Lab 1

1. Create the following directory structure under your home directory:

```
Documents
        development
        cs101
        lab1b
        test1
 essays
        engl101
```

Ans.

```
mkdir ~/Documents

cd ~/Documents

mkdir development cs101 lab1b test1

cd ../

mkdir essays

mkdir ~/Documents/essays/eng101
```

2. Delete the Documents directory from your home directory. Show that the Documents directory is deleted.

Ans:- rm -rf ~/Documents

3. Bash Script to calculate factorial of a number.

Ans.

```
#!      /bin/bash
#       Takes one input
#       Calculate it's factorial

echo Enter a number
read n
p=1
for((i=1;i<=n;i=$(($i+1))))
do
        p=$(($p*$i))
done
echo Factorial is $p
```

4. Open firefox. Use ps to find process ID of firefox and kill it.

Ans.

```
ID_FIREFOX=$(ps -aux | grep firefox | grep -o root[‘ ’]*[0-9]* | grep -wo [‘ ’]*[0-9]*)
kill -9 ID_FIREFOX
```

5. Using grep, find occurrence of cheerful in file random.txt

Ans.

```
grep -w cheerful ~/Downloads/random.txt
```

6. Count number of occurrences of ‘the’ in file random.txt

Ans.

```
grep -wo the ~/Downloads/random.txt  | wc -w
```

7. Implement bubble sort in bash

Ans.

```
#!	/bin/bash
#	this code takes array as input
#	and sort it using bubble sort technique

declare -a arr;
for ((i=0;;i=$(($i+1))))
do
	read p
	if [ $p -eq -1 ]
	then
		echo Input Taken
		break
	else
		arr[i]=$p
	fi
done
echo Size of array is ${#arr[@]}
size=${#arr[@]}
for((i=0;i<$size;i=$(($i+1))))
do
	for((j=0;j<$(($size-1));j=$(($j+1))))
	do
		if [ ${arr[$j]} -gt ${arr[$(($j+1))]} ]; then
			temp=${arr[$j]}
			arr[$j]=${arr[$(($j+1))]}
			arr[$(($j+1))]=$temp
		fi
	done
done
echo ${arr[@]}
```

Lab 2

1. Run ps -aux, ls -l in order.

Ans.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/wait.h>
#include<sys/types.h>

int main()
{
	pid_t pid=fork();
	if(pid==0)
	{
		char *arg[]={"ps","-aux",NULL};
		execvp("ps",arg);
	}
	else
	{
		wait(0);
		pid=fork();
		if(pid==0)
		{
			char *arg[]={"ls","-l",NULL};
			execvp("ls",arg);
		}
	wait(0);
	}
}
```

2. Write a program to add numbers from a to b using three processes. a and b are given as input by user

Ans.

```
#include<stdio.h>
#include<sys/wait.h>
#include<sys/types.h>
#include<unistd.h>
#include<stdlib.h>
void main()
{
	printf("Program Started\n ");
	int start,end;
	scanf("%d",&start);
	scanf("%d",&end);
	pid_t pid=fork();
	if(pid==0)
	{
		pid_t pid2=fork();
		if(pid2==0)
		{
			int i;
			int sum=0;
			for(i=start;i<=(end-start+1)/3;i++)
			{
				sum=sum+i;
			}
			exit(sum);
		}
		else
		{
			int i;
			int sum=0;
```

```
                        int temp=(start+end-1)/3;
                        for(i=temp+1;i<=2*temp;i++)
                                sum+=i;
                        int status;
                        wait(&status);
                        exit(sum+WEXITSTATUS(status));
                }
        }
        else
        {
                int sum=0;
                int i;
                for(i=2*(start+end-1)/3+1;i<=end;i++)
                        sum+=i;
                int status;
                wait(&status);
                printf("%d\n",WEXITSTATUS(status)+sum);
        }

}
```

Lab 3

1. Implement above question using pipes.

Ans.

Lab 4

1. Write a program to implement FCFS in java.

Ans.

```
import java.util.Scanner;
class FCFS
{
	public static void sort(String[] process,int[] arrival_time,int[] burst_time)
	{
		String temp_process;
		int temp;
		for(int i=0;i<process.length;i++)
		{
			for(int j=0;j<process.length-1;j++)
			{
				if(arrival_time[j]>arrival_time[j+1])
				{
					temp_process=process[j];
					process[j]=process[j+1];
					process[j+1]=temp_process;
					temp=burst_time[j];
					burst_time[j]=burst_time[j+1];
					burst_time[j+1]=temp;
					temp=arrival_time[j];
					arrival_time[j]=arrival_time[j+1];
					arrival_time[j+1]=temp;
				}
			}
		}
	}
	public static void main(String args[])
	{
		String process[];
		int burst_time[];
		int arrival_time[];
		int turnaround_time[];
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter number of process");
		int p=sc.nextInt();
		process=new String[p];
		arrival_time=new int[p];
		burst_time=new int[p];
		turnaround_time=new int[p];
		for(int i=0;i<p;i++)
			process[i]=sc.next();
		System.out.println("Enter arrival time");
		for(int i=0;i<p;i++)
			arrival_time[i]=sc.nextInt();
		System.out.println("Enter burst time");
		for(int i=0;i<p;i++)
			burst_time[i]=sc.nextInt();
		int waiting_time[]=new int[p];
		sort(process,arrival_time,burst_time);
		waiting_time[0]=0;
		int total_time=arrival_time[0]+burst_time[0];
		turnaround_time[0]=waiting_time[0]+burst_time[0];
		for(int i=1;i<process.length;i++)
		{
			waiting_time[i]=(total_time-arrival_time[i])>0 ? total_time-arrival_time[i] : 0;
			total_time=total_time+burst_time[i]+((arrival_time[i]>total_time) ? arrival_time[i]-
total_time : 0);
			turnaround_time[i]=waiting_time[i]+burst_time[i];
```

```
                }
                display(process,arrival_time,burst_time,waiting_time,turnaround_time);
        }

        public static void display(String[] process,int arrival_time[],int burst_time[],int waiting_time[],int
turnaround_time[])
        {
                for(int i=0;i<process.length;i++)
                {

        System.out.print(process[i]+"\t"+arrival_time[i]+"\t"+burst_time[i]+"\t"+waiting_time[i]+"\t"+turnaround_tim
e[i]);
                        System.out.println();
                }
        }
}
```

2. Write a program to implement SJF in java.

Ans.

```
import java.util.ArrayList;
import java.util.Scanner;
public class SJF
{
   static class Data
   {
      int waiting_time;
      int turnaround_time;
      String PName;
      int arrival_time;
      int burst_time;
      Data(String _PName,int _at,int _bt)
      {
         this.PName=_PName;
         this.arrival_time=_at;
         this.burst_time=_bt;
         this.waiting_time=0;
         this.turnaround_time=0;
      }
   }
/* This function sorts array containing process details. If sort_by is 'a', data is sorted by arrival time.
   If sort_by is 'b' , it sorts by burst time
   Return value:- nothing
*/
   public static void sort(Data[] data,char sort_by)
   {
      Data temp;
      if(sort_by=='a')
      {
         for(int i=0;i<data.length;i++)
         {
            for(int j=0;j<data.length-1;j++)
            {
               if(data[j].arrival_time>=data[j+1].arrival_time)
               {
                  if(data[j].arrival_time==data[j+1].arrival_time)
                  {
                     if(data[j].burst_time>data[j+1].burst_time)
                     {
                        temp=data[j];
                        data[j]=data[j+1];
```

```
                        data[j+1]=temp;
                    }
                }
                else
                {
                    temp=data[j];
                    data[j]=data[j+1];
                    data[j+1]=temp;
                }
            }
        }
    }
    else
    {
        for(int i=0;i<data.length;i++)
        {
            for(int j=0;j<data.length-1;j++)
            {
                if(data[j].burst_time>data[j+1].burst_time)
                {
                    temp=data[j];
                    data[j]=data[j+1];
                    data[j+1]=temp;
                }
            }
        }
    }
}
public static void main(String args[])
{
    Scanner sc=new Scanner(System.in);
    Data[] data;
    System.out.println("Enter number of processes");
    int t=sc.nextInt();
    System.out.println("Enter name, arrival time and burst time for the processes");
    data = new Data[t];
    for(int i=0;i<t;i++)
        data[i]=new Data(sc.next(),sc.nextInt(),sc.nextInt());
    display(data);
    sort(data,'a');
    total_time=data[0].arrival_time;
    doSJF(data,0);
    System.out.println("After SJF");
    display(data);
}
static int total_time;
static int count=0;
/*
        This function performs SJF on set of processes whose details are included in data.
        start is starting value after which process has to be scheduled.
*/
public static void doSJF(Data[] data,int start)
{
    ArrayList<Data> temp=new ArrayList<>();
    data[start].waiting_time=(total_time-data[start].arrival_time)>0 ? total_time-data[start].arrival_time: 0;
    data[start].turnaround_time=data[start].burst_time+data[start].waiting_time;
    total_time=total_time+((data[start].arrival_time-total_time)>0 ? data[start].arrival_time-
total_time:0)+data[start].burst_time;
    int i=start+1;
    if(i !=data.length)
      while(total_time>data[i].arrival_time)
       {
            temp.add(data[i]);
            i++;
            if(i==data.length)
                break;
```

```
            }
        if(start==data.length-1)
            return;
        Data temp_array[]=new Data[temp.size()];
        temp_array=temp.toArray(temp_array);
        sort(temp_array,'b');
        if(temp_array.length!=0)
        {
            Data t;
            int index=find(data,temp_array[0],start);
            t=data[index];
            data[index]=data[start+1];
            data[start+1]=t;
        }
        doSJF(data,start+1);
    }
    public static int find(Data[] data,Data element,int start)
    {
        for(int i=start;i<data.length;i++)
        {
            if(data[i].equals(element))
                return i;
        }
        return -1;
    }
    public static void display(Data[] data)
    {
        for(int i=0;i<data.length;i++)
        {
            System.out.println(data[i].PName+"  "+data[i].arrival_time+"  "+data[i].burst_time+"
"+data[i].waiting_time+"  "+data[i].turnaround_time);
        }
    }
}
```

3. WAP to implement RR.

Ans.

```
import java.util.ArrayList;
import java.util.Scanner;

//timeslice is set as 3.

public class RR
{
    static class Data
    {
        int waiting_time;
        int turnaround_time;
        String PName;
        int arrival_time;
        int burst_time;
        Data(String _PName,int _at,int _bt)
        {
            this.PName=_PName;
            this.arrival_time=_at;
            this.burst_time=_bt;
            this.waiting_time=-1;
            this.turnaround_time=0;

        }
    }
```

```
/* This functions sorts by arrival time or burst time. If sort_by is 'a', it sorts by arrival time. This code is copied
from my SJF code which I have included */

   public static void sort(ArrayList<Data> data,char sort_by)
   {
      Data temp;
      if(sort_by=='a')
      {
         for(int i=0;i<data.size();i++)
         {
            for(int j=0;j<data.size()-1;j++)
            {
               if(data.get(j).arrival_time>data.get(j+1).arrival_time)
               {
                  temp=data.get(j);
                  data.set(j, data.get(j+1));
                  data.set(j+1,temp);
               }
            }
         }
      }
   }

   static int timeslice=10;
   static int total_time=0;
   static ArrayList<Data> res_list;

/* This method inserts a object of class Data in ArrayList<Data> data in correct position.
   Takes arguments:- ArrayList<Data> data which contains a list of details of processes.
   Element is Object of class Data which we have to insert
   total_time is service time till that moment when function is called
*/
   public static void insert(ArrayList<Data> data,Data element,int total_time)
   {
      int flag=1;
      for(int i=0;i<data.size();i++)
      {
         if(data.get(i).arrival_time>total_time)
         {
            data.add(i,element);
            flag=0;
            break;
         }
      }
      if(flag==1)
         data.add(element);
   }

/* This function performs round robin scheduling
   Takes argument:- ArrayList named data which contains arrival_time, burst_time etc. described in class Data
   Returns:- null
*/

   public static void roundrobin(ArrayList<Data> data)
   {
      if(data.get(0).waiting_time==-1)
      {
         data.get(0).waiting_time= (total_time-data.get(0).arrival_time) >0 ? total_time-data.get(0).arrival_time:0;
         total_time+=(data.get(0).arrival_time-total_time)>0? data.get(0).arrival_time-total_time : 0 ;
      }
      if(timeslice < data.get(0).burst_time)
      {
         total_time+=timeslice;
         data.get(0).burst_time=data.get(0).burst_time-timeslice;
         data.get(0).turnaround_time+=timeslice;
         insert(data,data.remove(0),total_time);
      }
```

```
        else
        {
            total_time+=data.get(0).burst_time;
            data.get(0).turnaround_time=total_time-data.get(0).arrival_time;
            res_list.add(data.remove(0));
        }
        if(data.isEmpty())
            return;
        roundrobin(data);
    }
    public static void main(String args[])
    {
        res_list=new ArrayList<>();
        Scanner sc=new Scanner(System.in);
        ArrayList<Data> data=new ArrayList<>();
        System.out.println("Enter number of processes.");
        int t=sc.nextInt();
        System.out.println("Enter process name, arrival time and burst time");
        for(int i=0;i<t;i++)
        {
            data.add(new Data(sc.next(),sc.nextInt(),sc.nextInt()));
        }
        sort(data,'a');
        total_time=data.get(0).arrival_time;
        roundrobin(data);
        System.out.println("After round robin.");
        display(res_list);
    }
    public static void display(ArrayList<Data> data)
    {
        for(int i=0;i<data.size();i++)
        {
            System.out.println(data.get(i).PName+"  "+data.get(i).arrival_time+"  "+data.get(i).burst_time+"
"+data.get(i).waiting_time+"  "+data.get(i).turnaround_time);
        }
    }
}
```

Lab 5

1. Write a program to multiply two matrices using threads.

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

int **result;
int **a;
int **b;
pthread_t th[3];
int r1,c1,r2,c2;
void* matrix_multiplication(void *d)
{
	int i,j,k;
	if(th[0]==pthread_self())
	{
		for(i=0;i<r1/3;i++)
		{
			for(j=0;j<c2;j++)
			{
				for(k=0;k<r2;k++)
				{
					result[i][j] += a[i][k]*b[k][j];
				}
			}
		}
	}
	else if(th[1]==pthread_self())
	{
		i=r1/3;
	 while(i<2*r1/3)
	 {
		for(j=0;j<c2;j++)
		{
			for(k=0;k<r2;k++)
			{
				result[i][j]+=a[i][k]*b[k][j];
			}
		}
				i++;
	 }
	}
	else if(th[2]==pthread_self())
	{
	 for(i=2*r1/3;i<r1;i++)
	 {
		for(j=0;j<c2;j++)
		{
			for(k=0;k<r2;k++)
			{
				result[i][j]+=a[i][k]*b[k][j];
			}
		}
	 }
	}
}
void main()
{
	int i,j;
	printf("\nEnter row and column number\n");
	scanf("%d\n%d",&r1,&c1);
	a=(int **)malloc(r1*sizeof(int));
	for(i=0;i<r1;i++)
```

```
        {
                a[i]=(int *)malloc(c1*sizeof(int));
        }
        printf("\nEnter the values\n");
        for(i=0;i<r1;i++)
        {
                for(j=0;j<c1;j++)
                {
                        printf("Enter value of a[%d][%d] ",i,j);
                        scanf("%d",&a[i][j]);
                }
        }
        printf("Enter row and column number\n");
    scanf("%d%d",&r2,&c2);
    b=(int **)malloc(r2*sizeof(int));
    for(i=0;i<r2;i++)
    {
         b[i]=(int *)malloc(c2*sizeof(int));
    }
    printf("\nEnter the values\n");
    for(i=0;i<r2;i++)
    {
            for(j=0;j<c2;j++)
            {
                    printf("Enter value of b[%d][%d] ",i,j);
              scanf("%d",&b[i][j]);
            }
        }
        result=(int **)malloc(r1*sizeof(int));
        for(i=0;i<r1;i++)
        {
                result[i]=(int *)malloc(c2*sizeof(int));
        }
        if(r2!=c1)
        {
                printf("Multiplication not possible. Program is terminating\n");
                exit(0);
        }
        for(i=0;i<3;i++)
        {
                pthread_create(&th[i],NULL,matrix_multiplication,NULL);
        }
        for(i=0;i<3;i++)
        {
                pthread_join(th[i],NULL);
        }
        for(i=0;i<r1;i++)
        {
                for(j=0;j<c2;j++)
                {
                        printf("%d\t",result[i][j]);
                }
                printf("\n");
        }
}
```

LAB 6

1. Write a program to sum all numbers from A to B using threads.

```
#include<pthread.h>
#include<stdio.h>
int *sum;
int num;
pthread_t th[3];
void* fun(void *a)
{
	int i;
	if(th[0]==pthread_self())
	{
		for(i=1;i<=num/3;i++)
			sum[0]=sum[0]+i;
	}
	else if(th[1]==pthread_self())
	{
		for(i=num/3+1;i<=2*num/3;i++)
			sum[1]=sum[1]+i;
	}
	else if(th[2]==pthread_self())
	{
		for(i=2*num/3+1;i<=num;i++)
			sum[2]=sum[2]+i;
	}
}
void main()
{
	scanf("%d",&num);
	int i,j;
	sum=(int *)malloc(3*sizeof(int));
	for(i=0;i<3;i++)
	{
		pthread_create(&th[i],NULL,fun,NULL);
	}
	pthread_join(th[0],NULL);
	pthread_join(th[1],NULL);
	pthread_join(th[2],NULL);
	printf("%d\n",sum[0]+sum[1]+sum[2]);
}
```

2. Write a program to implement parallel product using threads.

```
#include<pthread.h>
#include<stdio.h>
int *sum;
int num;
pthread_t th[3];
void* fun(void *a)
{
	int i;
	if(th[0]==pthread_self())
	{
		sum[0]=1;
		for(i=1;i<=num/3;i++)
			sum[0]=sum[0]*i;
	}
	else if(th[1]==pthread_self())
	{
		sum[1]=1;
		for(i=num/3+1;i<=2*num/3;i++)
```

```
                sum[1]=sum[1]*i;
        }
        else if(th[2]==pthread_self())
        {
                sum[2]=1;
                for(i=2*num/3+1;i<=num;i++)
                        sum[2]=sum[2]*i;
        }
}
void main()
{
        scanf("%d",&num);
        int i,j;
        sum=(int *)malloc(3*sizeof(int));
        for(i=0;i<3;i++)
        {
                pthread_create(&th[i],NULL,fun,NULL);
        }
        pthread_join(th[0],NULL);
        pthread_join(th[1],NULL);
        pthread_join(th[2],NULL);
        printf("%d\n",sum[0]*sum[1]*sum[2]);
}
```

3. Write a program to find prime numbers upto 100 using threads.

```
#include<stdio.h>
#include<pthread.h>
#include<stdbool.h>
bool prime[101];
pthread_t th[3];
void* find_prime()
{
        int i,j;
        if(th[0]==pthread_self())
        {
                for(i=2;i<33;i++)
                {
                        if(prime[i]==1)
                        {
                                j=2;
                                while(i*j<101)
                                {
                                        prime[i*j]=0;
                                        ++j;
                                }
                        }
                }
        }
        if(th[1]==pthread_self())
    {
         for(i=33;i<66;i++)
         {
               if(prime[i]==1)
               {
                    j=2;
                    while(i*j<101)
                    {
                          prime[i*j]=0;
                          ++j;
                    }
               }
        }
    }
```

```
        if(th[2]==pthread_self())
    {
         for(i=66;i<101;i++)
         {
                if(prime[i]==1)
                {
                        j=2;
                        while(i*j<101)
                        {
                                prime[i*j]=0;
                                ++j;
                        }
                }
         }
    }
}

void main()
{
        int i;
        memset(prime,1,sizeof(prime));
        for(i=0;i<3;i++)
                pthread_create(&th[i],NULL,find_prime,NULL);
        for(i=0;i<3;i++)
                pthread_join(th[i],NULL);
        for(i=2;i<101;i++)
                if(prime[i]==true)
                        printf("%d\n",i);
}
```

4. Write a program to implement Block Matrix Multiplication using threads.

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

pthread_t th[4];
int **result;
int **a,**b;
int r1,c1,r2,c2;
void* block_mm()
{
        int i,j,k;
        if(th[0]==pthread_self())
        {
                for(i=0;i<r1/2;i++)
                {
                        for(j=0;j<c2/2;j++)
                        {
                                for(k=0;k<r2;k++)
                                {
                                        result[i][j]+=a[i][k]*b[k][j];
                                }
                        }
                }
        }
        if(th[1]==pthread_self())
    {
         for(i=r1/2;i<r1;i++)
         {
                for(j=0;j<c2/2;j++)
                {
                    for(k=0;k<r2;k++)
                    {
                            result[i][j]+=a[i][k]*b[k][j];
                    }
```

```
                }
            }
        }
        if(th[2]==pthread_self())
        {
            for(i=r1/2;i<r1;i++)
            {
                for(j=c2/2;j<c2;j++)
                {
                    for(k=0;k<r2;k++)
                    {
                        result[i][j]+=a[i][k]*b[k][j];
                    }
                }
            }
        }
        if(th[3]==pthread_self())
        {
            for(i=0;i<r1/2;i++)
            {
                for(j=c2/2;j<c2;j++)
                {
                    for(k=0;k<r2;k++)
                    {
                        result[i][j]+=a[i][k]*b[k][j];
                    }
                }
            }
        }
}

void main()
{
        int i,j;
        printf("Enter number of rows and columns for first matrix\n");
        scanf("%d%d",&r1,&c1);
        printf("Enter number of rows and columns for second matrix\n");
        scanf("%d%d",&r2,&c2);
        if(c1!=r2)
        {
                printf("Multiplication not possible");
                exit(0);
        }
        a=(int **)malloc(r1*sizeof(int));
        for(i=0;i<r1;i++)
        {
                a[i]=(int *)malloc(c1*sizeof(int));
        }
        b=(int **)malloc(r2*sizeof(int));
        for(i=0;i<r2;i++)
        {
                b[i]=(int *)malloc(c2*sizeof(int));
        }
        result=(int **)malloc(r1*sizeof(int));
        for(i=0;i<r1;i++)
        {
                result[i]=(int *)malloc(c2*sizeof(int));
        }
        printf("Enter elements of first array");
        for(i=0;i<r1;i++)
                for(j=0;j<c1;j++)
                        scanf("%d",&a[i][j]);
        printf("Enter elements of second array");
        for(i=0;i<r2;i++)
                for(j=0;j<c2;j++)
                        scanf("%d",&b[i][j]);
        for(i=0;i<4;i++)
```

```
                pthread_create(&th[i],NULL,*block_mm,NULL);
        for(i=0;i<4;i++)
                pthread_join(th[i],NULL);
        for(i=0;i<r1;i++)
        {
                for(j=0;j<c2;j++)
                        printf("%d\t",result[i][j]);
                printf("\n");
        }
}
```

Lab 7

1. Write a program to implement Peterson algorithm for two process.

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

int BUFFER_MAX=10;
int counter=0;
int turn;  // turn = 0 will be turn of producer and turn = 1 will be turn of consumer
bool flag[2];         //flag[0] will be a flag for producer and flag[1] will be flag for consumer
int buffer[10];
void* producer()
{
        int i=0;
        int k=0;
        while(k<20)
        {
                while(counter==BUFFER_MAX);
                buffer[i]=10*i;
                i=(i+1)%BUFFER_MAX;
                turn=1;
                flag[0]=true;
                while(flag[1]&&turn==1);    //Wait
                counter++;
                printf("Producer counter is %d\n",counter);
                flag[0]=false;
                k++;
        }
}
void* consumer()
{
        int i=0;
        int k=0,itemProcessed=0;
        while(k<20)
        {
                while(counter==0);
                itemProcessed = buffer[counter-1];
                flag[1]=true;
                turn=0;
                while(flag[0]&&turn==0);    //Wait
                counter--;            // critical section
                printf("Consumer counter is %d\n",counter);
                flag[1]=false;
                k++;
        }
}
int main()
{
        pthread_t Producer,Consumer;
        pthread_create(&Producer,NULL,producer,NULL);
        pthread_create(&Consumer,NULL,consumer,NULL);
        pthread_join(Producer,NULL);
        pthread_join(Consumer,NULL);
        return 1;
}
```

Lab 8

1. Write a program to calculate dot product of two vectors using semaphores.

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

int sum=0;
-int *vector;
int num;
pthread_t th[3];
sem_t sem;
void* fun()
{
	int i,temp_sum=0;
	if(th[0]==pthread_self())
	{
		for(i=1;i<=num/3;i++)
		{
			temp_sum=temp_sum+vector[i]*vector[i];
		}
		sem_wait(&sem);
		sum=temp_sum+sum;
		sem_post(&sem);
	}
	else if(th[1]==pthread_self())
	{
		temp_sum=0;
		for(i=num/3+1;i<=2*num/3;i++)
		{
			temp_sum+=vector[i]*vector[i];
		}
		sem_wait(&sem);
		sum+=temp_sum;
		sem_post(&sem);
	}
	else if(th[2]==pthread_self())
	{
		temp_sum=0;
		for(i=2*num/3+1;i<=num;i++)
		{
			temp_sum+=vector[i]*vector[i];
		}
		sem_wait(&sem);
		sum+=temp_sum;
		sem_post(&sem);
	}
}
void main()
{
	int i,j;
	sem_init(&sem,0,1);
	printf("Enter dimension of vector\n");
	scanf("%d",&num);
	vector=(int *)malloc((num+1)*sizeof(int));
	printf("Enter components of vector\n");
	for(i=1;i<=num;i++)
		scanf("%d",&vector[i]);
	for(i=0;i<3;i++)
	{
		pthread_create(&th[i],NULL,fun,NULL);
	}
	pthread_join(th[0],NULL);
```

```
	pthread_join(th[1],NULL);
	pthread_join(th[2],NULL);
	printf("%d\n",sum);
	sem_close(&sem);
}
```

2. Write a program to solve the following problem: - A mess has a capacity of k students, total number of students are n. Use semaphore so that every student must eat in mess.

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

sem_t sem;
pthread_mutex_t mutex;
pthread_t *student;

int capacity;
void* mess()
{
	int i=0,j=0;
	sem_wait(&sem);
	while(j<100000000)
	{
		while(i<100000000)
		{
			i++;
		}
		j++;
	}
	printf("Inside Critical Section\n");
	sem_post(&sem);
	printf("Outside Critical Section\n");
}
int main()
{
	int num,i;
	printf("Enter the capacity\n");
	scanf("%d",&capacity);
	printf("Enter number of students\n");
	scanf("%d",&num);
	student=(pthread_t *)malloc(sizeof(pthread_t)*num);
	sem_init(&sem,0,capacity);
	pthread_mutex_init(&mutex,NULL);
	for(i=0;i<num;i++)
		pthread_create(&student[i],NULL,*mess,NULL);
	for(i=0;i<num;i++)
		pthread_join(student[i],NULL);
	sem_close(&sem);
}
```

3. Write a program to implement third reader writer problem using semaphores.

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

sem_t serviceQueue;
sem_t resource;
sem_t readValueAccess;
int readValue;
pthread_t *reader,*writer;

void* Reader()
{
	sem_wait(&serviceQueue);
	sem_wait(&readValueAccess);
	if(readValue==0)
		sem_wait(&resource);
	readValue++;
	sem_post(&serviceQueue);
	sem_post(&readValueAccess);
	printf("I am reading\n");
	sem_wait(&readValueAccess);
	readValue--;
	if(readValue==0)
		sem_post(&resource);
	sem_post(&readValueAccess);
}

void* Writer()
{
	sem_wait(&serviceQueue);
	sem_wait(&resource);
	printf("I am writing\n");
	sem_post(&serviceQueue);
	sem_post(&resource);
}

int main()
{
	int i,num_reader ,num_writer;
	printf("Enter number of readers\n");
	scanf("%d",&num_reader);
	printf("Enter number of writers\n");
	scanf("%d",&num_writer);
	reader=(pthread_t *)malloc(num_reader*sizeof(pthread_t));
	writer=(pthread_t *)malloc(num_writer*sizeof(pthread_t));
	readValue=0;
	sem_init(&serviceQueue,0,1);
	sem_init(&resource,0,1);
	sem_init(&readValueAccess,0,1);
	for(i=0;i<num_reader;i++)
		pthread_create(&reader[i],NULL,Reader,NULL);
	for(i=0;i<num_writer;i++)
		pthread_create(&writer[i],NULL,Writer,NULL);
	for(i=0;i<num_writer;i++)
		pthread_join(writer[i],NULL);
	for(i=0;i<num_reader;i++)
		pthread_join(reader[i],NULL);
	sem_close(&serviceQueue);
	sem_close(&resource);
	sem_close(&readValueAccess);
	return 0;
}
```