OPERATING SYSTEM LAB

1. WAP a program with the help of shell program to check the eligibility of voting according to your age.

SOURCE CODE:

```
#!/bin/bash

read -p "Enter your age: " age

if ((age >= 18)); then
  echo "You are eligible to vote"
else
  echo "You are not eligible to vote"
fi
```

2. Write the functions of all shell commands:

1). Displaying the file contents on the terminal:

- cat: It is generally used to concatenate the files. It gives the output on the standard output.
- more: It is a filter for paging through text one screenful at a time.
- less: It is used to viewing the files instead of opening the file.Similar to *more* command but it allows backward as well as forward movement.
- head : Used to print the first N lines of a file. It accepts N as input and the default value of N is 10.
- tail : Used to print the last N-1 lines of a file. It accepts N as input and the default value of N is 10.

2). File and Directory Manipulation Commands:

- mkdir : Used to create a directory if not already exist. It accepts the directory name as an input parameter.
- cp : This command will copy the files and directories from the source path to the destination path. It can copy a file/directory with the new name to the destination path. It accepts the source file/directory and destination file/directory.
- mv : Used to move the files or directories. This command's working is almost similar to *cp* command but it deletes a copy of the file or directory from the source path.
- rm : Used to remove files or directories.
- touch : Used to create or update a file.

3). Extract, sort, and filter data Commands:

- grep : This command is used to search for the specified text in a file.
- sort : This command is used to sort the contents of files.
- wc : Used to count the number of characters, words in a file.
- cut : Used to cut a specified part of a file.

4). Basic Terminal Navigation Commands:

- ls : To get the list of all the files or folders.
- cd: Used to change the directory.
- du: Show disk usage.
- pwd: Show the present working directory.
- man: Used to show the manual of any command present in Linux.
- rmdir: It is used to delete a directory if it is empty.
- ln **file1 file2**: Creates a physical link.
- ln **-s file1 file2**: Creates a symbolic link.
- locate: It is used to locate a file in Linux System
- echo: This command helps us move some data, usually text into a file.
- df: It is used to see the available disk space in each of the partitions in your system.
- tar: Used to work with tarballs (or files compressed in a tarball archive)

5). File Permissions Commands: The *chmod* and *chown* commands are used to control access to files in UNIX and Linux systems.

- chown : Used to change the owner of the file.
- chgrp : Used to change the group owner of the file.
- chmod : Used to modify the access/permission of a user.

3. WAP to print the prime number within the given range.

SOURCE CODE:

```
#!/bin/bash

is_prime() {
  local number=$1
  if (($number < 2)); then
    return 1
  fi
  for ((i = 2; i <= $number / 2; i++)); do
    if (($number % i == 0)); then
      return 1
    fi
  done
  return 0
}

start=1
end=100

for ((number = start; number <= end; number++)); do
  if is_prime $number; then
    echo $number
  fi
done
```

4. WAP to print the array elements in Linux:

SOURCE CODE:

```
#!/bin/bash

arr=(susmita aritrika 22 22.7)
echo ${arr[@]}
echo ${arr[@]:0}
echo ${arr[@]:1}
for i in "${arr[@]}"
do
echo "$i"
done
```

5. Implement the FCFS scheduling algorithm without arrival time with the help of C.

SOURCE CODE:

```
#include <stdio.h>
int main()
{
  int pid[15];int bt[15];int n;
  printf("Enter the number of processes: ");
  scanf("%d",&n);
  printf("Enter process id of all the processes: ");
  for(int i=0;i<n;i++)
  {
    scanf("%d",&pid[i]);
  }
  printf("Enter burst time of all the processes: ");
  for(int i=0;i<n;i++)
  {
    scanf("%d",&bt[i]);
  }
  int i, wt[n];
  wt[0]=0;
```

```
    //for calculating waiting time of each process
    for(i=1; i<n; i++)
    {
       wt[i]= bt[i-1]+ wt[i-1];
    }
    printf("Process ID    Burst Time    Waiting Time    TurnAround Time\n");
    float twt=0.0;
    float tat= 0.0;
    for(i=0; i<n; i++)
    {
       printf("%d\t\t\t\t", pid[i]);
       printf("%d\t\t\t\t", bt[i]);
       printf("%d\t\t\t\t", wt[i]);
       //calculating and printing turnaround time of each process
       printf("%d\t\t", bt[i]+wt[i]);
       printf("\n");
       //for calculating total waiting time
       twt += wt[i];
       //for calculating total turnaround time
       tat += (wt[i]+bt[i]);
    }
    float att,awt;
    //for calculating average waiting time
    awt = twt/n;
    //for calculating average turnaround time
    att = tat/n;
    printf("Avg. waiting time= %f\n",awt);
    printf("Avg. turnaround time= %f",att);
}
```

6. Implement the FCFS scheduling algorithm with arrival time with the help of C.

SOURCE CODE:

```
#include<stdio.h>

int main(){

    int bt[10]={0},at[10]={0},tat[10]={0},wt[10]={0},ct[10]={0};
    int n,sum=0;
    float totalTAT=0,totalWT=0;

    printf("Enter number of processes        ");
    scanf("%d",&n);

    printf("Enter arrival time and burst time for each process\n\n");

    for(int i=0;i<n;i++)
    {

            printf("Arrival time of process[%d]        ",i+1);
            scanf("%d",&at[i]);

            printf("Burst time of process[%d]          ",i+1);
            scanf("%d",&bt[i]);

            printf("\n");
    }

    //calculate completion time of processes

    for(int j=0;j<n;j++)
    {
            sum+=bt[j];
            ct[j]+=sum;
    }
```

```
	//calculate turnaround time and waiting times

	for(int k=0;k<n;k++)
	{
		tat[k]=ct[k]-at[k];
		totalTAT+=tat[k];
	}


	for(int k=0;k<n;k++)
	{
		wt[k]=tat[k]-bt[k];
		totalWT+=wt[k];
	}

	printf("Solution: \n\n");
	printf("P#\t AT\t BT\t CT\t TAT\t WT\t\n\n");

	for(int i=0;i<n;i++)
	{
		printf("P%d\t %d\t %d\t %d\t %d\t %d\n",i+1,at[i],bt[i],ct[i],tat[i],wt[i]);
	}

	printf("\n\nAverage Turnaround Time = %f\n",totalTAT/n);
	printf("Average WT = %f\n\n",totalWT/n);

	return 0;
}
```

7. SJF (NON-PREEMPTIVE):

SOURCE CODE:

```
#include<stdio.h>

int main() {
  int time, burst_time[10], at[10], sum_burst_time = 0, smallest, n, i;
  int sumt = 0, sumw = 0;
  printf("enter the no of processes : ");
  scanf("%d", & n);
  for (i = 0; i < n; i++) {
    printf("the arrival time for process P%d : ", i + 1);
    scanf("%d", & at[i]);
    printf("the burst time for process P%d : ", i + 1);
    scanf("%d", & burst_time[i]);
    sum_burst_time += burst_time[i];
  }
  burst_time[9] = 9999;
  for (time = 0; time < sum_burst_time;) {
    smallest = 9;
    for (i = 0; i < n; i++) {
      if (at[i] <= time && burst_time[i] > 0 && burst_time[i] < burst_time[smallest])
        smallest = i;
    }
    printf("P[%d]\t|\t%d\t|\t%d\n", smallest + 1, time + burst_time[smallest] - at[smallest], time -
at[smallest]);
    sumt += time + burst_time[smallest] - at[smallest];
    sumw += time - at[smallest];
    time += burst_time[smallest];
    burst_time[smallest] = 0;
  }
```

```
 printf("\n\n average waiting time = %f", sumw * 1.0 / n);
 printf("\n\n average turnaround time = %f", sumt * 1.0 / n);
 return 0;
}
```

8. SJF (PREEMPTIVE):

SOURCE CODE:

```
#include<stdio.h>
 int main()
{
   int burst_time[20],p[20],waiting_time[20],tat[20],i,j,n,total=0,pos,temp;
   float avg_waiting_time,avg_tat;
   printf("please enter number of process: ");
   scanf("%d",&n);
   printf("\n enter the Burst Time:\n");
   for(i=0;i<n;i++)
   {
      printf("p%d:",i+1);
      scanf("%d",&burst_time[i]);
      p[i]=i+1;
   }
  // from here, burst times sorted
   for(i=0;i<n;i++)
   {
      pos=i;
      for(j=i+1;j<n;j++)
      {
         if(burst_time[j]<burst_time[pos])
            pos=j;
      }
      temp=burst_time[i];
      burst_time[i]=burst_time[pos];
      burst_time[pos]=temp;
      temp=p[i];
      p[i]=p[pos];
      p[pos]=temp;
   }
   waiting_time[0]=0;
   for(i=1;i<n;i++)
   {
      waiting_time[i]=0;
      for(j=0;j<i;j++)
         waiting_time[i]+=burst_time[j];
      total+=waiting_time[i];
   }
   avg_waiting_time=(float)total/n;
   total=0;
   printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
   for(i=0;i<n;i++)
   {
      tat[i]=burst_time[i]+waiting_time[i];
      total+=tat[i];
      printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],burst_time[i],waiting_time[i],tat[i]);
   }
   avg_tat=(float)total/n;
   printf("\n\n the average Waiting Time=%f",avg_waiting_time);
   printf("\n  the average Turnaround Time=%f\n",avg_tat);
}
```

9. PRIORITY SCHEDULING USING C: SOURCE CODE:

```c
#include<stdio.h>

int main()
{
   int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
   printf("Enter Total Number of Process:");
   scanf("%d",&n);

   printf("\nEnter Burst Time and Priority\n");
   for(i=0;i<n;i++)
   {
      printf("\nP[%d]\n",i+1);
      printf("Burst Time:");
      scanf("%d",&bt[i]);
      printf("Priority:");
      scanf("%d",&pr[i]);
      p[i]=i+1;
   }

   for(i=0;i<n;i++)
   {
      pos=i;
      for(j=i+1;j<n;j++)
      {
         if(pr[j]<pr[pos])
            pos=j;
      }

      temp=pr[i];
      pr[i]=pr[pos];
      pr[pos]=temp;

      temp=bt[i];
      bt[i]=bt[pos];
      bt[pos]=temp;

      temp=p[i];
      p[i]=p[pos];
      p[pos]=temp;
   }

   wt[0]=0;
   for(i=1;i<n;i++)
   {
      wt[i]=0;
      for(j=0;j<i;j++)
         wt[i]+=bt[j];

      total+=wt[i];
   }

   avg_wt=total/n;
   total=0;

   printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
   for(i=0;i<n;i++)
   {
      tat[i]=bt[i]+wt[i];
      total+=tat[i];
      printf("\nP[%d]\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
   }
```

```
    avg_tat=total/n;
    printf("\n\nAverage Waiting Time=%d",avg_wt);
    printf("\nAverage Turnaround Time=%d\n",avg_tat);

  return 0;
  }
```

10. ROUND ROBIN USING C:

SOURCE CODE:

```
#include<stdio.h>
void main()
{
    int n,i,qt,count=0,temp,sq=0,bt[10],wt[10],tat[10],rem_bt[10];
    float awt=0,atat=0;
    printf("Enter No of process ");
    scanf("%d",&n);
    printf("Enter The Burst Time");
    for(i=0;i<n;i++)
{
    scanf("%d",&bt[i]);
    rem_bt[i]=bt[i];
}
printf("Enter the Quantam Time");
scanf("%d",&qt);
while (1)
{
    for(i=0,count=0;i<n;i++){
temp=qt;
if(rem_bt[i]==0)
{
    count ++;
    continue;
}
if(rem_bt[i]>qt){
    rem_bt[i]=rem_bt[i]-qt;
}
else if(rem_bt[i]>=0){
    temp=rem_bt[i];
    rem_bt[i]=0;
}
sq=sq+temp;
tat[i]=sq;
    }
    if(n==count)
    break;
}
printf("\nProcess\tBurst Time\tTurn Around Time\t Waiting Time\n");
for(i=0;i<n;i++)
{
wt[i]=tat[i]-bt[i];
awt=awt+wt[i];
atat=atat+tat[i];
printf("\n %d \t %d \t\t %d \t %d \t",i+1,bt[i],tat[i],wt[i]);

}
awt=awt/n;
atat=atat/n;
printf("\nAvarage Waiting Time=%f \n",awt);
printf("\nAvarage Turn Around Time=%f",atat);

}
```

11. BANKER'S ALGORITHM USING C: SOURCE CODE:

```
// Banker's Algorithm
#include <stdio.h>
int main()
{
   // P0, P1, P2, P3, P4 are the Process names here

   int n, m, i, j, k;
   n = 5; // Number of processes
   m = 3; // Number of resources
   int alloc[5][3] = { { 0, 1, 0 }, // P0    // Allocation Matrix
                       { 2, 0, 0 }, // P1
                       { 3, 0, 2 }, // P2
                       { 2, 1, 1 }, // P3
                       { 0, 0, 2 } }; // P4

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

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

   int f[n], ans[n], ind = 0;
   for (k = 0; k < n; k++) {
      f[k] = 0;
   }
   int need[n][m];
   for (i = 0; i < n; i++) {
      for (j = 0; j < m; j++)
         need[i][j] = max[i][j] - alloc[i][j];
   }
   int y = 0;
   for (k = 0; k < 5; k++) {
      for (i = 0; i < n; i++) {
         if (f[i] == 0) {

            int flag = 0;
            for (j = 0; j < m; j++) {
               if (need[i][j] > avail[j]){
                  flag = 1;
                   break;
               }
            }

            if (flag == 0) {
               ans[ind++] = i;
               for (y = 0; y < m; y++)
                  avail[y] += alloc[i][y];
               f[i] = 1;
            }
         }
      }
   }

    int flag = 1;

    for(int i=0;i<n;i++)
   {
    if(f[i]==0)
    {
      flag=0;
```

```
      printf("The following system is not safe");
      break;
    }
  }

  if(flag==1)
  {
   printf("Following is the SAFE Sequence\n");
   for (i = 0; i < n - 1; i++)
    printf(" P%d ->", ans[i]);
   printf(" P%d", ans[n - 1]);
  }


  return (0);
}
```

12. FIFO PAGE REPLACEMENT USING C:

SOURCE CODE:

```
#include<stdio.h>
int main()
{
int i,j,n,a[50],frame[10],no,k,avail,count=0;
        printf("\n ENTER THE NUMBER OF PAGES:\n");
scanf("%d",&n);
        printf("\n ENTER THE PAGE NUMBER :\n");
        for(i=1;i<=n;i++)
        scanf("%d",&a[i]);
        printf("\n ENTER THE NUMBER OF FRAMES :");
        scanf("%d",&no);
for(i=0;i<no;i++)
        frame[i]= -1;
                j=0;
                printf("\tref string\t page frames\n");
for(i=1;i<=n;i++)
 {
     printf("%d\t\t",a[i]);
 avail=0;
         for(k=0;k<no;k++)
if(frame[k]==a[i])
          avail=1;
      if (avail==0)
  {
       frame[j]=a[i];
       j=(j+1)%no;
       count++;
            for(k=0;k<no;k++)
      printf("%d\t",frame[k]);
}
printf("\n");
}
                printf("Page Fault Is %d",count);
                return 0;
}
```

13. LRU PAGE REPLACEMENT:

SOURCE CODE:

```
#include<stdio.h>
int main()
{
int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];
```

```
printf("Enter no of pages:");
scanf("%d",&n);
printf("Enter the reference string:");
for(i=0;i<n;i++)
        scanf("%d",&p[i]);
printf("Enter no of frames:");
scanf("%d",&f);
q[k]=p[k];
printf("\n\t%d\n",q[k]);
c++;
k++;
for(i=1;i<n;i++)
        {
                c1=0;
                for(j=0;j<f;j++)
                {
                        if(p[i]!=q[j])
                        c1++;
                }
                if(c1==f)
                {
                        c++;
                        if(k<f)
                        {
                                q[k]=p[i];
                                k++;
                                for(j=0;j<k;j++)
                                printf("\t%d",q[j]);
                                printf("\n");
                        }
                        else
                        {
                                for(r=0;r<f;r++)
                                {
                                        c2[r]=0;
                                        for(j=i-1;j<n;j--)
                                        {
                                        if(q[r]!=p[j])
                                        c2[r]++;
                                        else
                                        break;
                                }
                        }
                        for(r=0;r<f;r++)
                         b[r]=c2[r];
                        for(r=0;r<f;r++)
                        {
                                for(j=r;j<f;j++)
                                {
                                        if(b[r]<b[j])
                                        {
                                                t=b[r];
                                                b[r]=b[j];
                                                b[j]=t;
                                        }
                                }
                        }
                        for(r=0;r<f;r++)
                        {
                                if(c2[r]==b[0])
                                q[r]=p[i];
                                printf("\t%d",q[r]);
                        }
```

```
                    printf("\n");
                }}}
printf("\nThe no of page faults is %d",c);
}
```

14. OPTIMAL PAGE REPLACEMENT

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
int main()
{
  int fr[5],i,j,k,t[5],p=1,flag=0,page[25],psz,nf,t1,u[5];
  printf("enter the number of frames:");
  scanf("%d",&nf);
  printf("\n enter the page size");
  scanf("%d",&psz);

  printf("\nenter the page sequence:");
  for(i=1; i<=psz; i++)
    scanf("%d",&page[i]);

  for(i=1; i<=nf; i++)
    fr[i]=-1;
  for(i=1; i<=psz; i++)
  {
    if(full(fr,nf)==1)
      break;
    else
    {
      flag=0;
      for(j=1; j<=nf; j++)
      {
        if(page[i]==fr[j])
        {
          flag=1;
          printf("        \t%d:\t",page[i]);
          break;
        }
      }
      if(flag==0)
      {
        fr[p]=page[i];
        printf("        \t%d:\t",page[i]);
        p++;
      }

      for(j=1; j<=nf; j++)
        printf(" %d  ",fr[j]);
      printf("\n");
    }
  }
  p=0;
  for(; i<=psz; i++)
  {
    flag=0;
    for(j=1; j<=nf; j++)
    {
      if(page[i]==fr[j])
      {
        flag=1;
        break;
```

```
            }
        }
        if(flag==0)
        {
            p++;
            for(j=1; j<=nf; j++)
            {
                for(k=i+1; k<=psz; k++)
                {
                    if(fr[j]==page[k])
                    {
                        u[j]=k;
                        break;
                    }
                    else
                        u[j]=21;
                }
            }
            for(j=1; j<=nf; j++)
                t[j]=u[j];
            for(j=1; j<=nf; j++)
            {
                for(k=j+1; k<=nf; k++)
                {
                    if(t[j]<t[k])
                    {
                        t1=t[j];
                        t[j]=t[k];
                        t[k]=t1;
                    }
                }
            }
            for(j=1; j<=nf; j++)
            {
                if(t[1]==u[j])
                {
                    fr[j]=page[i];
                    u[j]=i;
                }
            }
            printf("page fault\t");
        }
        else
            printf("         \t");
        printf("%d:\t",page[i]);
        for(j=1; j<=nf; j++)
            printf(" %d  ",fr[j]);
        printf("\n");
    }
    printf("\ntotal page faults:  %d",p+3);
}
int full(int a[],int n)
{
    int k;
    for(k=1; k<=n; k++)
    {
        if(a[k]==-1)
            return 0;
    }
    return 1;
}
```