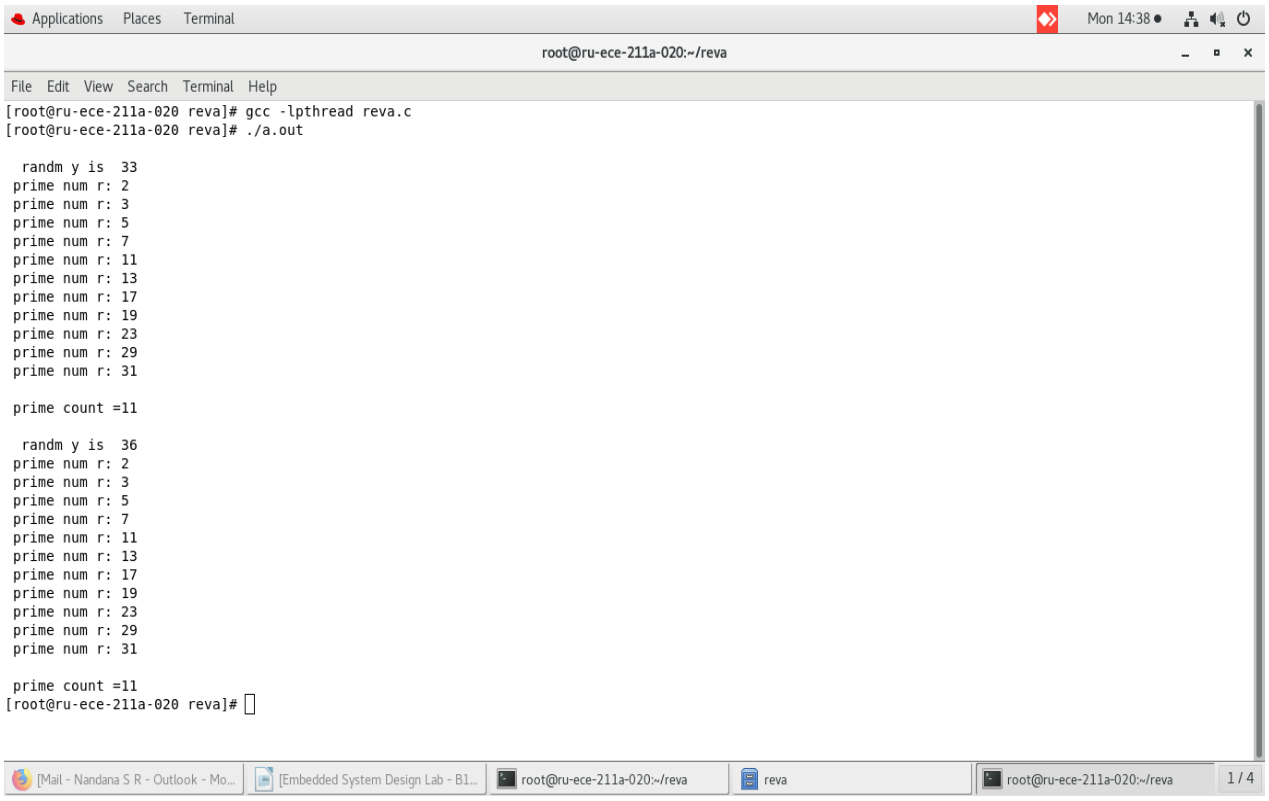
Program 6

Aim: Design and execute a program in C using POSIX thread library to create the number of threads specified by the user. Each thread independently generates a random integer as an upper limit and then computes and prints the number of primes less than or equal to that upper limit, along with the upper limit.

```
#include<stdio.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
#include<stdbool.h>
#include<time.h>
#define  N  2
int  y;
int flag=0;
pthread_mutex_t  m1= PTHREAD_MUTEX_INITIALIZER;
void *check_prime_num()
{
             int  i,j;
            int count =0;
            pthread_mutex_lock(&m1);
            y=(rand()%50);
            printf("\n  Random y is  %d \n", y);
            for ( i=0;i<=y; i++)
            {
            flag=0;
                        for ( j=2;j<=i/2; j++)
                        {
                        if (i%j ==0)
                                    {
                                                flag=1;
                                                break;
                                    }
```

```
                }
                if(flag==0 && i>=2)
                {
                        count++;
                        printf(" prime num r: %d \n", i);
                }
                }
        printf(" \n prime count =%d \n", count);
        pthread_mutex_unlock(&m1);
}
int main()
{
        pthread_t th[N];
        int i,j;
        for ( i=0;i<N; i++)
        {
                pthread_create(&th[i],NULL, &check_prime_num,NULL);
        }
        for ( j=0;j<N; j++)
        {
                if(pthread_join(th[j],NULL)!=0)
                {
                        perror("failed to join thred \n ");
                }
        }
}
```

Output:-


Applications Places Terminal
Mon 14:38 ●
root@ru-ece-211a-020:~/reva
File Edit View Search Terminal Help
[Mail - Nandana S R - Outlook - Mo...
[Embedded System Design Lab - B1...
root@ru-ece-211a-020:~/reva
reva
root@ru-ece-211a-020:~/reva
1 / 4

```
[root@ru-ece-211a-020 reva]# gcc -lpthread reva.c
[root@ru-ece-211a-020 reva]# ./a.out

  randm y is  33
 prime num r: 2
 prime num r: 3
 prime num r: 5
 prime num r: 7
 prime num r: 11
 prime num r: 13
 prime num r: 17
 prime num r: 19
 prime num r: 23
 prime num r: 29
 prime num r: 31

 prime count =11

  randm y is  36
 prime num r: 2
 prime num r: 3
 prime num r: 5
 prime num r: 7
 prime num r: 11
 prime num r: 13
 prime num r: 17
 prime num r: 19
 prime num r: 23
 prime num r: 29
 prime num r: 31

 prime count =11
[root@ru-ece-211a-020 reva]# 
```

Program 7

Aim: Write a program in C using POSIX thread library to implement a process with a producer thread and a consumer thread which makes use of a bounded buffer (Fixed Size of 512kb) for communication. Use appropriate synchronization techniques.

Theory:

Producer Consumer Problem:

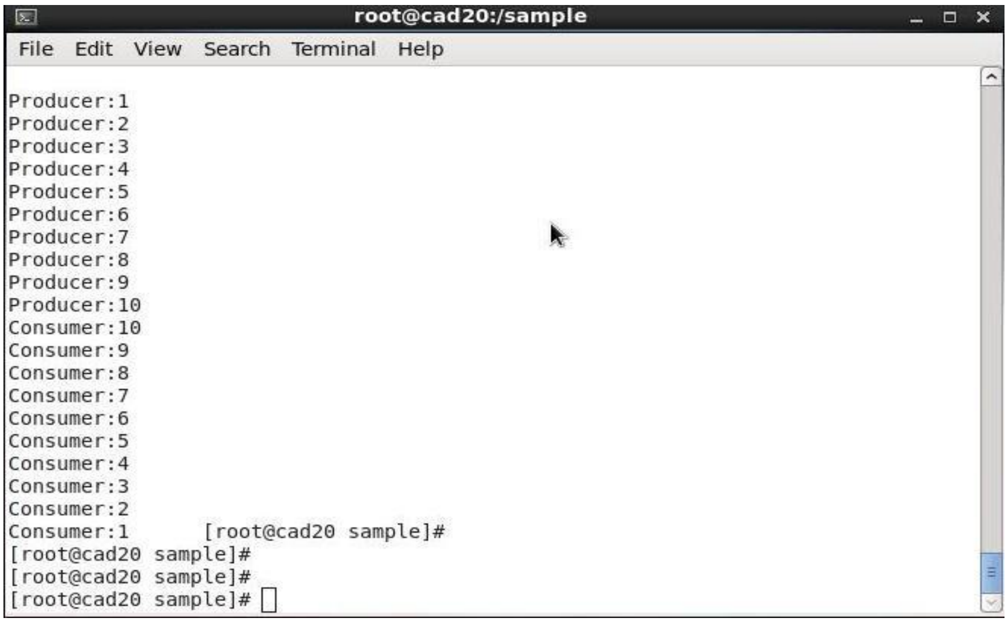
The producer–consumer problem (also known as the bounded-buffer problem) is a classic example of a multi-processsynchronization problem. The problem describes two processes, the producer and the consumer, who share a common, fixed-size buffer used as a queue. The producer's job is to generate data, put it into the buffer, and start again. At the same time, the consumer is consuming the data (i.e., removing it from the buffer), one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.

The solution for the producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer. The solution can be reached by means of inter-process communication, typically using Mutex.

```
#include <stdio.h>
#include <pthread.h>
#define Buffer_Limit 10
int Index_Value = 0,i, j;
int Buffer[Buffer_Limit];
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t Buffer_Full = PTHREAD_COND_INITIALIZER;
pthread_cond_t Buffer_Empty = PTHREAD_COND_INITIALIZER;
void *Consumer()
{
for(j=0;j<Buffer_Limit;j++)
{
```

```
pthread_mutex_lock(&mutex);
if(Index_Value == -1)
{
pthread_cond_wait(&Buffer_Empty, &mutex);
}
printf("\nConsumer:%d\t", Index_Value--);
pthread_mutex_unlock(&mutex);
pthread_cond_signal(&Buffer_Full);
}
}
void *Producer()
{
for(i=0;i<Buffer_Limit;i++)
{
pthread_mutex_lock(&mutex);
if(Index_Value == Buffer_Limit)
{
pthread_cond_wait(&Buffer_Full, &mutex);
}
Buffer[Index_Value++] = rand()%50;
printf("\nProducer:%d\t", Index_Value);
pthread_mutex_unlock(&mutex);
pthread_cond_signal(&Buffer_Empty);
}
}
int main()
{     pthread_t producer_thread_id, consumer_thread_id;
pthread_create(&producer_thread_id, NULL, &Producer, NULL);
pthread_create(&consumer_thread_id, NULL, &Consumer, NULL);
pthread_join(producer_thread_id, NULL);
pthread_join(consumer_thread_id, NULL);
return 0;
}
```

Output:-



```
Producer:1
Producer:2
Producer:3
Producer:4
Producer:5
Producer:6
Producer:7
Producer:8
Producer:9
Producer:10
Consumer:10
Consumer:9
Consumer:8
Consumer:7
Consumer:6
Consumer:5
Consumer:4
Consumer:3
Consumer:2
Consumer:1      [root@cad20 sample]#
[root@cad20 sample]#
[root@cad20 sample]#
[root@cad20 sample]#
```

Program 8

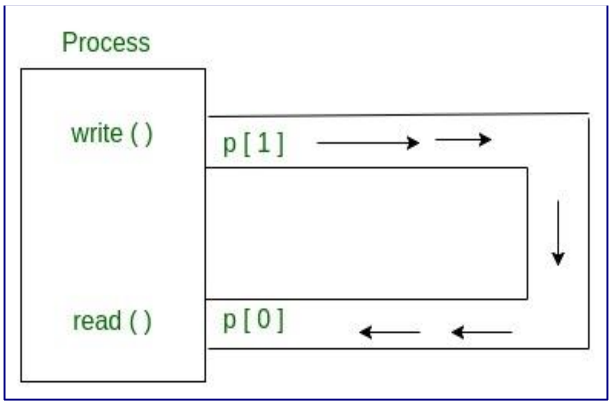
Aim: Write a program in C using POSIX thread library to Implement the usage of anonymous pipe with 512 bytes for data sharing between parent and child process using inheritance handling mechanism.

Theory:

Pipe:

Conceptually, a pipe is a connection between two processes, such that the standard output from one process becomes the standard input of the other process. In UNIX Operating System, Pipes are useful for communication between related processes(inter-process communication).

- Pipe is one-way communication only i.e we can use a pipe such that One process write to the pipe, and the other process reads from the pipe. It opens a pipe, which is an area of main memory that is treated as a *"virtual file"*.
- The pipe can be used by the creating process, as well as all its child processes, for reading and writing. One process can write to this "virtual file" or pipe and another related process can read from it.
- If a process tries to read before something is written to the pipe, the process is suspended until something is written.
- The pipe system call finds the first two available positions in the process's open file table and allocates them for the read and write ends of the pipe.



int pipe(int fds[2]);

Parameters :

fd[0] will be the fd(file descriptor) for the read end of pipe.

fd[1] will be the fd (file descriptor) for the write end of pipe.

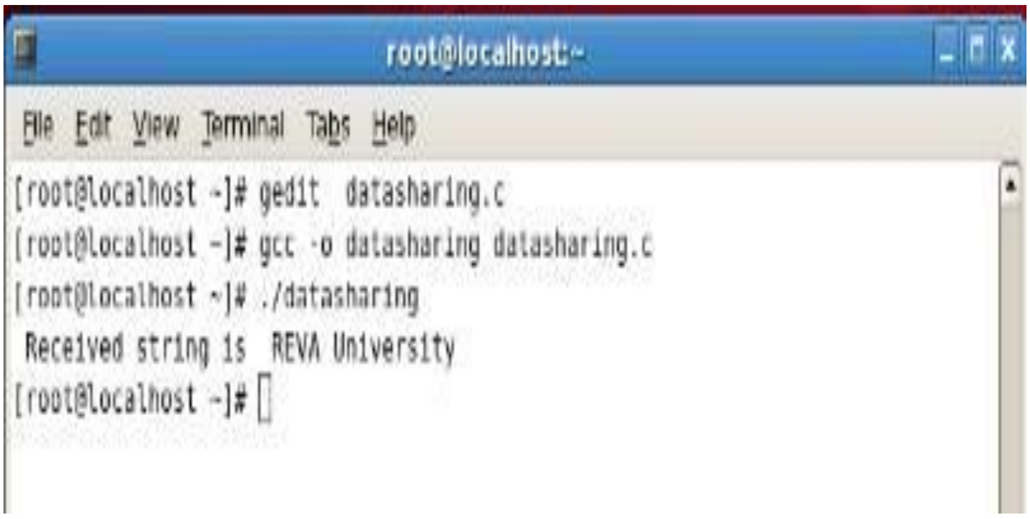
Returns : 0 on Success. -1 on error.

Pipes behave FIFO(First in First out), Pipe behave like a queue data structure. Size of read and write don’t have to match here. We can write 512 bytes at a time but we can read only 1 byte at a time in a pipe.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/types.h>
int main (void)
{
int fd[2];
pid_t childpid;
char string[]= " REVA University \n";
char readbuffer[80];
pipe(fd);
        if((childpid=fork())==-1)
        {
        perror("fork");
        return 0;
        }
        if (childpid==0)
        {
        close(fd[0]);

        write (fd[1], string, strlen(string));
        return 0;

        }
else
        {
```

```
        close(fd[1]);
        read (fd[0], readbuffer, sizeof(readbuffer));
        printf(" Received string is %s", readbuffer); }
        return(0);
        }
```

Output:- Anonymous pipe with 512 bytes for data sharing was implemented between parentand child thread.



```
[root@localhost ~]# gedit  datasharing.c
[root@localhost ~]# gcc -o datasharing datasharing.c
[root@localhost ~]# ./datasharing
 Received string is  REVA University
[root@localhost ~]# 
```

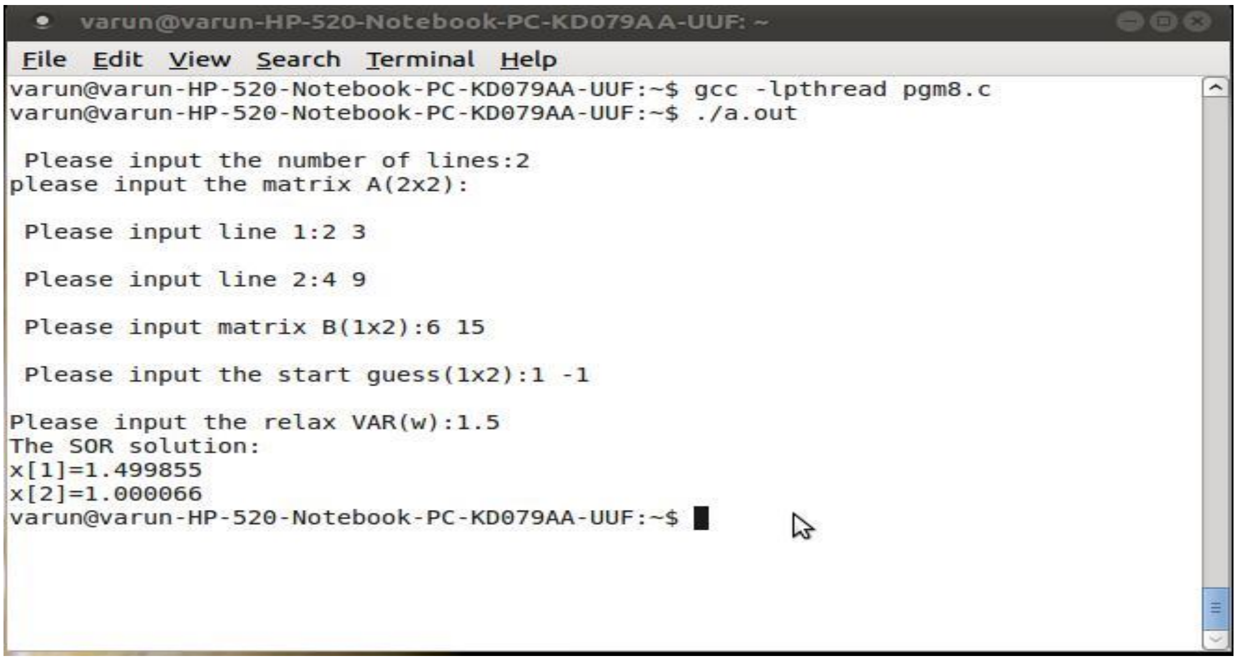
Challenge Experiment:

Aim: Write and execute a C program to solve a system of n linear equations using successive over-relaxation method and n processes which use shared memory API

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#define A(x,y) A[(u+1)*(x)+y]
#define PRECISION 0.0001
int u;
int is_converged(int n, float *A, float *B, float *p)
{
        float *sum;
        int i, j;
        sum=(float *)malloc(n*sizeof(float));
        for(i=0;i<n;++i)
        sum[i]=0;
        for(i=0;i<n;++i)
        {
                for(j=0;j<n;++j)
                sum[i]+=A(i,j)*p[j];
                sum[i]=fabs(B[i]-sum[i]);
        }
        for(i=0;i<n;++i)
        {
                if(sum[i]>PRECISION)
                return 0;
        }
return 1;
```

```
}
void SOR(int n, float *A, float *B, float *p, float w)
{
        float sum;
        int i, j;
        for(i=0;i<n;++i)
        {
                sum=0;
        for(j=0;j<i;++j)
        {
                sum+=A(i,j)*p[j];
        }
        for(j=i+1;j<n;++j)
        {
                sum+=A(i,j)*p[j];
        }
        p[i]+=w*(((B[i]-sum)/A(i,i))-p[i]);
}
}
int main(int argc, char* argv[])
{
        float *A, *p, *B;
        float w;
        int shmid;
        key_t key;
        int i,j;
        printf("\n Please input the number of lines:");
        scanf("%d", &u);
        A=(float *)calloc(u*(u+1),sizeof(float));
        B=(float *)malloc(u*sizeof(float));
        printf("please input the matrix A(%dx%d):\n",u,u);
        for(i=1;i<=u;++i)
        {
printf("\n Please input line %d:",i);
for(j=1;j<=u;++j)
scanf("%f", &A(i-1,j-1));
        }
printf("\n Please input matrix B(1x%d):",u);
for(i=0;i<u;++i)
scanf("%f", &B[i]);
key=5678;

if((shmid=shmget(key,sizeof(float)*u,IPC_CREAT|0666))<0)
{
        perror("shmget");
        exit(1);
}

if((p=(float*)shmat(shmid,NULL,0))==(float *)-1)
{
        perror("shmat");
        exit(1);
}
printf("\n Please input the start guess(1x%d):",u);for(i=0;i<u;++i)
scanf("%f",&p[i]);
printf("\nPlease input the relax VAR(w):");
scanf("%f",&w);
while(!is_converged(u,A,B,p))
{
        if(fork()==0)
        {
                if((shmid=shmget(key,sizeof(float) *u,0666))<0)
                {
                        perror("shmget");
                        exit(1);
                }
        if((p=(float*)shmat(shmid,NULL,0))==(float*)-1)
        {
                perror("shmat");
                exit(1);
        }
        SOR(u,A,B,p,w);
        exit(0);
        }
        wait();
}
printf("The SOR solution:\n");
for(i=0;i<u;++i)
printf("x[%d]=%f\n",i+1,p[i]);
return 0;
}
```

Output:-


varun@varun-HP-520-Notebook-PC-KD079AA-UUF: ~
File Edit View Search Terminal Help

```
varun@varun-HP-520-Notebook-PC-KD079AA-UUF:~$ gcc -lpthread pgm8.c
varun@varun-HP-520-Notebook-PC-KD079AA-UUF:~$ ./a.out

 Please input the number of lines:2
please input the matrix A(2x2):

 Please input line 1:2 3

 Please input line 2:4 9

 Please input matrix B(1x2):6 15

 Please input the start guess(1x2):1 -1

Please input the relax VAR(w):1.5
The SOR solution:
x[1]=1.499855
x[2]=1.000066
varun@varun-HP-520-Notebook-PC-KD079AA-UUF:~$
```

Viva Questions

1.What is watchdog timer?

A watchdog timer (or computer operating properly timer) is a computer hardware timing device that triggers a system reset if the main program, due to some fault condition, such as a hang, neglects to regularly service the watchdog. The intention is to bring the system back from the hung state into normal operation.

2.What is semaphore?

In computer science, a semaphore is a protected variable or abstract data type which constitutes the classic method for restricting access to shared resources such as shared memory in a parallel programming environment. A counting semaphore is a counter for a set of available resources, rather than a locked/unlocked flag of a single resource.

3.What is mutex?

Mutual exclusion (often abbreviated to mutex) algorithms are used in concurrent programming to avoid the simultaneous use of a common resource, such as a global variable, by pieces of computer code called critical sections.

4.Advantages and disadvantages of using macro and inline functions?

Advantage:Macros and Inline functions are efficient than calling a normal function. The times spend in calling the function is saved in case of macros and inline functions as these are included directly into the code.

Disadvantage:Macros and inline functions increased the size of executable code.Difference in inline functions and macro

- Macro is expanded by preprocessor and inline function are expanded by compiler.
- Expressions passed as arguments to inline functions are evaluated only once while expression passed as argument to inline functions are evaluated more than once.
- More over inline functions are used to overcome the overhead of function calls.
- Macros are used to maintain the readability and easy maintenance of the code.

5. What is the difference between a ‘thread’ and a ‘process’?

In computing, a process is an instance of a computer program that is being sequentially executed by a computer system that has the ability to run several computer programs concurrently.

Thread:A single process may contain several executable programs (threads) thatwork together as a coherent whole. One thread might, for example, handle error signals, another might send a message about the error to the user, while a third thread is executing the actual task.

6.What are hard and soft Real time systems?

The hard real time systems are the once that depend on the output very strictly on time. The soft real time systems on the other are not very rigid as the hard real time systems. The performance of the system degrades with the lateness of response, but it is bearable and can be optimized to a certain level for reuse of the result.

7.What do you mean by interrupt latency?

Interrupt latency refers to the time taken for the system to start the handler for the specific interrupt. The time from the time of arrival of interrupt to the time it is being handled.

8.What is ISR? Can they be passed any parameter and can they return a value?

ISR refers to the Interrupt Service Routines. These are procedures stored at specific memory addresses which are called when certain type of interrupt occurs. The ISRs cannot return a value and they cannot be passed any parameters.

9.Which of the following can be used to refer to entities within the RTOS? a)threads

b)kernels

c)system

d) applications

Explanation:The threads and processes can be used to refer to entities within theRTOS. They provide an interchangeable replacement for the task. They have a slight difference in their function. A process is a program in execution and it has its own address space whereas threads have a shared address space. The task can be defined as a set of instructions which can be loaded into the memory.

10.Which of the following possesses threads execution?

a) Process

b) thread

c) kernel

d) operating system

Explanation*:* The process has threads of execution which are the paths through thecode.

11.Which of the following is inherited from the parent task?

a) task

b) process

c) hread

d) kernel

Explanation*:* The threads are a part of the process, that is, it uses a shared memoryof the process and therefore said that its resources are inherited from the parent process or task.