22CSE137 - Operating systems Lab (0 0 4)

Lab Manual

Program Name: CSE-CS, AIML and AIDE

Course: Operating Systems Lab

Semester: 2 Term: 4

SME: Dr.Gnanaprakasam T. Dr. Arjun M, Ms. Naiwrita Borah, Mr. Deepanshu S Satwaliya, Ms. Sapna Gupta

Program Name: To Implement Scheduling Algorithms FCFS		
1	**Background & Use-case**	Process is a program in Execution. Process life cycle include two cycles: CPU Cycles and I/o Cycles. Process to be executed with CPU waits in ready queue. In order to improve CPU utilization, various schemes are rendered, in the order in which process are allotted to the CPU. FCFS algorithm works on the principle of First Come First Serve.
2	**Problem Statement**	Implement FCFS the current state of the system as follows. Process CPU BURST P1 1 P2 2 P3 3

3

Research Methodology

Algorithmic Steps

1)Input the CPU burst time of processes bt[i]
2)Evaluate waiting time as wt[i]= wt[i-1] + bt[i-1]
3)Evaluate turnaround time as tt[i] =wt[i]+bt[i]
4)Evaluate avg waiting time =total waiting time/n
5)Evaluate avg turnaround time=total turnaround time/n

Code

```
import java.util.*;
 public class FCFS {
public static void main(String args[])
{
Scanner sc = new Scanner(System.in);
System.out.println("enter no of process: ");
int n = sc.nextInt();
int pid[] = new int[n];   // process ids
int ar[] = new int[n];     // arrival times
int bt[] = new int[n];     // burst or execution times
int ct[] = new int[n];     // completion times
int ta[] = new int[n];     // turn around times
int wt[] = new int[n];     // waiting times
int temp;
float avgwt=0,avgta=0;

for(int i = 0; i < n; i++)
{
System.out.println("enter process " + (i+1) + " arrival time: ");
ar[i] = sc.nextInt();
System.out.println("enter process " + (i+1) + " brust time: ");
bt[i] = sc.nextInt();
pid[i] = i+1;
}

//sorting according to arrival times
for(int i = 0 ; i <n; i++)
{
for(int  j=0;  j < n-(i+1) ; j++)
```

```
{
if( ar[j] > ar[j+1] )
{
temp = ar[j];
ar[j] = ar[j+1];
ar[j+1] = temp;
temp = bt[j];
bt[j] = bt[j+1];
bt[j+1] = temp;
temp = pid[j];
pid[j] = pid[j+1];
pid[j+1] = temp;
}
}
}
// finding completion times
for(int  i = 0 ; i < n; i++)
{
if( i == 0)
{
ct[i] = ar[i] + bt[i];
}
else
{
if( ar[i] > ct[i-1])
{
ct[i] = ar[i] + bt[i];
}
else
ct[i] = ct[i-1] + bt[i];
}
ta[i] = ct[i] - ar[i] ;          // turnaround time= completion time- arrival time
wt[i] = ta[i] - bt[i] ;          // waiting time= turnaround time- burst time
avgwt += wt[i] ;               // total waiting time
avgta += ta[i] ;               // total turnaround time
}
System.out.println("\npid  arrival  brust  complete turn waiting");
for(int  i = 0 ; i< n;  i++)
{
```

```
System.out.println(pid[i] + "  \t " + ar[i] + "\t" + bt[i] + "\t" + ct[i] + "\t" +
ta[i] + "\t"  + wt[i] ) ;
}
sc.close();
System.out.println("\naverage waiting time: "+ (avgwt/n));     // printing average
waiting time.
System.out.println("average turnaround time:"+(avgta/n));    // printing average
turnaround time.
}
}
```

4	**Skills Required**	Programming in Java and knowledge of IDE to execute the program
5	**Expected output**	Waiting Time of each process, Turnaround time, Average Waiting time, Average Turnaround time Snapshot of the Result

```
Enter the number of processes: 3
Enter the burst time for each process:
Process 1: 1
Process 2: 2
Process 3: 3
Execution Table:
+------+------------+--------------+--------------+
| PID | Burst Time | Waiting Time | Turnaround Time |
+------+------------+--------------+--------------+
|  P1 |      1     |       0      |      1       |
+------+------------+--------------+--------------+
|  P2 |      2     |       1      |      3       |
+------+------------+--------------+--------------+
|  P3 |      3     |       3      |      6       |
+------+------------+--------------+--------------+

Gantt chart:
+--------++--------++--------++
|  P1 |  P2 |  P3 |
+--------++--------++--------++
0     1     3     6
```

		Average waiting time: 1.33 Average turnaround time: 3.33

Program Name: To Implement Scheduling Algorithms SJF		
1	**Background & Use-case**	Process is a program in Execution. Process life cycle includes two cycles: CPU Cycles and I/o Cycles. Process to be executed with CPU waits in the ready queue. In order to improve CPU utilization, various schemes are rendered, in the order in which processes are allotted to the CPU. SJF algorithm works on the principle of Shortest jobs will be executed First
2	**Problem Statement**	Implement FCFS the current state of the system as follows. Process CPU BURST P1 1 P2 2 P3 3
3	**Research Methodology**	Algorithmic Steps 1)Input the CPU burst time of processes bt[i] 2)Sort the process in the ascending order of their burst time. 3)Evaluate waiting time as wt[i]= wt[i-1] + bt[i-1]. 4)Evaluate turnaround time as tt[i] =wt[i]+bt[i] 5)Evaluate avg waiting time =total waiting time/n

6)Evaluate avg turnaround time=total turnaround time/n

Code

```
import java.util.*;

public class SJF {
public static void main(String args[])
{
Scanner sc = new Scanner(System.in);
System.out.println ("enter no of process:");
int n = sc.nextInt();
int pid[] = new int[n];
int at[] = new int[n]; // at means arrival time
int bt[] = new int[n]; // bt means burst time
int ct[] = new int[n]; // ct means complete time
int ta[] = new int[n]; // ta means turn around time
int wt[] = new int[n];  //wt means waiting time
int f[] = new int[n];
int st=0, tot=0;
float avgwt=0, avgta=0;
for(int i=0;i<n;i++)
{
System.out.println ("enter process " + (i+1) + " arrival time:");
at[i] = sc.nextInt();
System.out.println ("enter process " + (i+1) + " brust time:");
bt[i] = sc.nextInt();
pid[i] = i+1;
f[i] = 0;
}
boolean a = true;
while(true)
{
int c=n, min=999;
if (tot == n) // total no of process = completed process loop will be terminated
break;
for (int i=0; i<n; i++)
{
/*
* If i'th process arrival time <= system time and its flag=0 and burst<min
* That process will be executed first
*/
if ((at[i] <= st) && (f[i] == 0) && (bt[i]<min))
{
min=bt[i];
c=i;
}
}
/* If c==n means c value can not updated because no process arrival time< system time
so we increase the system time */
if (c==n)
st++;
else
{
ct[c]=st+bt[c];
st+=bt[c];
ta[c]=ct[c]-at[c];
wt[c]=ta[c]-bt[c];
f[c]=1;
tot++;
}
}
System.out.println("\npid  arrival brust  complete turn waiting");
for(int i=0;i<n;i++)
{
avgwt+= wt[i];
avgta+= ta[i];
System.out.println(pid[i]+"\t"+at[i]+"\t"+bt[i]+"\t"+ct[i]+"\t"+ta[i]+"\t"+wt[i]);
}
System.out.println ("\naverage tat is "+ (float)(avgta/n));
System.out.println ("average wt is "+ (float)(avgwt/n));
sc.close();
}
}
```

4	**Skills Required**	Programming in Java and knowledge of IDE to execute the program
5	**Expected output**	Waiting Time of each process, Turnaround time, Average Waiting time, Average Turnaround time Snapshot of the Result Enter number of processes: 3 Enter Burst Time: Process 1: 3 Process 2: 2 Process 3: 1 Process Burst Time Waiting Time Turnaround Time P3 1 0 1 P2 2 1 3 P1 3 3 6 Average Waiting Time = 1.33 Average Turnaround Time = 3.33

Program Name: To Implement Scheduling Algorithms priority based scheduling		
1	**Background & Use-case**	Process is a program in Execution. Process life cycle includes two cycles: CPU Cycles and I/o Cycles. Process to be executed with CPU waits in the ready queue. In order to improve CPU utilization, various schemes are rendered, in the order in which processes are allotted to the CPU. Priority algorithm works on the principle of highest priority jobs will be Executed First
2	**Problem Statement**	Implement Priority Scheduling the current state of the system as follows. Process CPU BURST Priority P1 1 2

		P2 2 1 P3 3 3

3	**Research Methodology**	Algorithmic Steps
		1)Input the CPU burst time of processes bt[i]
		2)Sort the process in the ascending order of their priority.
		3)Evaluate waiting time as wt[i]= wt[i-1] + bt[i-1].
		4)Evaluate turnaround time as tt[i] =wt[i]+bt[i]
		5)Evaluate avg waiting time =total waiting time/n
		6)Evaluate avg turnaround time=total turnaround time/n
		
		Code
		
		import java.util.Scanner;
		
		public class NonPreemptivePriorityCPUSchedulingAlgorithm
		{
		
		    int burstTime[];
		    int priority[];
		    int arrivalTime[];
		    String[] processId;
		    int numberOfProcess;
		
		    void getProcessData(Scanner input)
		    {
		        System.out.print("Enter the number of Process for Scheduling : ");
		        int inputNumberOfProcess = input.nextInt();
		        numberOfProcess = inputNumberOfProcess;
		        burstTime = new int[numberOfProcess];
		        priority = new int[numberOfProcess];
		        arrivalTime = new int[numberOfProcess];
		        processId = new String[numberOfProcess];
		        String st = "P";
		        for (int i = 0; i < numberOfProcess; i++)

```
        {
            processId[i] = st.concat(Integer.toString(i));
            System.out.print("Enter the burst time   for Process - " + (i) + " : ");
            burstTime[i] = input.nextInt();
            System.out.print("Enter the arrival time for Process - " + (i) + " : ");
            arrivalTime[i] = input.nextInt();
            System.out.print("Enter the priority     for Process - " + (i) + " : ");
            priority[i] = input.nextInt();
        }
    }

    void sortAccordingArrivalTimeAndPriority(int[] at, int[] bt, int[] prt, String[]
pid)
    {

        int temp;
        String stemp;
        for (int i = 0; i < numberOfProcess; i++)
        {

            for (int j = 0; j < numberOfProcess - i - 1; j++)
            {
                if (at[j] > at[j + 1])
                {
                    //swapping arrival time
                    temp = at[j];
                    at[j] = at[j + 1];
                    at[j + 1] = temp;

                    //swapping burst time
                    temp = bt[j];
                    bt[j] = bt[j + 1];
                    bt[j + 1] = temp;

                    //swapping priority
                    temp = prt[j];
                    prt[j] = prt[j + 1];
                    prt[j + 1] = temp;
                    //swapping process identity
                    stemp = pid[j];
                    pid[j] = pid[j + 1];
                    pid[j + 1] = stemp;

                }
                //sorting according to priority when arrival timings are same
                if (at[j] == at[j + 1])
                {
                    if (prt[j] > prt[j + 1])
                    {
                        //swapping arrival time
                        temp = at[j];
                        at[j] = at[j + 1];
                        at[j + 1] = temp;

                        //swapping burst time
                        temp = bt[j];
                        bt[j] = bt[j + 1];
                        bt[j + 1] = temp;

                        //swapping priority
                        temp = prt[j];
                        prt[j] = prt[j + 1];
                        prt[j + 1] = temp;

                        //swapping process identity
                        stemp = pid[j];
                        pid[j] = pid[j + 1];
                        pid[j + 1] = stemp;

                    }
                }
            }

        }
    }

    void priorityNonPreemptiveAlgorithm()
```

```
    {
        int finishTime[] = new int[numberOfProcess];
        int bt[] = burstTime.clone();
        int at[] = arrivalTime.clone();
        int prt[] = priority.clone();
        String pid[] = processId.clone();
        int waitingTime[] = new int[numberOfProcess];
        int turnAroundTime[] = new int[numberOfProcess];

        sortAccordingArrivalTimeAndPriority(at, bt, prt, pid);

        //calculating waiting & turn-around time for each process
        finishTime[0] = at[0] + bt[0];
        turnAroundTime[0] = finishTime[0] - at[0];
        waitingTime[0] = turnAroundTime[0] - bt[0];

        for (int i = 1; i < numberOfProcess; i++)
        {
            finishTime[i] = bt[i] + finishTime[i - 1];
            turnAroundTime[i] = finishTime[i] - at[i];
            waitingTime[i] = turnAroundTime[i] - bt[i];
        }
        float sum = 0;
        for (int n : waitingTime)
        {
            sum += n;
        }
        float averageWaitingTime = sum / numberOfProcess;

        sum = 0;
        for (int n : turnAroundTime)
        {
            sum += n;
        }
        float averageTurnAroundTime = sum / numberOfProcess;

        //print on console the order of processes along with their finish time & turn
around time
        System.out.println("Priority Scheduling Algorithm : ");
```

<table>
<tr><td></td><td></td><td><pre>
        System.out.format("%20s%20s%20s%20s%20s%20s%20s\n", "ProcessId", "BurstTime",
"ArrivalTime", "Priority", "FinishTime", "WaitingTime", "TurnAroundTime");
        for (int i = 0; i < numberOfProcess; i++) {
            System.out.format("%20s%20d%20d%20d%20d%20d%20d\n", pid[i], bt[i], at[i],
prt[i], finishTime[i], waitingTime[i], turnAroundTime[i]);
        }

        System.out.format("%100s%20f%20f\n", "Average", averageWaitingTime,
averageTurnAroundTime);
    }

    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);
        NonPreemptivePriorityCPUSchedulingAlgorithm obj = new
NonPreemptivePriorityCPUSchedulingAlgorithm();
        obj.getProcessData(input);
        obj.priorityNonPreemptiveAlgorithm();
    }
}
</pre></td></tr>
<tr><td>4</td><td><b>Skills Required</b></td><td>Programming in Java and knowledge of IDE to execute the program</td></tr>
<tr><td>5</td><td><b>Expected output</b></td><td>Waiting Time of each process, Turnaround time, Average Waiting time, Average Turnaround time

Snapshot of the Result<pre>
Enter number of processes: 3
Enter Burst Time and Priority:
Process 1: 1
2
Process 2: 2
1
Process 3: 3
3
Process   Burst Time Priority   Waiting TimeTurnaround Time
</pre></td></tr>
</table>

		P2 2 1 0 2 P1 1 2 2 3 P3 3 3 3 6 Average Waiting Time = 1.67 Average Turnaround Time = 3.67

Program Name: To Implement Scheduling Algorithms RR		
1	**Background & Use-case**	Process is a program in Execution. Process life cycle includes two cycles: CPU Cycles and I/o Cycles. Process to be executed with CPU waits in the ready queue. In order to improve CPU utilization, various schemes are rendered, in the order in which processes are allotted to the CPU. The RR algorithm works on the principle of distribution of fixed timeslice among the processes..
2	**Problem Statement**	Process CPU BURST

		P1 P2 P3	5 10 15
3	**Research Methodology**	Code	

```
public class Roundsch
{
        // Method to find the waiting time for all
        // processes
        static void findWaitingTime(int processes[], int n,
                                int bt[], int wt[], int quantum)
        {
                // Make a copy of burst times bt[] to store remaining
                // burst times.
                int rem_bt[] = new int[n];
                for (int i = 0 ; i < n ; i++)
                        rem_bt[i] = bt[i];

                int t = 0; // Current time

                // Keep traversing processes in round robin manner
                // until all of them are not done.
                while(true)
                {
                        boolean done = true;

                        // Traverse all processes one by one repeatedly
                        for (int i = 0 ; i < n; i++)
                        {
                                // If burst time of a process is greater than 0
                                // then only need to process further
                                if (rem_bt[i] > 0)
                                {
```

```
                done = false; // There is a pending process

                if (rem_bt[i] > quantum)
                {
                    // Increase the value of t i.e. shows
                    // how much time a process has been processed
                    t += quantum;

                    // Decrease the burst_time of current process
                    // by quantum
                    rem_bt[i] -= quantum;
                }

                // If burst time is smaller than or equal to
                // quantum. Last cycle for this process
                else
                {
                    // Increase the value of t i.e. shows
                    // how much time a process has been processed
                    t = t + rem_bt[i];

                    // Waiting time is current time minus time
                    // used by this process
                    wt[i] = t - bt[i];

                    // As the process gets fully executed
                    // make its remaining burst time = 0
                    rem_bt[i] = 0;
                }
            }
        }

        // If all processes are done
        if (done == true)
        break;
    }
}

// Method to calculate turn around time
```

```
static void findTurnAroundTime(int processes[], int n,
                                    int bt[], int wt[], int tat[])
{
      // calculating turnaround time by adding
      // bt[i] + wt[i]
      for (int i = 0; i < n ; i++)
            tat[i] = bt[i] + wt[i];
}

// Method to calculate average time
static void findavgTime(int processes[], int n, int bt[],
                                                      int quantum)
{
      int wt[] = new int[n], tat[] = new int[n];
      int total_wt = 0, total_tat = 0;

      // Function to find waiting time of all processes
      findWaitingTime(processes, n, bt, wt, quantum);

      // Function to find turn around time for all processes
      findTurnAroundTime(processes, n, bt, wt, tat);

      // Display processes along with all details
      System.out.println("PN " + " B " +
                        " WT " + " TAT");

      // Calculate total waiting time and total turn
      // around time
      for (int i=0; i<n; i++)
      {
            total_wt = total_wt + wt[i];
            total_tat = total_tat + tat[i];
            System.out.println(" " + (i+1) + "\t\t" + bt[i] +"\t " +
                                    wt[i] +"\t\t " + tat[i]);
      }

      System.out.println("Average waiting time = " +
                              (float)total_wt / (float)n);
      System.out.println("Average turn around time = " +
                                                            (float)total_tat / (float)n);
          }

          // Driver Method
          public static void main(String[] args)
          {
                    // process id's
                    int processes[] = { 1, 2, 3};
                    int n = processes.length;

                    // Burst time of all processes
                    int burst_time[] = {10, 5, 8};

                    // Time quantum
                    int quantum = 2;
                    findavgTime(processes, n, burst_time, quantum);
          }
}
```

4	**Skills Required**	Programming in Java and knowledge of IDE to execute the program
5	**Expected output**	Waiting Time of each process, Turnaround time, Average Waiting time, Average Turnaround time Snapshot of the Result

```
Enter number of processes: 3
Enter Burst Time:
Process 1: 5
Process 2: 10
Process 3: 15
Enter Time Quantum: 5
Process     Burst Time  Waiting TimeTurnaround Time
P1          5           0           5
P2          10          10          20
P3          15          15          30

Average Waiting Time = 8.33
```

		Average Turnaround Time = 18.33

Program Name: Bankers Algorithm (Merin,Nawrita & Sreenidhi)		
1	**Background & Use-case**	Deadlock Avoidance Algorithm
2	**Problem Statement**	To run the safety algorithm to avoid Deadlock
3	**Research Methodology**	

```
//Java Program for Bankers Algorithm
public class Bankers
{
      int n = 5; // Number of processes
      int m = 3; // Number of resources
      int need[][] = new int[n][m];
      int [][]max;
      int [][]alloc;
      int []avail;
      int safeSequence[] = new int[n];

      void initializeValues()
      {
      // P0, P1, P2, P3, P4 are the Process names here
      // Allocation Matrix
      alloc = new int[][] { { 0, 1, 0 }, //P0
                            { 2, 0, 0 }, //P1
                            { 3, 0, 2 }, //P2
                            { 2, 1, 1 }, //P3
                            { 0, 0, 2 } }; //P4

      // MAX Matrix
      max = new int[][] { { 7, 5, 3 }, //P0
                        { 3, 2, 2 }, //P1
                        { 9, 0, 2 }, //P2
                        { 2, 2, 2 }, //P3
                        { 4, 3, 3 } }; //P4
```

```
// Available Resources
avail = new int[] { 3, 3, 2 };
}

void isSafe()
{
int count=0;

//visited array to find the already allocated process
boolean visited[] = new boolean[n];
for (int i = 0;i < n; i++)
{
      visited[i] = false;
}

//work array to store the copy of available resources
int work[] = new int[m];
for (int i = 0;i < m; i++)
{
      work[i] = avail[i];
}

while (count<n)
{
      boolean flag = false;
      for (int i = 0;i < n; i++)
      {
            if (visited[i] == false)
            {
            int j;
            for (j = 0;j < m; j++)
            {
                  if (need[i][j] > work[j])
                  break;
            }
            if (j == m)
            {
            safeSequence[count++]=i;
```

```
        visited[i]=true;
        flag=true;

            for (j = 0;j < m; j++)
            {
            work[j] = work[j]+alloc[i][j];
            }
        }
        }
    }
    if (flag == false)
    {
        break;
    }
}
if (count < n)
{
    System.out.println("The System is UnSafe!");
}
else
{
    //System.out.println("The given System is Safe");
    System.out.println("Following is the SAFE Sequence");
            for (int i = 0;i < n; i++)
    {
        System.out.print("P" + safeSequence[i]);
        if (i != n-1)
        System.out.print(" -> ");
    }
}
}

void calculateNeed()
{
for (int i = 0;i < n; i++)
{
    for (int j = 0;j < m; j++)
    {
    need[i][j] = max[i][j]-alloc[i][j];
```

```
            }
        }
        }

        public static void main(String[] args)
        {
        int i, j, k;
        GfGBankers gfg = new GfGBankers();

        gfg.initializeValues();
        //Calculate the Need Matrix
        gfg.calculateNeed();

        // Check whether system is in safe state or not
        gfg.isSafe();
        }
}
```

4	**Skills Required**	Java programming
5	**Expected output**	System is in safe state and following is the safe sequence: P1, P4, P2, P3, P0

Program Name: To Implement Solution for Classic Problems of Synchronization		
1	**Background & Use-case**	Process Synchronisation is used to Synchronize the execution of critical Section Statement.

2	**Problem Statement**	Implement the solution to the classic Problem of Synchronization Producer -Consumer Problem.
3	**Research Methodology**	Code:

```
// Java implementation of a producer and consumer
// that use semaphores to control synchronization.

import java.util.concurrent.Semaphore;

class Q {
        // an item
        int item;

        // semCon initialized with 0 permits
        // to ensure put() executes first
        static Semaphore semCon = new Semaphore(0);

        static Semaphore semProd = new Semaphore(1);

        // to get an item from buffer
        void get()
        {
                try {
                        // Before consumer can consume an item,
                        // it must acquire a permit from semCon
                        semCon.acquire();
                }
                catch (InterruptedException e) {
                        System.out.println("InterruptedException caught");
                }

                // consumer consuming an item
                System.out.println("Consumer consumed item : " + item);

                // After consumer consumes the item,
                // it releases semProd to notify producer
                semProd.release();
        }

        // to put an item in buffer
        void put(int item)
        {
                try {
                        // Before producer can produce an item,
                        // it must acquire a permit from semProd
                        semProd.acquire();
                }
                catch (InterruptedException e) {
                        System.out.println("InterruptedException caught");
                }

                // producer producing an item
                this.item = item;

                System.out.println("Producer produced item : " + item);

                // After producer produces the item,
                // it releases semCon to notify consumer
                semCon.release();
        }
}

// Producer class
class Producer implements Runnable {
        Q q;
        Producer(Q q)
        {
                this.q = q;
                new Thread(this, "Producer").start();
        }

        public void run()
        {
                for (int i = 0; i < 5; i++)
                        // producer put items
                        q.put(i);
```

```
      }
}

// Consumer class
class Consumer implements Runnable {
      Q q;
      Consumer(Q q)
      {
            this.q = q;
            new Thread(this, "Consumer").start();
      }

      public void run()
      {
            for (int i = 0; i < 5; i++)
                  // consumer get items
                  q.get();
      }
}

// Driver class
class PC {
      public static void main(String args[])
      {
            // creating buffer queue
            Q q = new Q();

            // starting consumer thread
            new Consumer(q);

            // starting producer thread
            new Producer(q);
      }
}
```

4	**Skills Required**	Java programming
5	**Expected output**	Producer produced item : 0 Consumer consumed item : 0 Producer produced item : 1 Consumer consumed item : 1 Producer produced item : 2 Consumer consumed item : 2 Producer produced item : 3 Consumer consumed item : 3 Producer produced item : 4 Consumer consumed item : 4

Program Name: Memory Allocation Algorithm (Best Fit, First Fit ,Worst Fit)(

1	**Background & Use-case**	First-Fit Allocation is a memory allocation technique used in operating systems to allocate memory to a process. In First-Fit, the operating system searches through the list of free blocks of memory, starting from the beginning of the list, until it finds a block that is large enough to accommodate the memory request from the process. Once a suitable block is found, the operating system splits the block into two parts: the portion that will be allocated to the process, and the remaining free block. **Advantages** of First-Fit Allocation include its simplicity and efficiency, as the search for a suitable block of memory can be performed quickly and easily. Additionally, First-Fit can also help to minimize memory fragmentation, as it tends to allocate memory in larger blocks. **Disadvantages** of First-Fit Allocation include poor performance in situations where the memory is highly fragmented, as the search for a suitable block of memory can become time-consuming and inefficient. Additionally, First-Fit can also lead to poor memory utilization, as it may allocate larger blocks of memory than are actually needed by a process. Overall, First-Fit Allocation is a widely used memory allocation technique in operating systems, but its effectiveness may vary depending on the specifics of the system and the workload being executed. For both fixed and dynamic memory allocation schemes, the operating system must keep list of each memory location noting which are free and which are busy. Then as new jobs come into the system, the free partitions must be allocated. These partitions may be allocated by 3 ways: 1. First-Fit Memory Allocation 2. Best-Fit Memory Allocation 3. Next-Fit Memory Allocation
2	**Problem Statement**	● Write a Program Code for First Fit Algorithm. ● Write a Program Code for Best Fit Algorithm. ● Write a Program Code for Worst Fit Algorithm.
3	**Research Methodology**	**Write a Program for First Fit Algorithm.**

```
class FirstFit {
    // Method to allocate memory to
    // blocks as per First fit algorithm
    static void firstFit(int blockSize[], int m,
                         int processSize[], int n) {
        // Stores block id of the
        // block allocated to a process
        int allocation[] = new int[n];

        // Initially no block is assigned to any process
        for (int i = 0; i < allocation.length; i++)
            allocation[i] = -1;

        // pick each process and find suitable blocks
        // according to its size and assign to it
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (blockSize[j] >= processSize[i]) {
                    // allocate block j to p[i] process
                    allocation[i] = j;

                    // Reduce available memory in this block
                    blockSize[j] -= processSize[i];
            }
            System.out.println();
        }
    }

    // Driver Code
    public static void main(String[] args) {
        int blockSize[] = {100, 500, 200, 300, 600, 300};
        int processSize[] = {212, 417, 112, 426, 100};
        int m = blockSize.length;
        int n = processSize.length;

        System.out.println("Block Sizes:");
        for (int i = 0; i < m; i++) {
            System.out.println("Block " + (i + 1) + ": " + blockSize[i]);
        }

        System.out.println("\nProcess Sizes:");
        for (int i = 0; i < n; i++) {
            System.out.println("Process " + (i + 1) + ": " + processSize[i]);
        }

        System.out.println();

        firstFit(blockSize, m, processSize, n);
    }

}
```

Write a Program for Best Fit Algorithm.

```
class BestFit {
    // Method to allocate memory to blocks as per Best fit algorithm
    static void bestFit(int blockSize[], int m, int processSize[], int n) {
        // Stores block id of the block allocated to a process
        int allocation[] = new int[n];

        // Initially no block is assigned to any process
        for (int i = 0; i < allocation.length; i++)
            allocation[i] = -1;

        // Iterate over each process and find the best fit block
        for (int i = 0; i < n; i++) {
            // Find the best fit block for the current process
            int bestFitIndex = -1;
            for (int j = 0; j < m; j++) {
                if (blockSize[j] >= processSize[i]) {
                    if (bestFitIndex == -1 || blockSize[j] < blockSize[bestFitIndex]) {
                        bestFitIndex = j;
                    }
                }
            }
            // If a suitable block is found, allocate it to the process
            if (bestFitIndex != -1) {
                allocation[i] = bestFitIndex;
                blockSize[bestFitIndex] -= processSize[i];
            }
        }
        System.out.println("\nProcess No.\tProcess Size\tAllocated Block no.");
        for (int i = 0; i < n; i++) {
            System.out.print(" " + (i + 1) + "\t\t" + processSize[i] + "\t\t");
            if (allocation[i] != -1) {
                System.out.print(allocation[i] + 1 + "\t\t");
            } else {
                System.out.print("Not Allocated");
            }
            System.out.println();
        }
    }

    // Driver Code
    public static void main(String[] args) {
        int blockSize[] = {100, 500, 200, 300, 600, 300};
        int processSize[] = {212, 417, 112, 426, 100};
        int m = blockSize.length;
        int n = processSize.length;

        System.out.println("Block Sizes:");
        for (int i = 0; i < m; i++) {
            System.out.println("Block " + (i + 1) + ": " + blockSize[i]);
        }

        System.out.println("\nProcess Sizes:");
        for (int i = 0; i < n; i++) {
            System.out.println("Process " + (i + 1) + ": " + processSize[i]);
        }

        System.out.println();

        bestFit(blockSize, m, processSize, n);
    }
}
```

Write a Program for Worst Fit Algorithm.

```
class WorstFit {
    // Method to allocate memory to blocks as per Worst fit algorithm
    static void worstFit(int blockSize[], int m, int processSize[], int n) {
        // Stores block id of the block allocated to a process
        int allocation[] = new int[n];
        // Initially no block is assigned to any process
        for (int i = 0; i < allocation.length; i++)
            allocation[i] = -1;
        // Iterate over each process and find the worst fit block
        for (int i = 0; i < n; i++) {
            // Find the worst fit block for the current process
            int worstFitIndex = -1;
            for (int j = 0; j < m; j++) {
                if (blockSize[j] >= processSize[i]) {
                    if (worstFitIndex == -1 || blockSize[j] > blockSize[worstFitIndex])
{
                        worstFitIndex = j;
                    }
                }
            }
            // If a suitable block is found, allocate it to the process
            if (worstFitIndex != -1) {
                allocation[i] = worstFitIndex;
                blockSize[worstFitIndex] -= processSize[i];
            }
        }
        System.out.println("\nProcess No.\tProcess Size\tBlock no.");
        for (int i = 0; i < n; i++) {
            System.out.print(" " + (i + 1) + "\t\t" + processSize[i] + "\t\t");
            if (allocation[i] != -1) {
                System.out.print(allocation[i] + 1 + "\t\t" );
            } else {
                System.out.print("Not Allocated");
            }
            System.out.println();
        }
    }
    // Driver Code
    public static void main(String[] args) {
        int blockSize[] = {10, 30, 50, 70, 90, 120};
        int processSize[] = {20, 40, 60, 80, 100};
        int m = blockSize.length;
        int n = processSize.length;

        System.out.println("Block Sizes:");
        for (int i = 0; i < m; i++) {
            System.out.println("Block " + (i + 1) + ": " + blockSize[i]);
        }

        System.out.println("\nProcess Sizes:");
        for (int i = 0; i < n; i++) {
            System.out.println("Process " + (i + 1) + ": " + processSize[i]);
        }

        System.out.println();

        worstFit(blockSize, m, processSize, n);
```

		} }
4	**Skills Required**	Programming in Java and knowledge of IDE to execute the program
5	**Expected output**	Show the First Fit, Best Fit & Worst Fit Allocation. **Result of First Fit:-** ***(as given below)*** Block Sizes: Block 1: 100 Block 2: 500 Block 3: 200 Block 4: 300 Block 5: 600 Block 6: 300 Process Sizes: Process 1: 212 Process 2: 417 Process 3: 112 Process 4: 426 Process 5: 100 Process No. Process Size Block no 1 212 2 2 417 5 3 112 2 4 426 Not Allocated 5 100 1

Result of Best Fit:- ***(as given below)***

```
Block Sizes:
Block 1: 100
Block 2: 500
Block 3: 200
Block 4: 300
Block 5: 600
Block 6: 300

Process Sizes:
Process 1: 212
Process 2: 417
Process 3: 112
Process 4: 426
Process 5: 100

Process No.     Process Size    Block no.
 1              212             5
 2              417             2
 3              112             5
 4              426             Not Allocated
 5              100             4
```

Result of Worst Fit:- ***(as given below)***

```
Block Sizes:
Block 1: 10
Block 2: 30
Block 3: 50
Block 4: 70
Block 5: 90
Block 6: 120
```

<table>
<tr>
<td></td>
<td></td>
<td><pre>
Process Sizes:
Process 1: 20
Process 2: 40
Process 3: 60
Process 4: 80
Process 5: 100

Process No.   Process Size   Block no.
 1   20   6
 2   40   6
 3   60   5
 4   80   Not Allocated
 5   100   Not Allocated
</pre></td>
</tr>
</table>

Program Name: Optimal Page Replacement Algorithm (Optional)		
1	**Background & Use-case**	OPT is an algorithm for optimal page replacement used in computer operating systems to reduce page faults when managing virtual memory. It is useful when physical memory is constrained and virtual memory contains more pages than can fit in physical memory. It chooses the page that will be inactive for the longest amount of time in the future. It offers the best performance possible in terms of page faults, but it is difficult to implement because it requires knowledge of future page access patterns. Consequently, it is typically employed for benchmarking and performance evaluation rather than in operational systems.
2	**Problem Statement**	To implement Optimal page replacement algorithm

3 **Research Methodology**

The Optimal page replacement algorithm chooses the page that will be inactive for the longest amount of time in the future. It is difficult to implement because the future page access pattern must be known. Here are the instructions for the OPT algorithm in sequential order:

1. When a page error occurs, check if a free page frame is available. If there is a free page frame, load the page into it.
2. Select the page that will not be used for the longest period of time in the future if there is no available page frame. This requires knowledge of the page access pattern in the future, which is generally impossible to predict.
3. If the selected page is dirty, it should be written back to disk before being replaced.
4. Load the requested page into the page frame of choice.
5. Repeat the preceding steps for each page fault.

Notably, it is challenging to implement the OPT algorithm in practice because it requires knowledge of future page access patterns. Consequently, it is commonly used for benchmarking and performance evaluation.

Source Code :

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

class OptimalPR {
    // Function to check whether a page exists
    // in a frame or not
    static boolean search(int key, int[] fr) {
        for (int i = 0; i < fr.length; i++)
            if (fr[i] == key)
                return true;
        return false;
    }

    // Function to find the frame that will not be used
    // recently in future after given index in pg[0..pn-1]
    static int predict(int pg[], int[] fr, int pn, int index) {
```

```
    int res = -1, farthest = index;
    for (int i = 0; i < fr.length; i++) {
        int j;
        for (j = index; j < pn; j++) {
            if (fr[i] == pg[j]) {
                if (j > farthest) {
                    farthest = j;
                    res = i;
                }
                break;
            }
        }

        if (j == pn)
            return i;
    }

    return (res == -1) ? 0 : res;
}
```

4	**Skills Required**	To implement page replacement algorithms in C, a student should follow the following steps: 1. Have a good understanding of C programming language syntax and structure. 2. Understand operating systems concepts such as memory management, page replacement, and paging. 3. Get familiar with algorithms and data structures such as linked lists, queues, and stacks, which are commonly used in page replacement algorithms. 4. Develop proficiency in debugging and testing code to ensure that it works correctly. 5. Gain knowledge of optimization techniques, such as loop unrolling and caching, to help improve code performance. 6. Be aware of version control systems such as Git, which can help keep track of changes to code and collaborate with others.
5	**Expected output**	Page Frames: Page 7: 7 Page 0: 7 Page 1: 7 1 Page 2: 7 1 2 Page 0: 7 1 2 Page 3: 7 1 2 3 Page 0: 1 2 3 Page 4: 4 2 3 Page 2: 4 2 3 Page 3: 4 2 3 Page 0: 4 2 3 Page 3: 4 2 3 Page 2: 4 2 3 No. of hits = 7 No. of misses = 6

Program Name: LRU Page Replacement Algorithm		
1	kground & Use-case	LRU is a popular page replacement algorithm that attempts to keep the most frequently used pages in memory. When a page fault occurs, it replaces the most recently used page, utilising the principle of locality to reduce the number of page faults and improve performance. LRU is widely used in desktop and server operating systems, but it is more difficult to implement than other algorithms and requires the maintenance of data structures, which can be computationally expensive.
2	Problem Statement	To implement LRU page replacement algorithm

3	**Research Methodology**	The problem that LRU (Least Recently Used) page replacement algorithm aims to solve is how to manage memory effectively in an operating system. As programs run, they access different pages of memory, and the operating system needs to keep track of which pages are currently in use and which can be removed from memory to make space for new pages. The goal is to minimize the number of page faults, which occur when a program requests a page that is not currently in memory, by keeping the most frequently used pages in memory.
		The LRU algorithm solves this problem by replacing the page that has been used least recently when a page fault occurs. This allows the most frequently used pages to remain in memory and reduces the likelihood of page faults, improving overall system performance. The challenge is to maintain the data structures required to keep track of the most recently used pages efficiently, which can be computationally expensive.
		Here are the stepwise algorithm for LRU (Least Recently Used) page replacement algorithm:
		1. Initialize a page frame of fixed size.
		2. Initialize a counter for each page frame to keep track of when it was last accessed.
		3. When a new page needs to be added to the page frame, check if it is already present in the page frame. If it is, update its counter to the current time and continue. If it is not, find the page with the lowest counter value, which indicates it was accessed least recently.
		4. Replace the page with the lowest counter value with the new page and update its counter to the current time.
		5. Repeat steps 3 and 4 for every new page that needs to be added to the page frame.
		The basic idea behind LRU is to keep track of when pages were last accessed and replace the least recently used page when a new page needs to be added to the page frame. This allows the most frequently used pages to remain in memory, reducing the number of page faults and improving system performance.
		Source Code :
		`import java.io.*;`
		`import java.util.*;`

```
class LRU {
    static boolean search(int key, int[] fr) {
        for (int i = 0; i < fr.length; i++) {
            if (fr[i] == key) {
                return true;
            }
        }
        return false;
    }

    static int getLRUPage(int[] counter) {
        int minimum = Integer.MAX_VALUE;
        int page = -1;
        for (int i = 0; i < counter.length; i++) {
            if (counter[i] < minimum) {
                minimum = counter[i];
                page = i;
            }
        }
        return page;
    }

    static void lruPage(int pg[], int pn, int fn) {
        int[] fr = new int[fn];
        int[] counter = new int[fn];
        int hit = 0;
        System.out.println("Page Frames:");
        for (int i = 0; i < pn; i++) {
            if (search(pg[i], fr)) {
                hit++;
                System.out.print("Page " + pg[i] + ": ");
                displayFrames(fr);
                continue;
            }

            int emptyIndex = -1;
            for (int j = 0; j < fn; j++) {
                if (fr[j] == 0) {
                    emptyIndex = j;
                    break;
                }
            }

            if (emptyIndex != -1) {
                fr[emptyIndex] = pg[i];
                counter[emptyIndex] = i;
            } else {
                int lruPage = getLRUPage(counter);
                fr[lruPage] = pg[i];
                counter[lruPage] = i;
            }

            System.out.print("Page " + pg[i] + ": ");
            displayFrames(fr);
        }
        System.out.println("No. of hits = " + hit);
        System.out.println("No. of misses = " + (pn - hit));
    }

    static void displayFrames(int[] fr) {
        for (int i = 0; i < fr.length; i++) {
            if (fr[i] != 0) {
                System.out.print(fr[i] + " ");
            }
        }
        System.out.println();
    }

    public static void main(String[] args) {
        int pg[] = {7, 2, 1, 2, 3, 3, 1, 4, 2, 3, 2, 3, 2};
        int pn = pg.length;
        int fn = 5;
        lruPage(pg, pn, fn);
    }
}
```

4	**lls Required**	To implement page replacement algorithms in C, a student should follow the following steps: 1. Have a good understanding of C programming language syntax and structure. 2. Understand operating systems concepts such as memory management, page replacement, and paging. 3. Get familiar with algorithms and data structures such as linked lists, queues, and stacks, which are commonly used in page replacement algorithms. 4. Develop proficiency in debugging and testing code to ensure that it works correctly. 5. Gain knowledge of optimization techniques, such as loop unrolling and caching, to help improve code performance. 6. Be aware of version control systems such as Git, which can help keep track of changes to code and collaborate with others.

5	**Expected Output**	Page Frames: Page 7: 7 Page 2: 7 2 Page 1: 7 2 1 Page 2: 7 2 1 Page 3: 7 2 1 3 Page 3: 7 2 1 3 Page 1: 7 2 1 3 Page 4: 7 2 1 3 4 Page 2: 7 2 1 3 4 Page 3: 7 2 1 3 4 Page 2: 7 2 1 3 4 Page 3: 7 2 1 3 4 Page 2: 7 2 1 3 4 No. of hits = 8 No. of misses = 5

Program Name: Linux Shell Scripting		
1	Background & Use-case	Linux is an open source operating system (OS). An operating system is the software that directly manages a system's hardware and resources, like CPU, memory, and storage. The OS sits between applications and hardware and makes the connections between all of your software and the physical resources that do the work. The Linux command is a utility of the Linux operating system. All basic and advanced tasks can be done by executing commands. The commands are executed on the Linux terminal. The terminal is a command-line interface to interact with the system, which is similar to the command prompt in the Windows OS. Commands in Linux are case-sensitive. **What is Linux Good For?** Linux appears almost everywhere. You can find it on the desktop, server, cloud, mobile devices, supercomputers, and as part of the Internet of Things (IoT), among other platforms. Linux is important because it is customizable, you don't have to jump through someone else's hoop to create an application, it's fast, and it works on older hardware. The Linux learning curve can cause woe initially, but the experience gained in working with Linux translates to all of the platforms it supports, which are many. Linux is actually the basis for other operating systems, like Android, because it does provide so much flexibility. **Why is Linux Important?** The most critical reason to use Linux is that it provides a means of increasing return on investment (ROI) in every environment because you don't necessarily need to pay anything for the operating system and it's potentially possible to use hardware that you already have. Linux is important because it can make the difference between your organization

seeing a profit or experiencing a loss.

Linux is also one of the most successful examples of an open source project. Its code is publicly accessible, anyone can modify and distribute its code, and it’s developed collaboratively with the Linux community. These facets directly contribute to its key benefits, like strong security.

Benefits of Linux

There’s a strong case to make for using Linux in your business, research project, web application, specialized environment, or to address some other practical need. The sections below outline ten benefits of using Linux in your organization.

Strong Security

Linux is more secure than most other operating systems today because:

Linux has numerous distributions and more appear every day. Some well-known distributions include Ubuntu, Debian, and CentOS. All of them are updated regularly, so creating a virus that targets Linux as a whole is nearly impossible because it’s a moving target.

Linux assumes that everyone only has privileges to their own applications and data. When an application is installed and configured by the administrator, the user can’t do anything with it other than use it.

Linux also isn’t the most popular operating system out there when you consider its use in a standalone venue. Virus writers tend to go for the lower hanging fruit: Operating systems that affect a lot of people.

Low Cost

One of the pros of Linux is that the Linux kernel is free and it comes under the open source GNU GPL (General Public License) so you can add whatever you want to it to create a custom configuration. It’s possible to download just about every Linux distribution. For example you can download a form of the Linux kernel with a few add-ons, that is fully functional, out of the box, without cost. Add-ons such as the paid services for Ubuntu, and Red Hat Enterprise, may cost you.

Terminal Support

You don’t need to install special software to contact the backend servers for your project using add-on software with Linux. All you need is the Secure Shell (SSH) utility to access the server securely. In addition, you have access to editors like Emacs, Nano, and Vim that allow you to update config files or hosted Python scripts on the fly. These advantages of using Linux mean that developers spend more time writing and testing code than figuring out some arcane process to complete tasks.

Amazing Driver Support

Linux comes with the drivers you need right out of the box so you won’t waste time searching for the driver disk to use with your device. One of the pros of Linux is that the device support doesn’t end with today’s devices. It’s quite possible to revitalize an older machine to use as a firewall, router, or backup server.

Great Scalability

The ability to scale is essential for any practical software need. Linux provides scalability in several ways:

It runs on so many different platforms that it is hard to find a device that you can’t use Linux on. It scales to the device you need to use, including the ubiquitous robots and industrial computers.
It has a minimal footprint so you can use it on devices of nearly any capability.
It doesn’t bog down under load.
It only uses resources when the application isn’t using them.

Strong Developer Support

Flexibility doesn’t sell a user on an operating system, applications do. Linux supports a plethora of programming languages that developers use in various disciplines. For example, if you’re a data scientist and rely on Python as your programming language, then you run your applications on Linux because they run faster and with fewer resources. Linux supports all of the major languages including C, C++, Java, and JavaScript. If you have any sort of business, research, or other practical software need, Linux runs the code faster with fewer resources.

Available Source Code

Source code for the Linux kernel is readily available, so it’s possible to view whenever needed. This availability has a number of advantages over other operating systems:

There are many people looking for potential flaws in Linux and many coming up with solutions.

It’s possible to track flaws down or modify Linux to meet specific needs.

It’s possible to discover how the operating system works by observing the source code in a debugger.

Compatible Applications

Because of the way the operating system is designed and the fact that the kernel is open source, software developers have better knowledge of how to create compatible, reliable, applications. There are also fewer layers to deal with when working with Linux.

Incredible Server Support

Most developers view Linux as a server operating system and are careful to create applications that run in such an environment. This means that Linux applications often provide robust security and reliability, and still offer an application that runs quickly. Many professionals are attracted to Linux because they don’t have time to keep fixing the operating system and want it to be as invisible as possible. Consequently, you often find Linux used as a workstation operating system for professionals of all stripes.

		Flexible UI Customization When working with most operating systems, the user only has access to one user interface (UI). If the user doesn’t like how that UI works, that’s too bad. With Linux, it’s possible to install any of a number of user interfaces and customize those interfaces as needed. This means extra work at the outset and a higher learning curve, but the user eventually ends up with the right UI to meet specific needs. In short, the UI becomes invisible. Linux provides a powerful command-line interface compared to other operating systems such as Windows and MacOS. We can do basic work and advanced work through its terminal. We can do some basic tasks such as creating a file, deleting a file, moving a file, and more. In addition, we can also perform advanced tasks such as administrative tasks (including package installation, user management), networking tasks (ssh connection), security tasks, and many more.
2	Problem Statement	**Linux Commands** 1. pwd Command The pwd command stands for “print working directory,” and it outputs the absolute path of the directory you’re in. For example, if your username is “Aurobindo” and you’re in your Documents directory, its absolute path would be: /home/Aurobindo/Documents. To use it, simply type pwd in the terminal: `pwd`

```
# My result: /home/kinsta/Documents/linux-commands
```

2. cd Command

The cd command refers to “change directory” and, as its name suggests, switches you to the directory you’re trying to access.

For instance, if you’re inside your Documents directory and you’re trying to access one of its subfolders called CSE, you can enter it by typing:

```
cd CSE
```

You can also supply the absolute path of the folder:

```
cd /home/Aurobindo/CSE
```

		some tricks with the cd command Go to the home folder: **cd** Move a level up: **cd ..** Return to the previous directory: **cd -** 3. **mkdir Command:** The mkdir command is used to create a new directory under any directory. **Syntax: mkdir <directory name>** 4. **rmdir Command:** The rmdir command is used to delete a directory. Syntax: **rmdir <directory name>** 5. **ls Command:** The ls command is used to display a list of content of a directory. Syntax: ls **6. touch Command:** The touch command is used to create empty files. We can create multiple empty files by executing it once. Syntax: 1. touch <file name> 2. touch <file1> <file2> .... 7. **cat Command:** The cat command is a multi-purpose utility in the Linux system. It can

be used to create a file, display content of the file, copy the content of one file to another file, and more.

Syntax: cat [OPTION]... [FILE]..

To create a file, execute it as follows:

1. cat > <file name>
2. // Enter file content

Press "CTRL+ D" keys to save the file. To display the content of the file, execute it as follows:

1. cat <file name>

8. rm Command: The rm command is used to remove a file.

   Syntax:

   rm <file name>

9. cp Command: The cp command is used to copy a file or directory.

Syntax:To copy in the same directory:

1. cp <existing file name> <new file name>
2. cp file_to_copy.txt new_file.txt

3. You can also copy entire directories by using the recursive flag:

4.
```
cp -r dir_to_copy/ new_copy_dir/
```

5. Remember that in Linux, folders end with a forward slash (/).

10. rm Command: rm command to remove files and directories.

```
rm file_to_copy.txt
```

```
rm -r dir_to_remove/
```

to remove a directory with content inside of it, you need to use the force (-f) and recursive flags:

```
rm -rf dir_with_content_to_remove/
```

11. mv Command: mv command to move (or rename) files and directories through your file system.

```
mv source_file destination_folder/
```

		`mv command_list.txt commands/`
		12. mkdir Command: To create folders in the shell, you use the mkdir command. Just specify the new folder's name, ensure it doesn't exist, and you're ready to go.
		`mkdir images/`
		To create subdirectories with a simple command, use the parent (-p) flag:
		`mkdir -p movies/2004/`
		13. man Command
		It displays the manual page of any other command (as long as it has one).
		To see the manual page of the mkdir command, type:

		man mkdir You could even refer to the man manual page: man man 14. sdf 15.
3	**Research Methodology**	Experiment all commands in linux terminal
4	**Skills Required**	Shell Scripting
5	**Expected output**	Linux commands implementation

Program Name: Any Subdivisions for Shell Scripting		
1	**Background & Use-case**	Shell scripting
2	**Problem Statement**	To find a number is prime or not

3	**Research Methodology**	`#!/bin/bash`
		`echo -e "Enter Number : \c"`
		`read n`
		`for((i=2; i<=$n/2; i++))`
		`do`
		`  ans=$(( n%i ))`
		`  if [ $ ans -eq 0  ]`
		`  then`
		`      echo "$n is not a prime number."`
		`      exit 0`

		fi
		done
		echo "$n is a prime number."

4	**Skills Required**	scripting in Linux
5	**Expected output**	Enter the number 3 3 is a prime number

Program Name: Any Subdivisions for Shell Scripting		
1	**Background & Use-case**	Shell scripting
2	**Problem Statement**	To find the reverse of a number and reverse of a string.
3	**Research Methodology**	#!/bin/bash echo "Enter a String" read input reverse=""

```
len=${#input}

for (( i=$len-1; i>=0; i-- ))

do

      reverse="$reverse${input:$i:1}"

done

if [ $input == $reverse ]

then

    echo "$input is palindrome"

else

    echo "$input is not palindrome"

fi



-----------------------------------------

echo "Enter the number"

read num
```

```
# Storing the remainder
s=0

# Store number in reverse order
rev=""

# Store original number in another variable
temp=$num

while [ $num -gt 0 ]
do
    # Get Remainder
    s=$(( $num % 10 ))

    # Get next digit
    num=$(( $num / 10 ))
     # Store previous number and
```

		# current digit in reverse
		    rev=$(echo ${rev}${s})
		done
		if [$temp -eq $rev];
		then
		    echo "Number is palindrome"
		else
		    echo "Number is NOT palindrome"
		fi

4	**Skills Required**	scripting in Linux
5	**Expected output**	Enter the string: madam madam is palindrome Enter the number: 133 Number is NOT palindrome