**Linux Terminal Commands**

**1) mkdir**

mkdir (or make directory) is used to create a new directory.

**Syntax:** mkdir <directory name>

**2) cd**

cd (or change directory) is used to navigate to different directories.

**Syntax:** cd <directory path>

To Move out of a directory use the following commands.

* cd ../-Use to move out of sub directory.
* cd \-Use to move out of multiple sub directories.

**3) touch**

It is used to create an empty file.

**Syntax:** touch <filename + extension>

**4) cat**

It can be used to edit, view or merge files.

**Syntax:**

(i) cat > <filename> - allows editing the file, ctrl+D is used to save and exit

(ii) cat <filename> - displays the contents of the file

(iii) cat <file1> <file2> > <file3> - merges the content of file1 & file2 in file3

**5) ls**

It is used to display the contents of a directory.

**Syntax:**

ls - will display all types of files and directories

**(6) ls \***

Will only display files with specified extension

**Syntax:** ls \*.<extension name>

**7) pwd**

It displays the complete path of the current working directory.

**Syntax:** pwd

**8) cp**

It is used to copy a file from one directory to another.

**Syntax:** cp <file path> <location where the file must be copied to>

**9) mv**

It is used to move a file from one directory to another. The file is removed from the original location.

**Syntax:** mv <file path> <location where the file must be moved to>

**10) head**

Displays the first 10 lines of content of a file.

**Syntax:** head <filename>

**11) tail**

Displays the last 10 lines of content of a file.

**Syntax:** tail <filename>

**12) tac**

Displays the content of a file in reverse order.

**Syntax:** tac <filename>

**13) more**

Displays the content of large files.

**Syntax:** More <filename>

**14) vi**

Used to edit a file. After entering the command, and hitting enter, one must input ‘i’ to open insert mode. After making the changes, :wq must be entered.

**Syntax:** vi <filename>

The following are basic commands used in a Vim Text Editor.

* :q- To exit Vim.
* :! - To override previous commands.
* :wq - To write and quit.
* :w- to save

**15) id**

It is used to display id of user/group.

**Syntax:** id

**16) clear**

It is used to clear the screen.

**Syntax:** clear

**17)** **vi**

It is a text editor used to write programs of text.

**18) grep**

Filter to search the given pattern in the file content.

**Syntax:** grep <word> <filename.txt>

**19) diff**

It is used to compare the content between two files.

**Syntax:** diff <filename1.txt> <filename2.txt>

**20) ping**

It is used to check the connectivity status of the user.

**Syntax:** ping <searchengine>

**21) history**

It is used to review all the commands that you have entered.

**Syntax:** history

**22) hostname**

It is used to display hostname of the user.

**Syntax:** hostname

**23) hostname-i**

It is used to display hostname ip .

**Syntax:** hostname -i

**24) chmod**

It is used to change the user/group permissions to access a file.

**Syntax:**

* For read only : chmod u=r
* For write only: chmod u=w

**25) nl**

It is used to display the line numbers.

**Syntax:** nl <filename.txt>

**26) wc**

It is used to display the no. of lines, words and characters available in the file content.

**Syntax:** wc <filename.txt>

**27) uniq**

It is used to remove duplicate file contents but it can remove only continuous duplicates.

**Syntax:** uniq

**28) ps**

It prints the report of the current processes.

**Syntax:** ps

**29) rmdir**

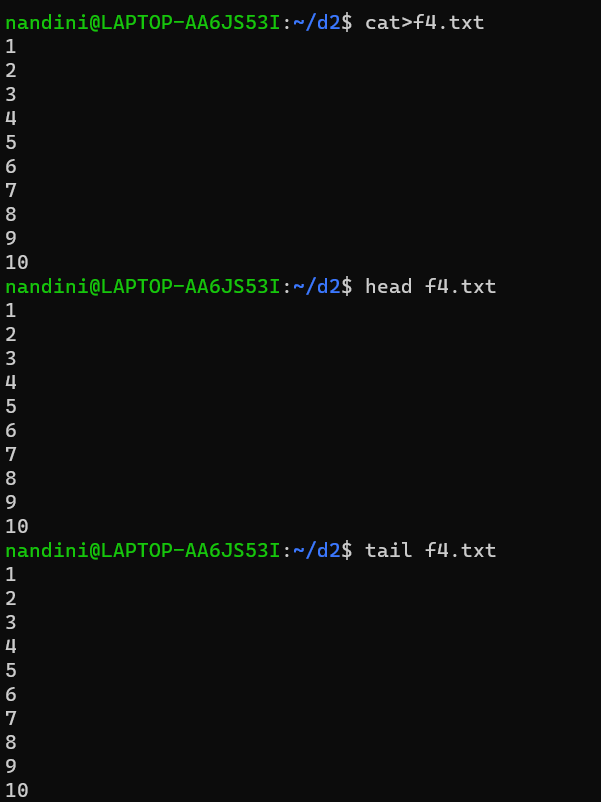
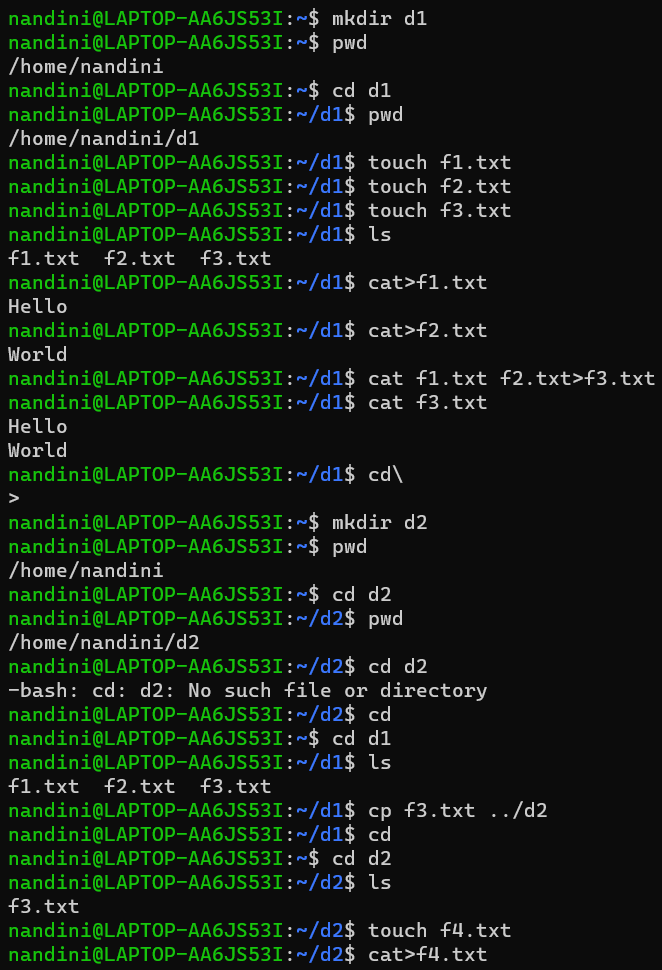
It removes the specified directory(directory should be empty).

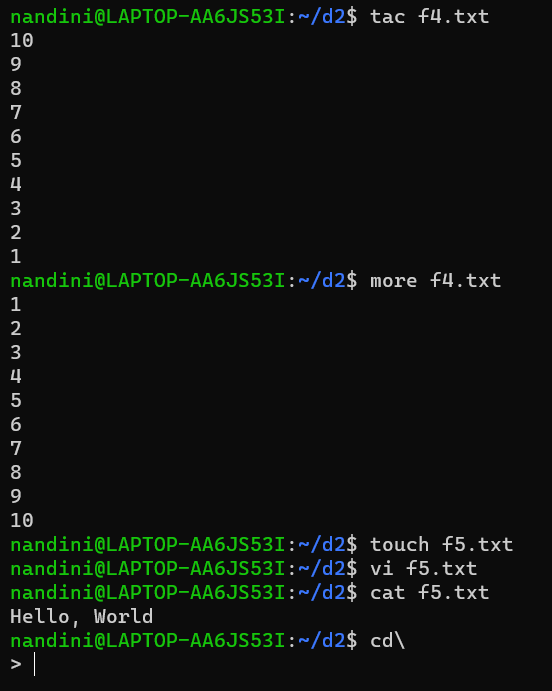
**Syntax:** rmdir <directory name>

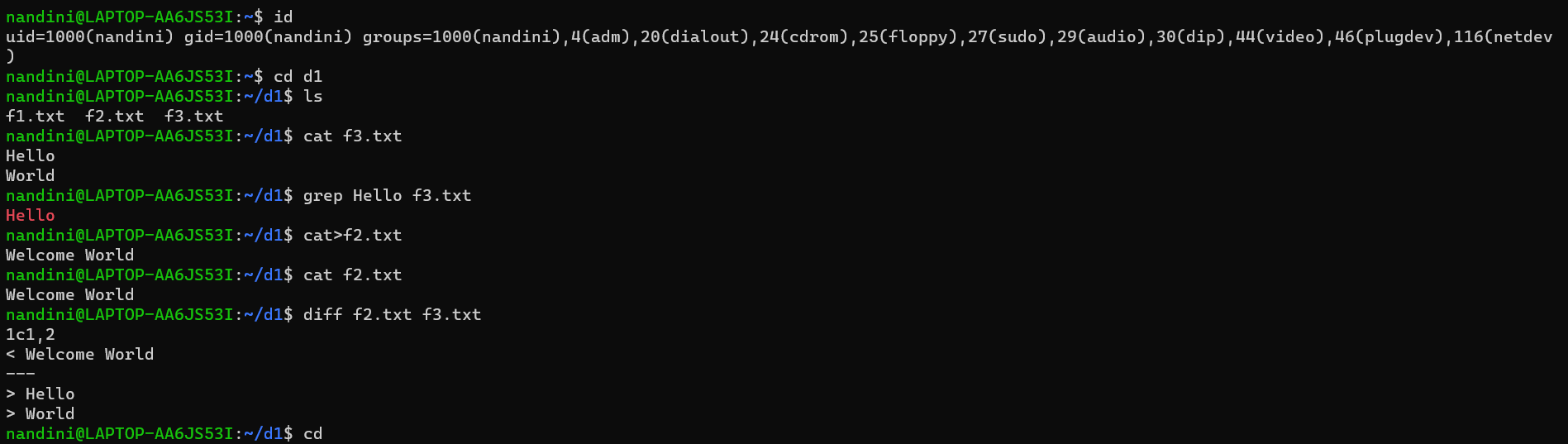
**30) rm**

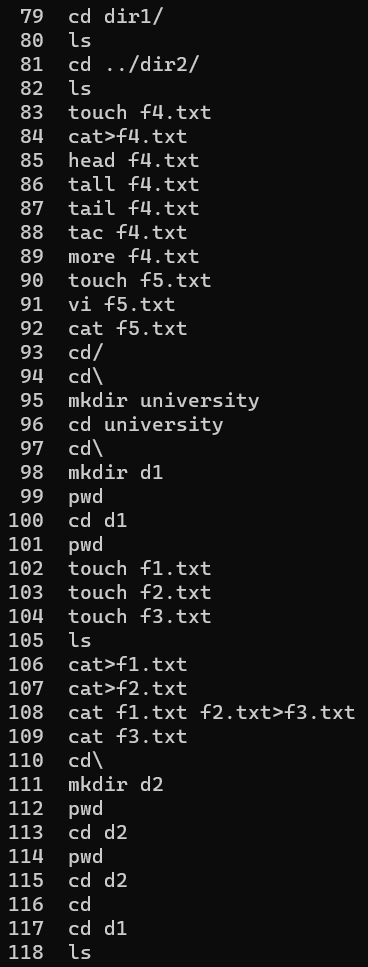
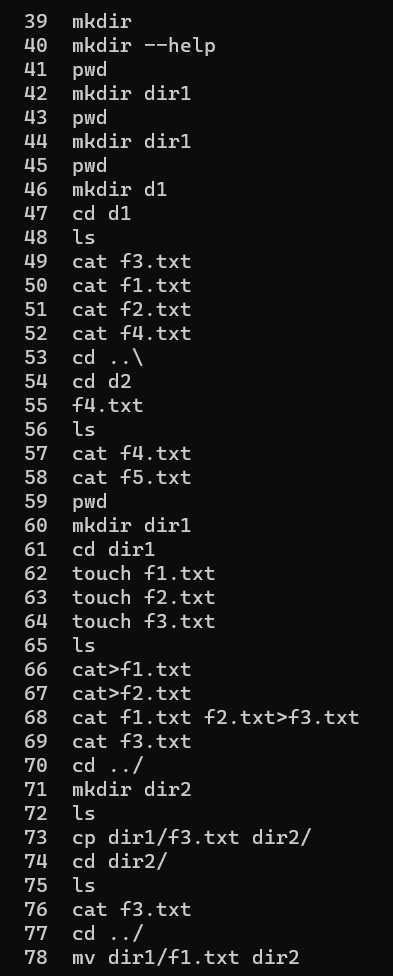
It removes the file .

**Syntax:** rm <filename>



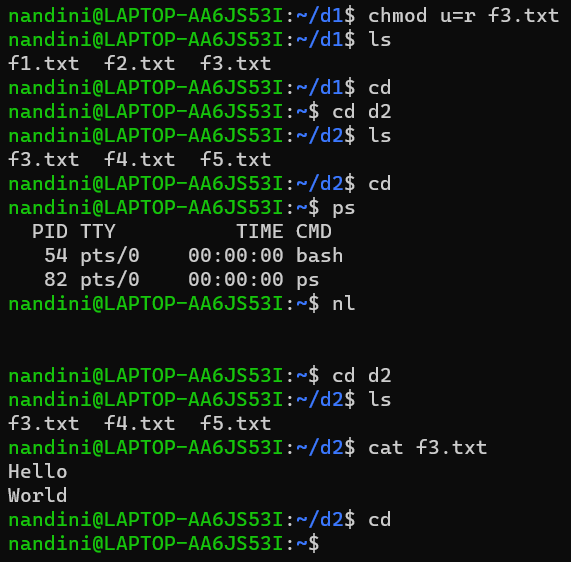




  Text

Description automatically generated with medium confidenceText

Description automatically generated



**Linux Shell Commands**

1. **$cat /etc/shells**

Used to find the available shells in your system.

1. **$ chmod +x file\_name.txt**

Used to execute the permission for a script.

1. **. ./file\_name.txt**

Used to execute a file.

1. **Echo**

Used to print the content.

* echo “My name is Nandini Sain”
* echo -e ”\033[0;31m Message” : - Used to change the colour of the text.
* name=”nandini sain”

age=”19”

echo “ my name is {$name} and my age is{$age}.

1. **which bash**

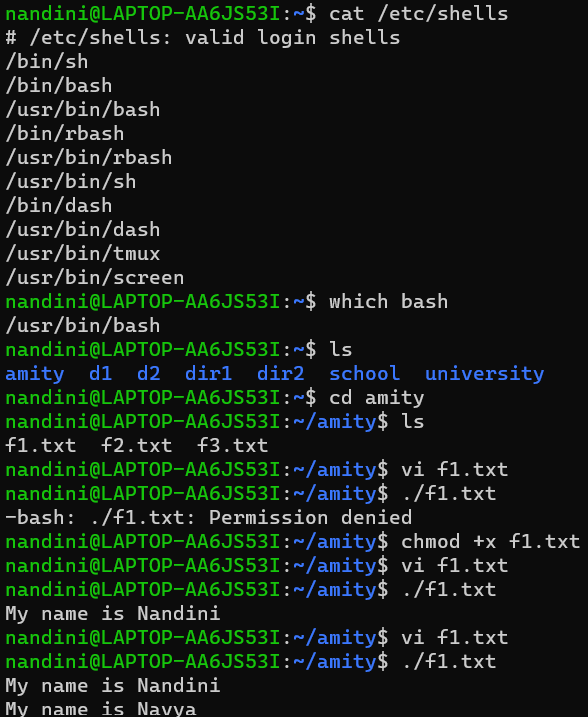
It tells the user to tell the path.

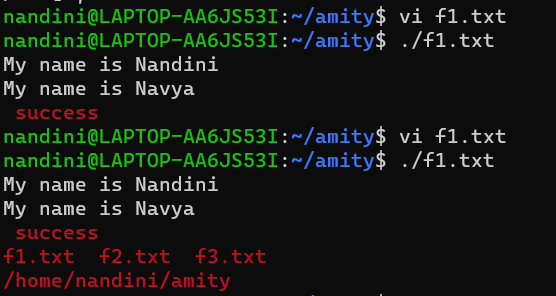
1. **env**

It will display all the system commands.

Some of them are: -

* echo $SHELL
* echo $HOME
* echo $OSTYPE
* echo $PATH
* echo $$: - tells the process id.
* echo $PPID: -tells the parent process id.
* echo $PWD





A picture containing shape

Description automatically generated

**Linux Shell Programs**

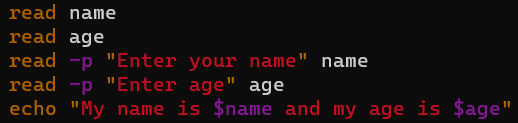
1. Write a shell script for addition of two numbers.

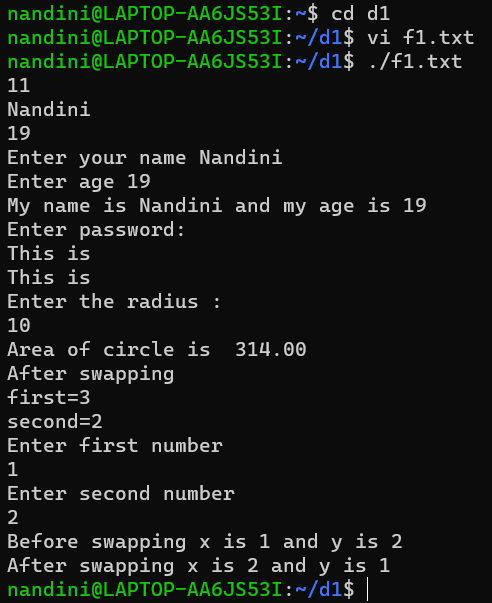
Text

Description automatically generated



1. Write a shell script to print the name and the age.





1. Write a shell script to enter the password and input two values.

Text

Description automatically generated

Text

Description automatically generated

1. Write a shell script for calculating the area of circle. Radius is to be entered by user.

Text

Description automatically generated

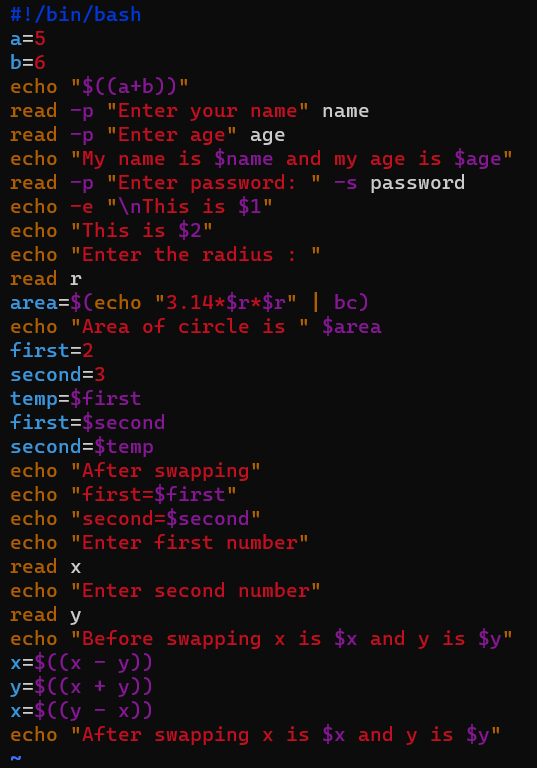
Graphical user interface, text, application

Description automatically generated

1. Write a shell script for swapping two numbers:

(i) Using third variable

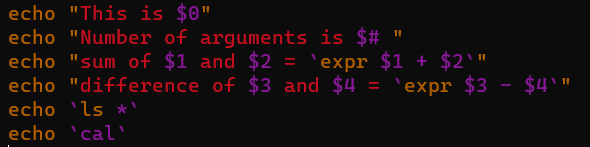
(ii) Using without third variable



Text

Description automatically generated

1. Write the following for commands.
2. Shell Script Name ($0),
3. No. of Arguments (i.e., $#),
4. And actual argument (i.e. $1, $2 etc.)
5. $ sum (11, 20)
6. $ math (4 – 7)
7. $ Ls \*
8. $ cal



Text

Description automatically generated with medium confidence

**Shell Script**

Some Basic Commands Used in GNU Linux Commands are-

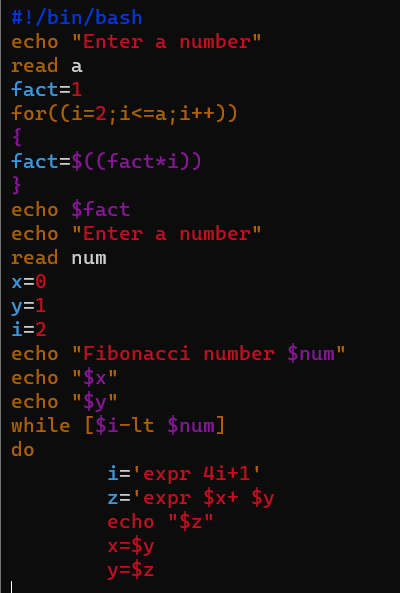
Text

Description automatically generated

Text

Description automatically generated

1. Write a shell script to find the factorial of any number.



Text

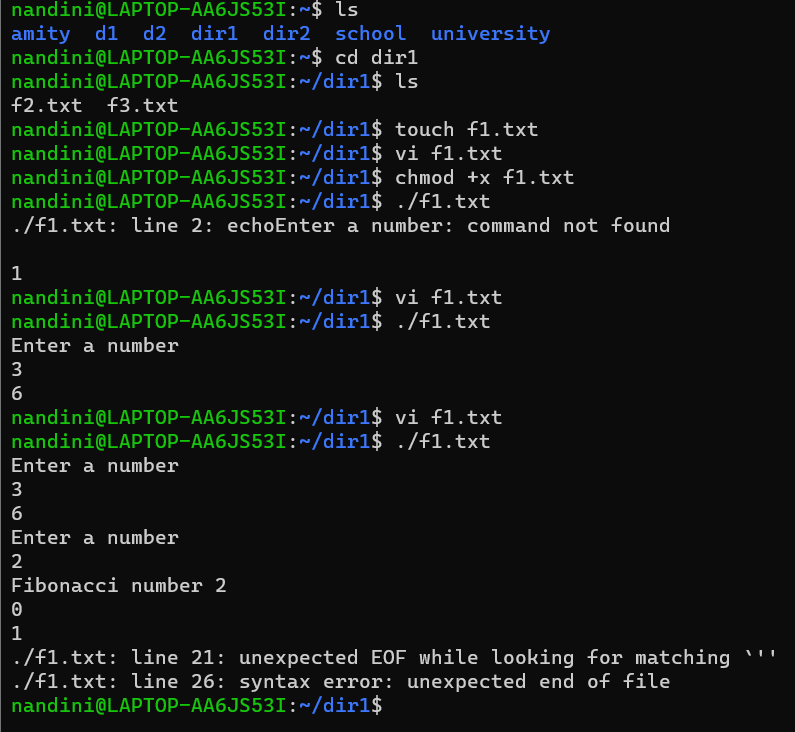
Description automatically generated

1. Write a shell script to print the Fibonacci series of ‘n’ elements and print the sum of the given series.

Text

Description automatically generatedText

Description automatically generated



1. Write a shell script for simple calculator.

Text

Description automatically generated

Text

Description automatically generated

1. Write a shell script that accept a file name starting and ending line numbers as arguments and display all the lines between given line number.

Text

Description automatically generated

Text

Description automatically generated

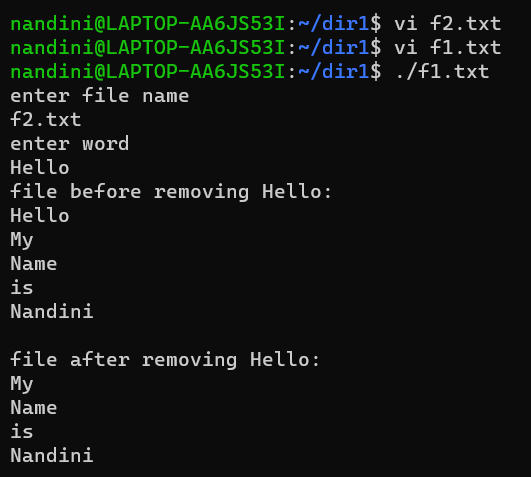
Text

Description automatically generated

1. Write a shell script that delete all lines containing a specified word.

Text

Description automatically generated



1. Write a shell script that displays a list of all the files in the current directory

Text

Description automatically generated

Text

Description automatically generated

**FCFS ALGORITHM**

//C Program to demonstrate the First Come First Serve CPU Scheduling Algorithm .

//CODE

#include <stdio.h>

void calculateTime(int process[], int n, int arrival[], int burst[])

{

int exit[n], turn\_around[n], waiting\_time[n];

float avg\_waiting\_time = 0, avg\_turn\_around\_time = 0;

exit[0] = arrival[0] + burst[0];

for (int i = 1; i < n; i++)

exit[i] = exit[i - 1] + burst[i];

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

{

turn\_around[i] = exit[i] - arrival[i];

waiting\_time[i] = turn\_around[i] - burst[i];

avg\_waiting\_time += waiting\_time[i];

avg\_turn\_around\_time += turn\_around[i];

}

avg\_waiting\_time /= n;

avg\_turn\_around\_time /= n;

printf("\nProcess\t Arrival Time\t Burst Time\t Exit Time\t Turn Around Time\t Waiting Time\n");

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

printf("%d\t\t %d\t\t %d\t\t %d\t\t %d\t\t %d\n", process[i], arrival[i], burst[i], exit[i],

turn\_around[i], waiting\_time[i]);

printf("\nAverage Waiting Time: %.2f\n", avg\_waiting\_time);

printf("Average Turn Around Time: %.2f\n", avg\_turn\_around\_time);

}

int main()

{

int n;

printf("Enter the number of processes: ");

scanf("%d", &n);

int process[n], arrival[n], burst[n];

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

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

{

process[i] = i + 1;

printf("\nEnter the arrival time: ");

scanf("%d", &arrival[i]);

printf("\nEnter the burst time: ");

scanf("%d",&burst[i]);

}

calculateTime(process, n, arrival, burst);

return 0;

}

**OUTPUT**

Text

Description automatically generated

**SJF ALGORITHM**

//C Program to demonstrate the Shortest Job First (SJF) CPU Scheduling Algorithm .

//CODE

#include <stdio.h>

struct process{

int pid;

int at;

int bt;

int prior;

int tat;

int wt;

int comp;

};

typedef struct process process;

int main()

{

int n;

printf("Enter the number of processes: ");

scanf("%d",&n);

process p\_array[n];

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

{

printf("Enter process id: ");

scanf("%d",&p\_array[i].pid);

printf("Enter process arrival time: ");

scanf("%d",&p\_array[i].at);

printf("Enter process burst time: ");

scanf("%d",&p\_array[i].bt);

p\_array[i].tat =0;

p\_array[i].wt =0;

p\_array[i].prior =0;

p\_array[i].comp =0;

}

int cpu\_time=0;

int t\_n = n;

while(t\_n>0)

{

int j=-1;

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

{

if(j==-1 && p\_array[i].at<=cpu\_time && p\_array[i].comp==0)

{

j=i;

}

else if(p\_array[i].at<=cpu\_time && p\_array[i].comp==0 && p\_array[j].bt>p\_array[i].bt)

{

j=i;

}

}

if(j!=-1)

{

cpu\_time+=p\_array[j].bt;

p\_array[j].tat = cpu\_time-p\_array[j].at;

p\_array[j].wt = p\_array[j].tat - p\_array[j].bt;

p\_array[j].comp = 1;

t\_n-=1;

}

else

{

cpu\_time+=1;

}

}

float t\_wt = 0;

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

{

t\_wt += p\_array[i].wt;

printf("%d %d %d %d %d",p\_array[i].pid,p\_array[i].at,p\_array[i].bt,p\_array[i].tat,p\_array[i].wt);

printf("\n");

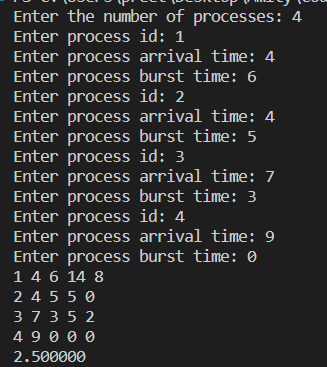
}

printf("%f",t\_wt/n);

    return 0;

}

**OUTPUT**



**SRTF ALGORITHM**

//C Program to demonstrate the Shortest Remaining Time First (SRTF)

CPU Scheduling Algorithm.

//CODE

#include <stdio.h>

struct process{

int pid;

int at;

int bt;

int t\_bt;

int prior;

int tat;

int wt;

int comp;

};

typedef struct process process;

int main()

{

int n;

printf("Enter the number of processes: ");

scanf("%d",&n);

process p\_array[n];

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

{

printf("Enter process id: ");

scanf("%d",&p\_array[i].pid);

printf("Enter process arrival time: ");

scanf("%d",&p\_array[i].at);

printf("Enter process burst time: ");

scanf("%d",&p\_array[i].bt);

p\_array[i].t\_bt=p\_array[i].bt;

p\_array[i].tat =0;

p\_array[i].wt =0;

p\_array[i].prior =0;

p\_array[i].comp =0;

}

int cpu\_time=0;

int t\_n = n;

while(t\_n>0)

{

int j=-1;

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

{

if(j==-1 && p\_array[i].at<=cpu\_time && p\_array[i].comp==0)

{

j=i;

}

else if(p\_array[i].at<=cpu\_time && p\_array[i].comp==0 && p\_array[j].bt>p\_array[i].bt)

{

j=i;

}

}

if(j!=-1)

{

cpu\_time+=1;

p\_array[j].bt-=1;

if(p\_array[j].bt==0)

{

p\_array[j].tat = cpu\_time-p\_array[j].at;

p\_array[j].wt = p\_array[j].tat - p\_array[j].t\_bt;

p\_array[j].comp = 1;

t\_n-=1;

}

}

else

{

cpu\_time+=1;

}

}

float t\_wt = 0;

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

{

t\_wt += p\_array[i].wt;

printf("%d %d %d %d %d",p\_array[i].pid,p\_array[i].at,p\_array[i].t\_bt,p\_array[i].tat,p\_array[i].wt);

printf("\n");

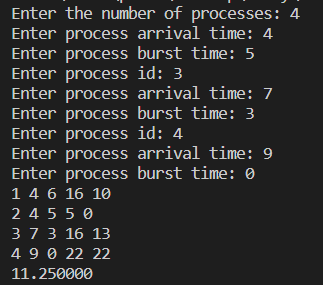
}

printf("%f",t\_wt/n);

return 0;

}

**OUTPUT**



**PRIORITY SCHEDULING ALGORITHM**

//1. C Program to demonstrate the Priority Scheduling Algorithm (Non-Preemptive)

//CODE

#include<stdio.h>

#define MIN -9999;

struct proc

{

int no,at,bt,ct,wt,tat,pri,status;

};

struct proc read(int i)

{

struct proc p;

printf("\nProcess No: %d\n",i);

p.no=i;

printf("Enter Arrival Time: ");

scanf("%d",&p.at);

printf("Enter Burst Time: ");

scanf("%d",&p.bt);

printf("Enter Priority: ");

scanf("%d",&p.pri);

p.status=0;

return p;

}

int main()

{

int n,l,ct=0,remaining;

struct proc p[10],temp;

float avgtat=0,avgwt=0;

printf("<--Highest Priority First Scheduling Algorithm (Non-Preemptive)-->\n");

printf("Enter Number of Processes: ");

scanf("%d",&n);

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

p[i]=read(i+1);

for(int i=0;i<n-1;i++)

for(int j=0;j<n-i-1;j++)

if(p[j].at>p[j+1].at)

{

temp=p[j];

p[j]=p[j+1];

p[j+1]=temp;

}

p[9].pri=MIN;

remaining=n;

printf("\nProcessNo\tAT\tBT\tPri\tCT\tTAT\tWT\tRT\n");

for(ct=p[0].at;remaining!=0;)

{

l=9;

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

if(p[i].at<=ct && p[i].status!=1 && p[i].pri>p[l].pri)

l=i;

p[l].ct=ct=ct+p[l].bt;

p[l].tat=p[l].ct-p[l].at;

avgtat+=p[l].tat;

p[l].wt=p[l].tat-p[l].bt;

avgwt+=p[l].wt;

p[l].status=1;

remaining--;

printf("P%d\t\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n",p[l].no,p[l].at,p[l].bt,p[l].pri,p[l].ct,p[l].tat,p[l].wt,

p

[l].wt);

}

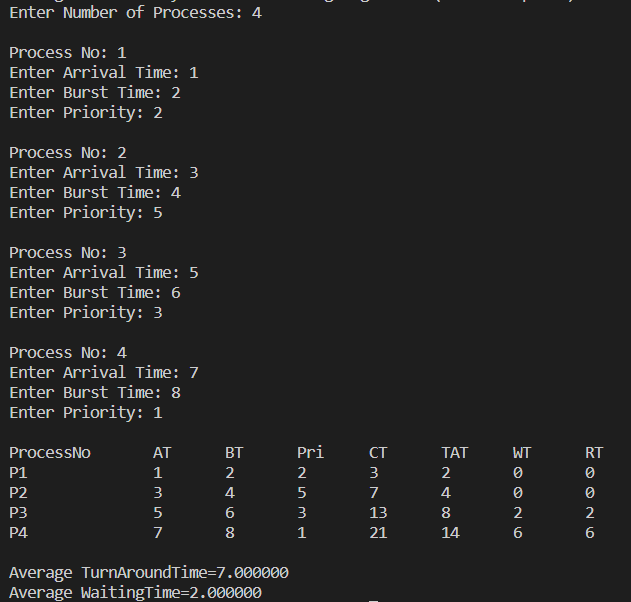
avgtat/=n;

avgwt/=n;

printf("\nAverage TurnAroundTime=%f\nAverage WaitingTime=%f",avgtat,avgwt);

}

**OUTPUT**



//2. C Program to demonstrate the Priority Scheduling Algorithm (Preemptive)

//CODE

#include<stdio.h>

#define MIN -9999;

struct proc

{

int no,at,bt,rt,ct,wt,tat,pri,temp;

};

struct proc read(int i)

{

struct proc p;

printf("\nProcess No: %d\n",i);

p.no=i;

printf("Enter Arrival Time: ");

scanf("%d",&p.at);

printf("Enter Burst Time: ");

scanf("%d",&p.bt);

p.rt=p.bt;

printf("Enter Priority: ");

scanf("%d",&p.pri);

p.temp=p.pri;

return p;

}

int main()

{

int i,n,c,remaining,max\_val,max\_index;

struct proc p[10],temp;

float avgtat=0,avgwt=0;

printf("<--Highest Priority First Scheduling Algorithm (Preemptive)-->\n");

printf("Enter Number of Processes: ");

scanf("%d",&n);

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

p[i]=read(i+1);

remaining=n;

for(int i=0;i<n-1;i++)

for(int j=0;j<n-i-1;j++)

if(p[j].at>p[j+1].at)

{

temp=p[j];

p[j]=p[j+1];

p[j+1]=temp;

}

max\_val=p[0].temp,max\_index=0;

for(int j=0;j<n&&p[j].at<=p[0].at;j++)

if(p[j].temp>max\_val)

max\_val=p[j].temp,max\_index=j;

i=max\_index;

c=p[i].ct=p[i].at+1;

p[i].rt--;

if(p[i].rt==0)

{

p[i].temp=MIN;

remaining--;

}

while(remaining>0)

{

max\_val=p[0].temp,max\_index=0;

for(int j=0;j<n&&p[j].at<=c;j++)

if(p[j].temp>max\_val)

max\_val=p[j].temp,max\_index=j;

i=max\_index;

p[i].ct=c=c+1;

p[i].rt--;

if(p[i].rt==0)

{

p[i].temp=MIN;

remaining--;

}

}

printf("\nProcessNo\tAT\tBT\tPri\tCT\tTAT\tWT\n");

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

{

p[i].tat=p[i].ct-p[i].at;

avgtat+=p[i].tat;

p[i].wt=p[i].tat-p[i].bt;

avgwt+=p[i].wt;

printf("P%d\t\t%d\t%d\t%d\t%d\t%d\t%d\n",p[i].no,p[i].at,p[i].bt,p[i].pri,p[i].ct,p[i].tat,p[i].wt);

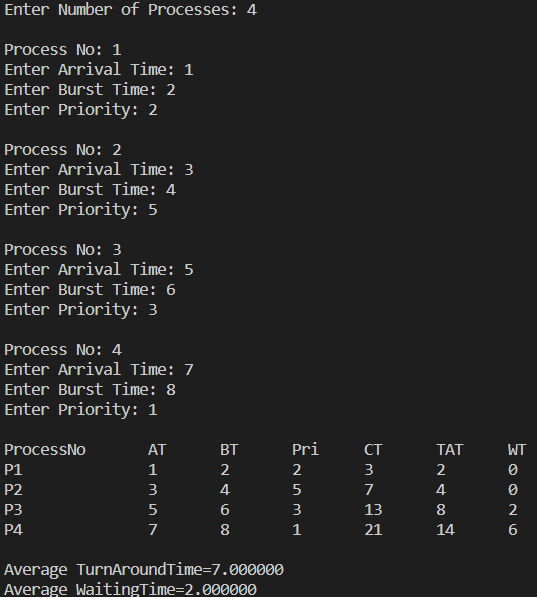
}

avgtat/=n,avgwt/=n;

printf("\nAverage TurnAroundTime=%f\nAverage WaitingTime=%f",avgtat,avgwt);

}

**OUTPUT**



**ROUND ROBIN SCHEDULING ALGORITHM**

//C++ Program to demonstrate the Round Robin Scheduling Algorithm

//CODE

#include <iostream>

#include <cstdlib>

#include <queue>

#include <cstdio>

using namespace std;

typedef struct process

{

    int id, at, bt, st, ft, pr;

    float wt, tat;

}

process;

process p[10], p1[10], temp;

queue<int> q1;

int accept(int ch);

void turnwait(int n);

void display(int n);

void ganttrr(int n);

int main()

{

    int i, n, ts, ch, j, x;

    p[0].tat = 0;

    p[0].wt = 0;

    n = accept(ch);

    ganttrr(n);

    turnwait(n);

    display(n);

    return 0;

}

int accept(int ch)

{

    int i, n;

    printf("Enter the Total Number of Process to Round Robin: ");

    scanf("%d", &n);

    if (n == 0)

    {

        printf("Invalid");

        exit(1);

    }

    cout << endl;

    for (i = 1; i <= n; i++)

    {

        printf("Enter an Arrival Time of the Process to Round Robin P%d: ", i);

        scanf("%d", &p[i].at);

        p[i].id = i;

    }

    cout << endl;

    for (i = 1; i <= n; i++)

    {

        printf("Enter a Burst Time of the Process to Round Robin P%d: ", i);

        scanf("%d", &p[i].bt);

    }

    for (i = 1; i <= n; i++)

    {

        p1[i] = p[i];

    }

    return n;

}

void ganttrr(int n)

{

    int i, ts, m, nextval, nextarr;

    nextval = p1[1].at;

    i = 1;

    cout << "\nEnter the Time Slice or Quantum: ";

    cin >> ts;

    for (i = 1; i <= n && p1[i].at <= nextval; i++)

    {

        q1.push(p1[i].id);

    }

    while (!q1.empty())

    {

        m = q1.front();

        q1.pop();

        if (p1[m].bt >= ts)

        {

            nextval = nextval + ts;

        }

        else

        {

            nextval = nextval + p1[m].bt;

        }

        if (p1[m].bt >= ts)

        {

            p1[m].bt = p1[m].bt - ts;

        }

        else

        {

            p1[m].bt = 0;

        }

        while (i <= n && p1[i].at <= nextval)

        {

            q1.push(p1[i].id);

            i++;

        }

        if (p1[m].bt > 0)

        {

            q1.push(m);

        }

        if (p1[m].bt <= 0)

        {

            p[m].ft = nextval;

        }

    }

}

void turnwait(int n)

{

    int i;

    for (i = 1; i <= n; i++)

    {

        p[i].tat = p[i].ft - p[i].at;

        p[i].wt = p[i].tat - p[i].bt;

        p[0].tat = p[0].tat + p[i].tat;

        p[0].wt = p[0].wt + p[i].wt;

    }

    p[0].tat = p[0].tat / n;

    p[0].wt = p[0].wt / n;

}

void display(int n)

{

    int i;

    cout<<"\n\n\n";

    printf("\nPID\tAT\tBT\tFT\tTAT\t\tWT");

    for (i = 1; i <= n; i++)

    {

        printf("\nP%d\t%d\t%d\t%d\t%.3f\t%.3f", p[i].id, p[i].at, p[i].bt, p[i].ft, p[i].tat, p[i].wt);

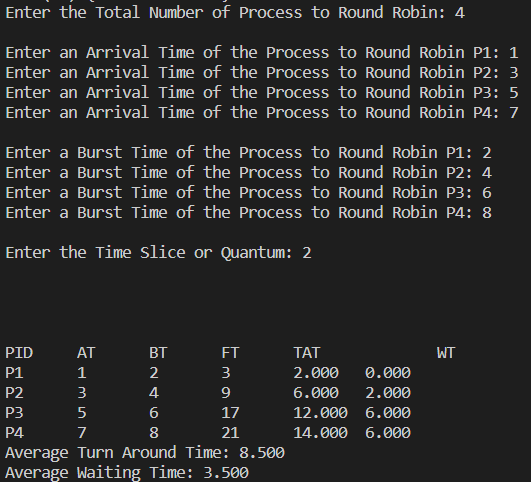
    }

    printf("\nAverage Turn Around Time: %.3f", p[0].tat);

    printf("\nAverage Waiting Time: %.3f", p[0].wt);

}

OUTPUT



**BANKER’S ALGORITHM**

//C Program to demonstrate the Round Robin Scheduling Algorithm

//CODE

#include <stdio.h>

#include <conio.h>

int main()

{

int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10], safeSequence[10];

int p, r, i, j, process, count;

count = 0;

printf("Enter the no of processes : ");

scanf("%d", &p);

for(i = 0; i< p; i++)

completed[i] = 0;

printf("\n\nEnter the no of resources : ");

scanf("%d", &r);

printf("\n\nEnter the Max Matrix for each process : ");

for(i = 0; i < p; i++)

{

printf("\nFor process %d : ", i + 1);

for(j = 0; j < r; j++)

scanf("%d", &Max[i][j]);

}

printf("\n\nEnter the allocation for each process : ");

for(i = 0; i < p; i++)

{

printf("\nFor process %d : ",i + 1);

for(j = 0; j < r; j++)

scanf("%d", &alloc[i][j]);

}

printf("\n\nEnter the Available Resources : ");

for(i = 0; i < r; i++)

scanf("%d", &avail[i]);

for(i = 0; i < p; i++)

for(j = 0; j < r; j++)

need[i][j] = Max[i][j] - alloc[i][j];

do

{

printf("\n Max matrix:\tAllocation matrix:\n");

for(i = 0; i < p; i++)

{

for( j = 0; j < r; j++)

printf("%d ", Max[i][j]);

printf("\t\t");

for( j = 0; j < r; j++)

printf("%d ", alloc[i][j]);

printf("\n");

}

process = -1;

for(i = 0; i < p; i++)

{

if(completed[i] == 0)//if not completed

{

process = i ;

for(j = 0; j < r; j++)

{

if(avail[j] < need[i][j])

{

process = -1;

break;

}

}

}

if(process != -1)

break;

}

if(process != -1)

{

printf("\nProcess %d runs to completion!", process + 1);

safeSequence[count] = process + 1;

count++;

for(j = 0; j < r; j++)

{

avail[j] += alloc[process][j];

alloc[process][j] = 0;

Max[process][j] = 0;

completed[process] = 1;

}

}

}while(count != p && process != -1);

if(count == p)

{

printf("\nThe system is in a safe state!!\n");

printf("Safe Sequence : < ");

for( i = 0; i < p; i++)

printf("%d ", safeSequence[i]);

printf(">\n");

}

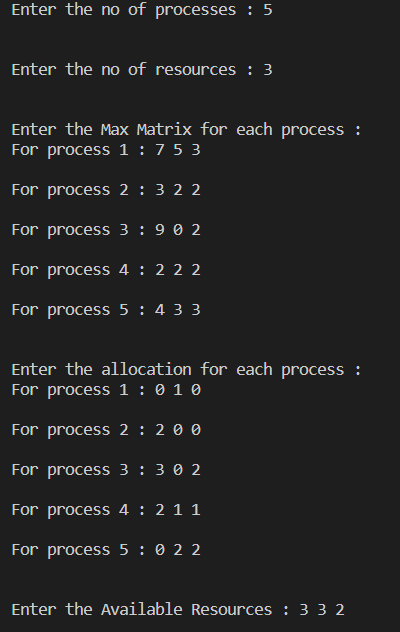
else

printf("\nThe system is in an unsafe state!!");

getch();

}

OUTPUT





**FIFO PAGE REPLACEMENT ALGORITHM**

#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");

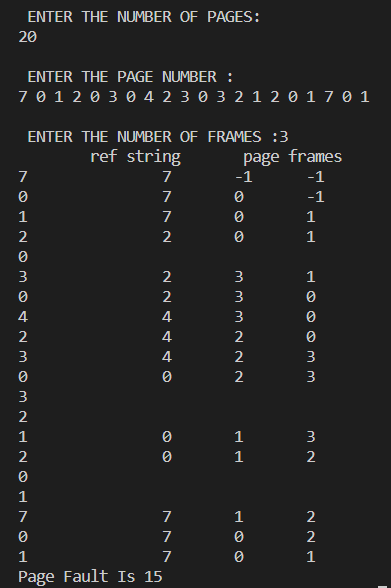
}

                        printf("Page Fault Is %d",count);

                        return 0;

}

OUTPUT



**OPTIMAL PAGE REPLACEMENT ALGORITHM**

#include <stdio.h>

#include <stdlib.h>

int search(int A[],int np,int p)

{

    int bool=0;

    for(int i = 0;i<np;i++)

    {

        if(p==A[i])

        {

            bool = 1;

            break;

        }

    }

    return(bool);

}

int \* replace(int A[],int np,int p1,int p2)

{

    for(int i=0;i<np;i++)

    {

        if(A[i]==p1)

        {

            A[i]=p2;

            break;

        }

    }

    return(A);

}

void traverse\_array(int A[], int n)

{

    for (int i = 0; i < n; i++)

    {

        if(A[i]==-1)

        {

            printf("\_ ");

        }

        else

        {

            printf("%d ", A[i]);

        }

    }

    printf("\n");

}

int to\_replace(int A[],int i,int np,int f[],int nf)

{

    int to\_replace = -1;

    int last\_use = -1;

    for(int j=0;j<nf;j++)

    {

        if(f[j]==-1)

        {

            to\_replace = -1;

            break;

        }

        else

        {

            for(int k =i;k<np;k++)

            {

                if(A[k]==f[j])

                {

                    if(last\_use==-1)

                    {

                        to\_replace=f[j];

                        last\_use=k;

                    }

                    else if(last\_use<k)

                    {

                        to\_replace=f[j];

                        last\_use=k;

                    }

                    break;

                }

                else if(k==np-1)

                {

                    to\_replace=f[j];

                    last\_use=k;

                }

            }

        }

    }

    return(to\_replace);

}

int main()

{

    int np;

    printf("Enter the number of pages: ");

    scanf("%d",&np);

    int nf;

    printf("Enter the number of frames: ");

    scanf("%d",&nf);

    int\* frame = (int \*)malloc(sizeof(int)\*nf);

    int\* pages = (int \*)malloc(sizeof(int)\*np);

    for(int i = 0;i<nf;i++)

    {

        frame[i]=-1;

    }

    for(int i =0;i<np;i++)

    {

        printf("Enter page %d: ",i);

        scanf("%d",&pages[i]);

    }

    printf("\n");

    int hit = 0;

    int fault = 0;

    for(int i = 0;i<np;i++)

    {

        traverse\_array(frame,nf);

        printf("%d ",pages[i]);

        if(search(frame,nf,pages[i])==1)

        {

            printf(" Hit\n");

            hit+=1;

        }

        else

        {

            printf("Fault %d\n",to\_replace(pages,i,np,frame,nf));

            int replace\_page = to\_replace(pages,i,np,frame,nf);

            fault+=1;

            frame = replace(frame,nf,replace\_page,pages[i]);

        }

        printf("\n");

    }

    printf("Total Hit:%d\nTotal Fault:%d",hit,fault);

    return 0;

}

OUTPUT

A screenshot of a computer

Description automatically generated with medium confidence Graphical user interface

Description automatically generated with medium confidence A screenshot of a computer

Description automatically generated with low confidence

**LEAST RECENTLY USED PAGE REPLACEMENT ALGORITHM**

#include<stdio.h>

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);

}

OUTPUT

Text

Description automatically generated

**C-LOOK DISK SCHEDULING ALGORITHM**

#include <stdio.h>

#include <stdlib.h>

#define MAX 200

void clook(int arr[], int head, int size)

{

int total\_movement = 0;

int current = head;

int start\_index = 0;

int end\_index = size-1;

// Sort the request queue in ascending order

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

for (int j = i+1; j < size; j++) {

if (arr[i] > arr[j]) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

// Find the index at which the head starts moving towards the end of the disk

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

if (arr[i] > head) {

start\_index = i;

break;

}

}

// If the head moves towards the end of the disk

if (start\_index != 0) {

end\_index = start\_index-1;

}

// Traverse the request queue in the direction of the head movement

for (int i = start\_index; i < size; i++) {

total\_movement += abs(current - arr[i]);

current = arr[i];

}

for (int i = 0; i <= end\_index; i++) {

total\_movement += abs(current - arr[i]);

current = arr[i];

}

printf("Total head movement = %d\n", total\_movement);

}

int main()

{

int n, head, arr[MAX];

printf("Enter the size of the request queue: ");

scanf("%d", &n);

printf("Enter the initial position of the disk head: ");

scanf("%d", &head);

printf("Enter the request queue: ");

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

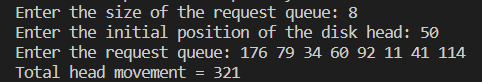
scanf("%d", &arr[i]);

}

clook(arr, head, n);

return 0;

}



**C-SCAN DISK SCHEDULING ALGORITHM**

#include <stdio.h>

#include <stdlib.h>

#define MAX 200

void c\_scan(int arr[], int head, int size)

{

int total\_movement = 0;

int current = head;

int start\_index = 0;

int i;

// Sort the request queue in ascending order

for (i = 0; i < size; i++) {

for (int j = i+1; j < size; j++) {

if (arr[i] > arr[j]) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

// Find the index at which the head starts moving towards the end of the disk

for (i = 0; i < size; i++) {

if (arr[i] > head) {

start\_index = i;

break;

}

}

// Traverse the request queue in the direction of the head movement

for (i = start\_index; i < size; i++) {

total\_movement += abs(current - arr[i]);

current = arr[i];

}

// If the last request is not at the end of the disk, move to the end of the disk

if (arr[size-1] != 199) {

total\_movement += abs(current - 199);

current = 199;

}

// Move to the beginning of the disk

total\_movement += 199;

current = 0;

// Traverse the request queue in the direction of the head movement again

for (i = 0; i < start\_index; i++) {

total\_movement += abs(current - arr[i]);

current = arr[i];

}

printf("Total head movement = %d\n", total\_movement);

}

int main()

{

int n, head, arr[MAX];

printf("Enter the size of the request queue: ");

scanf("%d", &n);

printf("Enter the initial position of the disk head: ");

scanf("%d", &head);

printf("Enter the request queue: ");

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

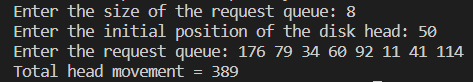
scanf("%d", &arr[i]);

}

c\_scan(arr, head, n);

return 0;

}



**FCFS DISK SCHEDULING ALGORITHM**

#include <stdio.h>

#include <stdlib.h>

#define MAX 200

void fcfs(int arr[], int head, int size)

{

int total\_movement = 0;

int current = head;

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

total\_movement += abs(current - arr[i]);

current = arr[i];

}

printf("Total head movement = %d\n", total\_movement);

}

int main()

{

int n, head, arr[MAX];

printf("Enter the size of the request queue: ");

scanf("%d", &n);

printf("Enter the initial position of the disk head: ");

scanf("%d", &head);

printf("Enter the request queue: ");

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

scanf("%d", &arr[i]);

}

fcfs(arr, head, n);

return 0;

}

Text

Description automatically generated

**LOOK DISK SCHEDULING ALGORITHM**

#include <stdio.h>

#include <stdlib.h>

#define MAX 200

void look(int arr[], int head, int size)

{

int total\_movement = 0;

int current = head;

int start\_index = 0;

int end\_index = size-1;

// Sort the request queue in ascending order

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

for (int j = i+1; j < size; j++) {

if (arr[i] > arr[j]) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

// Find the index at which the head starts moving towards the end of the disk

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

if (arr[i] > head) {

start\_index = i;

break;

}

}

// If the head moves towards the end of the disk

if (start\_index != 0) {

end\_index = start\_index-1;

}

// Traverse the request queue in the direction of the head movement

for (int i = start\_index; i < size; i++) {

total\_movement += abs(current - arr[i]);

current = arr[i];

}

for (int i = end\_index; i >= 0; i--) {

total\_movement += abs(current - arr[i]);

current = arr[i];

}

printf("Total head movement = %d\n", total\_movement);

}

int main()

{

int n, head, arr[MAX];

printf("Enter the size of the request queue: ");

scanf("%d", &n);

printf("Enter the initial position of the disk head: ");

scanf("%d", &head);

printf("Enter the request queue: ");

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

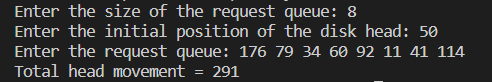
scanf("%d", &arr[i]);

}

look(arr, head, n);

return 0;

}



**SCAN DISK SCHEDULING ALGORITHM**

#include <stdio.h>

#include <stdlib.h>

#define MAX 200

void scan(int arr[], int head, int size)

{

int total\_movement = 0;

int current = head;

int start\_index = 0;

int i;

// Sort the request queue in ascending order

for (i = 0; i < size; i++) {

for (int j = i+1; j < size; j++) {

if (arr[i] > arr[j]) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

// Find the index at which the head starts moving towards the end of the disk

for (i = 0; i < size; i++) {

if (arr[i] > head) {

start\_index = i;

break;

}

}

// Traverse the request queue in the direction of the head movement

for (i = start\_index; i < size; i++) {

total\_movement += abs(current - arr[i]);

current = arr[i];

}

// Add the movement to the end of the disk

total\_movement += abs(current - 199);

current = 199;

// Reverse direction and move towards the beginning of the disk

for (i = start\_index-1; i >= 0; i--) {

total\_movement += abs(current - arr[i]);

current = arr[i];

}

printf("Total head movement = %d\n", total\_movement);

}

int main()

{

int n, head, arr[MAX];

printf("Enter the size of the request queue: ");

scanf("%d", &n);

printf("Enter the initial position of the disk head: ");

scanf("%d", &head);

printf("Enter the request queue: ");

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

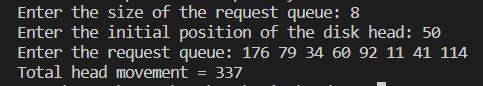
scanf("%d", &arr[i]);

}

scan(arr, head, n);

return 0;

}



**SSTF DISK SCHEDULING ALGORITHM**

#include <stdio.h>

#include <stdlib.h>

#define MAX 200

void sstf(int arr[], int head, int size)

{

int total\_movement = 0;

int visited[MAX] = {0};

int current = head;

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

int shortest\_distance = \_\_INT\_MAX\_\_;

int selected = -1;

for (int j = 0; j < size; j++) {

if (!visited[j]) {

int distance = abs(current - arr[j]);

if (distance < shortest\_distance) {

shortest\_distance = distance;

selected = j;

}

}

}

visited[selected] = 1;

total\_movement += shortest\_distance;

current = arr[selected];

}

printf("Total head movement = %d\n", total\_movement);

}

int main()

{

int n, head, arr[MAX];

printf("Enter the size of the request queue: ");

scanf("%d", &n);

printf("Enter the initial position of the disk head: ");

scanf("%d", &head);

printf("Enter the request queue: ");

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

scanf("%d", &arr[i]);

}

sstf(arr, head, n);

return 0;

}

Text

Description automatically generated

**OPEN ENDED QUESTION**

(a) Repository/Directory Synchronizer: Client-server application which is capable to synchronize the local changes to a remote folder.

(b) WAP for your own Signal Handler which will execute when you type (CTRL+D)

#include <stdio.h>

#include <stdlib.h>

#include <signal.h>

void handle\_signal(int signal) {

printf("\nReceived signal %d. Exiting...\n", signal);

exit(EXIT\_SUCCESS);

}

int main() {

// Install the signal handler for SIGINT

if (signal(SIGINT, handle\_signal) == SIG\_ERR) {

perror("signal");

return EXIT\_FAILURE;

}

// Loop until CTRL+D is typed

printf("Type CTRL+D to exit...\n");

while (1) {

if (getchar() == EOF) {

handle\_signal(SIGINT);

}

}

return EXIT\_SUCCESS;

}

Graphical user interface, text

Description automatically generated

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.No | Title | Date Allotted | Date Checked | Sign/ Remarks |
| 1. | Linux Terminal Commands | 3/01/2023 |  |  |
| 2. | Linux Shell Commands | 10/01/2023 |  |  |
| 3. | Linux Shell Programs | 17/01/2023 |  |  |
| 4. | Shell Script | 24/01/2023 |  |  |
| 5. | Process Scheduling(FCFS) | 7/02/2023 |  |  |
| 6. | Process Scheduling(SJF) | 14/02/2023 |  |  |
| 7. | Process Scheduling(SRTF) | 14/02/2023 |  |  |
| 8. | Process Scheduling(Non-preempti ve Priority) | 21/02/2023 |  |  |
| 9. | Process Scheduling(Preemptive Priority) | 21/02/2023 |  |  |
| 10. | Process Scheduling(Round Robin) | 28/02/2023 |  |  |
| 11. | Resource Allocation and Safe States(Banker’s Algorithm) | 13/03/2023 |  |  |
| 12. | Page Re-Allocation (FIFO) | 04/04/2023 |  |  |
| 13. | Page Re-Allocation (LRU) | 04/04/2023 |  |  |
| 14. | Page Re-Allocation (Optimal) | 04/04/2023 |  |  |

**AMITY SCHOOL OF ENGINEERING & TECHNOLOGY**

AMITY UNIVERSITY CAMPUS, SECTOR-125, NOIDA-201303

Logo

Description automatically generated

**OPERATING SYSTEM LAB**

PRACTICAL FILE

COURSE CODE: CSE202

Submitted to: Submitted by:

Dr Sumit Kumar Nandini Sain

4CSE6(X)