OPERATING SYSTEM

ITC11

DEPARTMENT OF INFORMATION TECHNOLOGY

NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

SHIVANI GUPTA

2018UIT2586

IT-2

INDEX

LINUX COMMANDS

1. **ip** : Used for performing several network administration tasks
2. **id** : Used to find out user and group names and numeric IDs (UID or group ID) of the current user or any other user in the server

```
shivani@shivani-VirtualBox:~$ ip
Usage: ip [ OPTIONS ] OBJECT { COMMAND | help }
       ip [ -force ] -batch filename
where  OBJECT := { link | address | addrlabel | route | rule | neigh | ntable |
                   tunnel | tuntap | maddress | mroute | mrule | monitor | xfrm
 |
                   netns | l2tp | fou | macsec | tcp_metrics | token | netconf
| ila |
                   vrf | sr }
       OPTIONS := { -V[ersion] | -s[tatistics] | -d[etails] | -r[esolve] |
                    -h[uman-readable] | -iec |
                    -f[amily] { inet | inet6 | ipx | dnet | mpls | bridge | lin
k } |
                    -4 | -6 | -I | -D | -B | -0 |
                    -l[oops] { maximum-addr-flush-attempts } | -br[ief] |
                    -o[neline] | -t[imestamp] | -ts[hort] | -b[atch] [filename]
 |
                    -rc[vbuf] [size] | -n[etns] name | -a[ll] | -c[olor]}
shivani@shivani-VirtualBox:~$ id
uid=1000(shivani) gid=1000(shivani) groups=1000(shivani),4(adm),24(cdrom),27(su
do),30(dip),46(plugdev),116(lpadmin),126(sambashare)
```

3. **passwd** : Used to change the user account passwords

```
shivani@shivani-VirtualBox:/home$ passwd
Changing password for shivani.
(current) UNIX password:
passwd: Authentication token manipulation error
passwd: password unchanged
```

4. **version** : gives the version of the architecture

```
shivani@shivani-VirtualBox:/$ arch --version
arch (GNU coreutils) 8.28
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Written by David MacKenzie and Karel Zak.
```

5. **pwd** : It prints the path of the working directory, starting from the root
6. **cd** : Known as change directory command. It is used to change current working directory
7. **who** : Used to get information about currently logged in user on to system

```
shivani@shivani-VirtualBox:~$ pwd
/home/shivani
shivani@shivani-VirtualBox:~$ cd ..
shivani@shivani-VirtualBox:/home$ who
shivani  :0           2020-04-24 04:43 (:0)
```

8. **dir** : Used to list the contents of a directory
9. **date** : Used to display the system date and time. It is also used to set date and time of the system

```
shivani@shivani-VirtualBox:/home$ dir
shivani
shivani@shivani-VirtualBox:/home$ date
Fri Apr 24 05:14:09 EDT 2020
```

10. **ls** : listing the contents of a directory or directories
11. **arch** : Used to print the computer architecture

```
shivani@shivani-VirtualBox:/home$ ls
shivani
shivani@shivani-VirtualBox:/home$ cd ..shivani@shivani-VirtualBox:/$ arch
x86_64
```

12. **w** : Used to show who is logged on and what they are doing
13. **clear** : **clear** the terminal screen in **Linux**

```
shivani@shivani-VirtualBox:/home$ w
 05:52:07 up  1:10,  1 user,  load average: 0.00, 0.05, 0.26
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
shivani  :0       :0               04:43   ?xdm?   1:17   0.01s /usr/lib/gdm3/
shivani@shivani-VirtualBox:/home$ clear
```

14. **hash** : Used to maintain a hash table of recently executed programs

```
shivani@shivani-VirtualBox:/home$ hash
hits    command
   1    /bin/date
   1    /usr/bin/touch
   1    /usr/bin/who
   1    /bin/mkdir
   1    /bin/dir
   1    /usr/bin/lpr
   1    /usr/bin/passwd
   2    /bin/ls
   3    /usr/bin/arch
   1    /usr/bin/clear
```

15. **time** : Used to execute a command and prints a summary of real-time, user CPU time and system CPU time spent by executing a command when it terminates.
16. **ssh** : Protocol used to securely connect to a remote server/system.
17. **rev** : Used to reverse the lines characterwise

```
shivani@shivani-VirtualBox:~$ time

real    0m0.000s
user    0m0.000s
sys     0m0.000s
shivani@shivani-VirtualBox:~$ ssh
usage: ssh [-46AaCfGgKkMNnqsTtVvXxYy] [-b bind_address] [-c cipher_spec]
           [-D [bind_address:]port] [-E log_file] [-e escape_char]
           [-F configfile] [-I pkcs11] [-i identity_file]
           [-J [user@]host[:port]] [-L address] [-l login_name] [-m mac_spec]
           [-O ctl_cmd] [-o option] [-p port] [-Q query_option] [-R address]
           [-S ctl_path] [-W host:port] [-w local_tun[:remote_tun]]
           [user@]hostname [command]
shivani@shivani-VirtualBox:~$ rev
Shivani
inavihS
```

18. **users** : Used to show the user names of users currently logged in to the current host.

19. **uname** : Displays the information about the system
20. **top** : Provides a dynamic real-time view of the running system

```
shivani@shivani-VirtualBox:/home$ users
shivani
shivani@shivani-VirtualBox:/home$ uname
Linux
shivani@shivani-VirtualBox:/home$ top
```

```
top - 05:53:56 up  1:12,  1 user,  load average: 0.02, 0.04, 0.23
Tasks: 206 total,   1 running, 171 sleeping,   0 stopped,   0 zombie
%Cpu(s):  9.1 us,  2.1 sy,  0.0 ni, 88.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  2990872 total,   983176 free,  1133028 used,   874668 buff/cache
KiB Swap:  1942896 total,  1941860 free,     1036 used.  1683528 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 1281 shivani   20   0 2995492 359388 104096 S  5.6 12.0   4:18.68 gnome-shell
 1134 shivani   20   0  407400  82588  45052 S  4.3  2.8   1:18.92 Xorg
 1636 shivani   20   0  803584  37300  27992 S  3.3  1.2   0:13.36 gnome-term+
 3302 shivani   20   0   51184   3900   3280 R  1.0  0.1   0:00.17 top
    1 root      20   0  225676   9380   6696 S  0.3  0.3   0:04.56 systemd
  480 root      -51   0       0      0      0 S  0.3  0.0   0:01.32 irq/18-vmw+
 1301 shivani   20   0  377916  10212   8056 S  0.3  0.3   0:06.84 ibus-daemon
 3264 root      20   0       0      0      0 I  0.3  0.0   0:00.07 kworker/u2+
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
    3 root       0 -20       0      0      0 I  0.0  0.0   0:00.00 rcu_gp
```

FIRST COME FIRST SERVE

```
#include <iostream>
using namespace std;
void findWaitingTime(int processes[], int n, int bt[], int wt[], int at[])
{
    int service_time[n];
    service_time[0] = 0;
    wt[0] = 0;
    for (int i = 1; i < n; i++)
    {
        service_time[i] = service_time[i - 1] + bt[i - 1];
        wt[i] = service_time[i] - at[i];
        if (wt[i] < 0)
            wt[i] = 0;
    }
}
void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[])
{
    for (int i = 0; i < n; i++)
        tat[i] = bt[i] + wt[i];
}
void findavgTime(int processes[], int n, int bt[], int at[])
{
    int wt[n], tat[n];
    findWaitingTime(processes, n, bt, wt, at);
    findTurnAroundTime(processes, n, bt, wt, tat);
    cout << "Processes "
         << " Burst Time "
         << " Arrival Time "
         << " Waiting Time "
         << " Turn-Around Time "
         << " Completion Time \n";
    int total_wt = 0, total_tat = 0;
    for (int i = 0; i < n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        int compl_time = tat[i] + at[i];
        cout << " " << i + 1 << "\t\t" << bt[i] << "\t\t"
             << at[i] << "\t\t" << wt[i] << "\t\t "
             << tat[i] << "\t\t " << compl_time << endl;
    }
    cout << "Average waiting time = "
         << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "
```

```
        << (float)total_tat / (float)n;
}
int main()
{
    int n;
    cout << "Enter the number of processes  ";
    cin >> n;
    int processes[n];
    int burst_time[n];
    int arrival_time[n];
    for (int i = 0; i < n; i++)
    {
        processes[i] = i + 1;
        cout << "Enter the arrival time of the " << i + 1 << "th process  ";
        cin >> arrival_time[i];
        cout << "Enter the burst time of the " << i + 1 << "th process  ";
        cin >> burst_time[i];
    }
    findavgTime(processes, n, burst_time, arrival_time);
    return 0;
}
```

OUTPUT

```
Enter the number of processes  4
Enter the arrival time of the 1th process  1
Enter the burst time of the 1th process  2
Enter the arrival time of the 2th process  3
Enter the burst time of the 2th process  4
Enter the arrival time of the 3th process  2
Enter the burst time of the 3th process  3
Enter the arrival time of the 4th process  1
Enter the burst time of the 4th process  4
Processes  Burst Time  Arrival Time  Waiting Time  Turn-Around Time  Completion Time
 1              2              1              0                2                3
 2              4              3              0                4                7
 3              3              2              4                7                9
 4              4              1              8                12               13
Average waiting time = 3
Average turn around time = 6.25

...Program finished with exit code 0
Press ENTER to exit console.
```

SJF PREEMPTIVE CPU SCHEDULING

```
#include <bits/stdc++.h>
using namespace std;

struct Process
{
    int pid, bt, art;
};

void findWaitingTime(Process proc[], int n, int wt[])
{
    int rt[n];
    for (int i = 0; i < n; i++)
        rt[i] = proc[i].bt;
    int complete = 0, t = 0, minm = INT_MAX;
    int shortest = 0, finish_time;
    bool check = false;
    while (complete != n)
    {
        for (int j = 0; j < n; j++)
        {
            if ((proc[j].art <= t) &&
                (rt[j] < minm) && rt[j] > 0)
            {
                minm = rt[j];
                shortest = j;
                check = true;
            }
        }
        if (check == false)
        {
            t++;
            continue;
        }
        rt[shortest]--;
        minm = rt[shortest];
        if (minm == 0)
            minm = INT_MAX;
        if (rt[shortest] == 0)
        {
            complete++;
            check = false;
            finish_time = t + 1;
            wt[shortest] = finish_time - proc[shortest].bt - proc[shortest].art;
```

```
            if (wt[shortest] < 0)
                wt[shortest] = 0;
        }
        t++;
    }
}
void findTurnAroundTime(Process proc[], int n, int wt[], int tat[])
{
    for (int i = 0; i < n; i++)
        tat[i] = proc[i].bt + wt[i];
}
void findavgTime(Process proc[], int n)
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;
    findWaitingTime(proc, n, wt);
    findTurnAroundTime(proc, n, wt, tat);
    cout << "Processes "
         << " Burst time "
         << " Waiting time "
         << " Turn around time\n";
    for (int i = 0; i < n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << " " << proc[i].pid << "\t\t"
             << proc[i].bt << "\t\t " << wt[i]
             << "\t\t " << tat[i] << endl;
    }
    cout << "\nAverage waiting time = "
         << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "
         << (float)total_tat / (float)n;
}
int main()
{
    int n;
    cout << "Enter number of Process:  ";
    cin >> n;
    Process proc[n];
    for (int i = 0; i < n; i++)
    {
        cout << "...Process " << i + 1 << "...\n";
        cout << "Enter Process Id: ";
        cin >> proc[i].pid;
        cout << "Enter Arrival Time: ";
```

```
        cin >> proc[i].art;
        cout << "Enter Burst Time: ";
        cin >> proc[i].bt;
    }
    cout << "Before Arrange...\n";
    cout << "Process ID\tArrival Time\tBurst Time\n";
    for (int i = 0; i < n; i++)
        cout << proc[i].pid << "\t\t" << proc[i].art << "\t\t" << proc[i].bt <<
"\n";
    findavgTime(proc, n);
    return 0;
}
```

OUTPUT

```
Enter number of Process:  3
...Process 1...
Enter Process Id: 1
Enter Arrival Time: 3
Enter Burst Time: 2
...Process 2...
Enter Process Id: 2
Enter Arrival Time: 1
Enter Burst Time: 3
...Process 3...
Enter Process Id: 3
Enter Arrival Time: 5
Enter Burst Time: 3
Before Arrange...
Process ID      Arrival Time    Burst Time
1               3               2
2               1               3
3               5               3
Processes  Burst time  Waiting time  Turn around time
 1              2               1               3
 2              3               0               3
 3              3               1               4

Average waiting time = 0.666667
Average turn around time = 3.33333

...Program finished with exit code 0
Press ENTER to exit console.
```

SJF NON-PREEMPTIVE CPU SCHEDULING

```
#include <iostream>
using namespace std;
int mat[10][6];

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

void arrangeArrival(int num, int mat[][6])
{
    for (int i = 0; i < num; i++)
    {
        for (int j = 0; j < num - i - 1; j++)
        {
            if (mat[j][1] > mat[j + 1][1])
            {
                for (int k = 0; k < 5; k++)
                {
                    swap(mat[j][k], mat[j + 1][k]);
                }
            }
        }
    }
}

void completionTime(int num, int mat[][6])
{
    int temp, val;
    mat[0][3] = mat[0][1] + mat[0][2];
    mat[0][5] = mat[0][3] - mat[0][1];
    mat[0][4] = mat[0][5] - mat[0][2];

    for (int i = 1; i < num; i++)
    {
        temp = mat[i - 1][3];
        int low = mat[i][2];
        for (int j = i; j < num; j++)
        {
```

```
            if (temp >= mat[j][1] && low >= mat[j][2])
            {
                low = mat[j][2];
                val = j;
            }
        }
        mat[val][3] = temp + mat[val][2];
        mat[val][5] = mat[val][3] - mat[val][1];
        mat[val][4] = mat[val][5] - mat[val][2];
        for (int k = 0; k < 6; k++)
        {
            swap(mat[val][k], mat[i][k]);
        }
    }
}

int main()
{
    int num, temp;

    cout << "Enter number of Process: ";
    cin >> num;

    cout << "...Enter the process ID...\n";
    for (int i = 0; i < num; i++)
    {
        cout << "...Process " << i + 1 << "...\n";
        cout << "Enter Process Id: ";
        cin >> mat[i][0];
        cout << "Enter Arrival Time: ";
        cin >> mat[i][1];
        cout << "Enter Burst Time: ";
        cin >> mat[i][2];
    }

    cout << "Before Arrange...\n";
    cout << "Process ID\tArrival Time\tBurst Time\n";
    for (int i = 0; i < num; i++)
    {
        cout << mat[i][0] << "\t\t" << mat[i][1] << "\t\t" << mat[i][2] << "\n";
    }

    arrangeArrival(num, mat);
    completionTime(num, mat);
    cout << "Final Result...\n";
```

```
    cout << "Process ID\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n";
    for (int i = 0; i < num; i++)
    {
        cout << mat[i][0] << "\t\t" << mat[i][1] << "\t\t" << mat[i][2] << "\t\t" <<
mat[i][4] << "\t\t" << mat[i][5] << "\n";
    }
    int avgwaiting = 0, avgturnaround = 0;
    for (int i = 0; i < num; i++)
    {
        avgwaiting += mat[i][4], avgturnaround += mat[i][5];
    }
    cout << "Average waiting time = " << (float)avgwaiting / (float)num <<
"\nAverage turn around time = " << (float)avgturnaround / (float)num;
}
```

OUTPUT

```
Enter number of Process: 3
...Enter the process ID...
...Process 1...
Enter Process Id: 1
Enter Arrival Time: 2
Enter Burst Time: 4
...Process 2...
Enter Process Id: 2
Enter Arrival Time: 4
Enter Burst Time: 5
...Process 3...
Enter Process Id: 3
Enter Arrival Time: 5
Enter Burst Time: 7
Before Arrange...
Process ID      Arrival Time    Burst Time
1               2               4
2               4               5
3               5               7
Final Result...
Process ID      Arrival Time    Burst Time      Waiting Time    Turnaround Time
1               2               4               0               4
2               4               5               2               7
3               5               7               6               13
Average waiting time = 2.66667
Average turn around time = 8

...Program finished with exit code 0
Press ENTER to exit console.
```

SRTF CPU SCHEDULING

```
#include <bits/stdc++.h>
using namespace std;
struct Process
{
    int pid;
    int bt;
    int art;
};
void findWaitingTime(Process proc[], int n, int wt[])
{
    int rt[n];
    for (int i = 0; i < n; i++)
        rt[i] = proc[i].bt;
    int complete = 0, t = 0, minm = INT_MAX;
    int shortest = 0, finish_time;
    bool check = false;

    while (complete != n)
    {

        for (int j = 0; j < n; j++)
        {
            if ((proc[j].art <= t) &&
                (rt[j] < minm) && rt[j] > 0)
            {
                minm = rt[j];
                shortest = j;
                check = true;
            }
        }

        if (check == false)
        {
            t++;
            continue;
        }

        rt[shortest]--;

        minm = rt[shortest];
        if (minm == 0)
            minm = INT_MAX;
```

```
        if (rt[shortest] == 0)
        {

            complete++;
            check = false;

            finish_time = t + 1;

            wt[shortest] = finish_time -
                           proc[shortest].bt -
                           proc[shortest].art;

            if (wt[shortest] < 0)
                wt[shortest] = 0;
        }

        t++;
    }
}
void findTurnAroundTime(Process proc[], int n, int wt[], int tat[])
{
    for (int i = 0; i < n; i++)
        tat[i] = proc[i].bt + wt[i];
}
void findavgTime(Process proc[], int n)
{
    int wt[n], tat[n], total_wt = 0,
                       total_tat = 0;

    findWaitingTime(proc, n, wt);

    findTurnAroundTime(proc, n, wt, tat);

    cout << "Processes "
         << " Burst time "
         << " Waiting time "
         << " Turn around time\n";

    for (int i = 0; i < n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << " " << proc[i].pid << "\t\t"
             << proc[i].bt << "\t\t " << wt[i]
             << "\t\t " << tat[i] << endl;
```

```
    }

    cout << "\nAverage waiting time = "
         << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "
         << (float)total_tat / (float)n;
}

int main()
{
   // Process proc[] = {{1, 6, 1}, {2, 8, 1}, {3, 7, 2}, {4, 3, 3}};
   // int n = sizeof(proc) / sizeof(proc[0]);

    findavgTime(proc, n);
    return 0;
}
```

OUTPUT

```
Processes  Burst time  Waiting time  Turn around time
 1              6              3              9
 2              8              16             24
 3              7              8              15
 4              3              0              3

Average waiting time = 6.75
Average turn around time = 12.75

...Program finished with exit code 0
Press ENTER to exit console.
```

PRIORITY NON PREEMPTIVE CPU SCHEDULING

```
#include <iostream>
using namespace std;
int main()
{
    int bt[20], p[20], wt[20], tat[20], pr[20], i, j, n, total = 0, pos, temp,
avg_wt, avg_tat;
    cout << "Enter Total Number of Process:";
    cin >> n;
    cout << "\nEnter Burst Time and Priority\n";
    for (i = 0; i < n; i++)
    {
        cout << "\nP[" << i + 1 << "]\n";
        cout << "Burst Time:";
        cin >> bt[i];
        cout << "Priority:";
        cin >> pr[i];
        p[i] = i + 1; //contains process number
    }
    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; //average waiting time
    total = 0;
    cout << "\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time";
    for (i = 0; i < n; i++)
    {
        tat[i] = bt[i] + wt[i]; //calculate turnaround time
        total += tat[i];
        cout << "\nP[" << p[i] << "]\t\t  " << bt[i] << "\t\t    " << wt[i] <<
"\t\t\t" << tat[i];
    }
    avg_tat = total / n; //average turnaround time
    cout << "\n\nAverage Waiting Time=" << avg_wt;
    cout << "\nAverage Turnaround Time=" << avg_tat;
    return 0;
}
```

OUTPUT

```
Enter Total Number of Process:3

Enter Burst Time and Priority

P[1]
Burst Time:4
Priority:2

P[2]
Burst Time:6
Priority:2

P[3]
Burst Time:8
Priority:1

Process      Burst Time          Waiting Time     Turnaround Time
P[3]                8                    0                    8
P[2]                6                    8                    14
P[1]                4                    14                   18

Average Waiting Time=7
Average Turnaround Time=13

...Program finished with exit code 0
Press ENTER to exit console.
```

PRIORITY PREEMPTIVE CPU SCHEDULING

```
#include <iostream>
using namespace std;
int main()
{
    int bt[20], p[20], wt[20], tat[20], pr[20], i, j, n, total = 0, pos, temp,
avg_wt, avg_tat;
    cout << "Enter Total Number of Process:";
    cin >> n;
    cout << "\nEnter Burst Time and Priority\n";
    for (i = 0; i < n; i++)
    {
        cout << "\nP[" << i + 1 << "]";
        cout << "Burst Time:";
        cin >> bt[i];
        cout << "Priority:";
        cin >> 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;
    cout << "\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time";
    for (i = 0; i < n; i++)
    {
        tat[i] = bt[i] + wt[i];
        total += tat[i];
        cout << "\nP[" << p[i] << "]\t\t  " << bt[i] << "\t\t    " << wt[i] <<
"\t\t\t" << tat[i];
    }
    avg_tat = total / n;
    cout << "\n\nAverage Waiting Time=" << avg_wt;
    cout << "\nAverage Turnaround Time=" << avg_tat;
    return 0;
}
```

OUTPUT

```
Enter Total Number of Process:3

Enter Burst Time and Priority

P[1]Burst Time:4
Priority:5

P[2]Burst Time:2
Priority:5

P[3]Burst Time:2
Priority:4

Process     Burst Time          Waiting Time    Turnaround Time
P[3]              2                   0                2
P[2]              2                   2                4
P[1]              4                   4                8

Average Waiting Time=2
Average Turnaround Time=4
```

ROUND ROBIN CPU SCHEDULING

```
#include <iostream>
using namespace std;
void findWaitingTime(int processes[], int n, int bt[], int wt[], int quantum)
{
    int rem_bt[n], t=0;
    for (int i = 0; i < n; i++)
        rem_bt[i] = bt[i];
    while (1)
    {
        bool done = true;
        for (int i = 0; i < n; i++)
        {
            if (rem_bt[i] > 0)
            {
                done = false;
                if (rem_bt[i] > quantum)
                {
                    t += quantum;
                    rem_bt[i] -= quantum;
                }
                else
                {
                    t = t + rem_bt[i];
                    wt[i] = t - bt[i];
                    rem_bt[i] = 0;
                }
            }
        }
        if (done == true)
            break;
    }
}
void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[])
{
    for (int i = 0; i < n; i++)
        tat[i] = bt[i] + wt[i];
}
void findavgTime(int processes[], int n, int bt[], int quantum)
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;
    findWaitingTime(processes, n, bt, wt, quantum);
    findTurnAroundTime(processes, n, bt, wt, tat);
    cout << "Processes " << " Burst time " << " Waiting time "
         << " Turn around time\n";
```

```
    for (int i = 0; i < n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << " " << i + 1 << "\t\t" << bt[i] << "\t " << wt[i] << "\t\t " <<
tat[i] << endl;
    }
    cout << "Average waiting time = " << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = " << (float)total_tat / (float)n;
}
int main()
{
    int n;
    cout << "Enter Total Number of Process:";
    cin >> n;
    int processes[n], burst_time[n];
    for (int i = 0; i < n; i++)
    {
        processes[i] = i + 1;
        cout << "Enter the burst time of " << i << "th process ";
        cin >> burst_time[i];
    }
    int quantum;
    cout << "Enter the quantum value: " << endl;
    cin >> quantum;
    findavgTime(processes, n, burst_time, quantum);
    return 0;
}
```

OUTPUT

```
Enter Total Number of Process:3
Enter the burst time of 0th process 7
Enter the burst time of 1th process 8
Enter the burst time of 2th process 9
Enter the quantum value:
3
Processes  Burst time  Waiting time  Turn around time
 1              7         12              19
 2              8         13              21
 3              9         15              24
Average waiting time = 13.3333
Average turn around time = 21.3333
```

BEST FIT

```
#include <bits/stdc++.h>
using namespace std;
// Function to allocate memory to blocks as per Best fit
// algorithm
void bestFit(int blockSize[], int m, int processSize[], int n)
{
    // Stores block id of the block allocated to a
    // process
    int allocation[n];

    // Initially no block is assigned to any process
    memset(allocation, -1, sizeof(allocation));

    // pick each process and find suitable blocks
    // according to its size ad assign to it
    for (int i = 0; i < n; i++)
    {
        // Find the best fit block for current process
        int bestIdx = -1;
        for (int j = 0; j < m; j++)
        {
            if (blockSize[j] >= processSize[i])
            {
                if (bestIdx == -1)
                    bestIdx = j;
                else if (blockSize[bestIdx] > blockSize[j])
                    bestIdx = j;
            }
        }
        // If we could find a block for current process
        if (bestIdx != -1)
        {
            // allocate block j to p[i] process
            allocation[i] = bestIdx;
            // Reduce available memory in this block.
            blockSize[bestIdx] -= processSize[i];
        }
    }
    cout << "\nProcess No.\tProcess Size\tBlock no.\n";
    for (int i = 0; i < n; i++)
    {
        cout << "   " << i + 1 << "\t\t" << processSize[i] << "\t\t";
        if (allocation[i] != -1)
            cout << allocation[i] + 1;
```

```
        else
            cout << "Not Allocated";
        cout << endl;
    }
}
int main()
{
    int m, n;
    cout << "Enter the array size of blockSize: ";
    cin >> m;
    int blockSize[m];
    cout << "Enter the elements of blockSize array: ";
    for (int i = 0; i < m; i++)
        cin >> blockSize[i];
    cout << "Enter the number of processes: ";
    cin >> n;
    int processSize[n];
    cout << "Enter the processes: ";
    for (int i = 0; i < n; i++)
        cin >> processSize[i];
    bestFit(blockSize, m, processSize, n);
    return 0;
}
```

OUTPUT

```
Enter the array size of blockSize: 3
Enter the elements of blockSize array: 2 3 4
Enter the number of processes: 2
Enter the processes: 1 4

Process No.     Process Size    Block no.
   1            1               1
   2            4               3


...Program finished with exit code 0
Press ENTER to exit console.
```

WORST FIT

```
#include <iostream>
#include <climits>
using namespace std;

void worstfit(int blockSize[], int n, int processSize[], int m)
{
    int allocation[m];
    for (int i = 0; i < m; i++)
    {
        allocation[i] = -1;
    }
    bool blocks[n];
    for (int i = 0; i < n; i++)
    {
        blocks[i] = false;
    }
    for (int i = 0; i < m; i++)
    {
        int max = INT_MIN;
        int y = -1;
        for (int j = 0; j < n; j++)
        {
            if (blocks[j] || blockSize[j] < processSize[i])
            {
                continue;
            }
            if (blockSize[j] - processSize[i] > max)
            {
                max = blockSize[j] - processSize[i];
                y = j;
            }
        }
        blocks[y] = true;
        allocation[i] = y;
    }
    cout << "\nProcess No.\tProcess Size\tBlock no.\n";
    for (int i = 0; i < m; i++)
    {
        cout << "   " << i + 1 << "\t\t" << processSize[i] << "\t\t";
        if (allocation[i] != -1)
            cout << allocation[i] + 1;
        else
            cout << "Not Allocated";
        cout << endl;
```

```
    }
}

int main()
{
    int m, n;
    cout << "Enter the array size of blockSize: ";
    cin >> m;
    int blockSize[m];
    cout << "Enter the array elements of blockSize: ";
    for (int i = 0; i < m; i++)
        cin >> blockSize[i];
    cout << "Enter the number of processes: ";
    cin >> n;
    int processSize[n];
    cout << "Enter the processes: ";
    for (int i = 0; i < n; i++)
        cin >> processSize[i];
    worstfit(blockSize, m, processSize, n);
}
```

OUTPUT

```
Enter the array size of blockSize: 3
Enter the array elements of blockSize: 2 3 4
Enter the number of processes: 2
Enter the processes: 2 4

Process No.     Process Size    Block no.
   1            2               3
   2            4               Not Allocated


...Program finished with exit code 0
Press ENTER to exit console.
```

FIRST FIT

```
#include <iostream>
#include <climits>
using namespace std;
void firstfit(int blockSize[], int n, int processSize[], int m)
{
    int allocation[m];
    for (int i = 0; i < m; i++)
    {
        allocation[i] = -1;
    }
    bool blocks[n];
    for (int i = 0; i < n; i++)
    {
        blocks[i] = false;
    }
    for (int i = 0; i < m; i++)
    {
        int y = -1;
        for (int j = 0; j < n; j++)
        {
            if (blocks[j] || blockSize[j] < processSize[i])
            {
                continue;
            }
            y = j;
            break;
        }
        blocks[y] = true;
        allocation[i] = y;
    }
    cout << "\nProcess No.\tProcess Size\tBlock no.\n";
    for (int i = 0; i < m; i++)
    {
        cout << "   " << i + 1 << "\t\t" << processSize[i] << "\t\t";
        if (allocation[i] != -1)
            cout << allocation[i] + 1;
        else
            cout << "Not Allocated";
        cout << endl;
    }
}
int main()
{
```

```
    int m, n;
    cout << "Enter the array size of blockSize: ";
    cin >> m;
    int blockSize[m];
    cout << "Enter the array elements of blockSize: ";
    for (int i = 0; i < m; i++)
        cin >> blockSize[i];
    cout << "Enter the number of processes: ";
    cin >> n;
    int processSize[n];
    cout << "Enter the processes: ";
    for (int i = 0; i < n; i++)
        cin >> processSize[i];
    firstfit(blockSize, m, processSize, n);
}
```

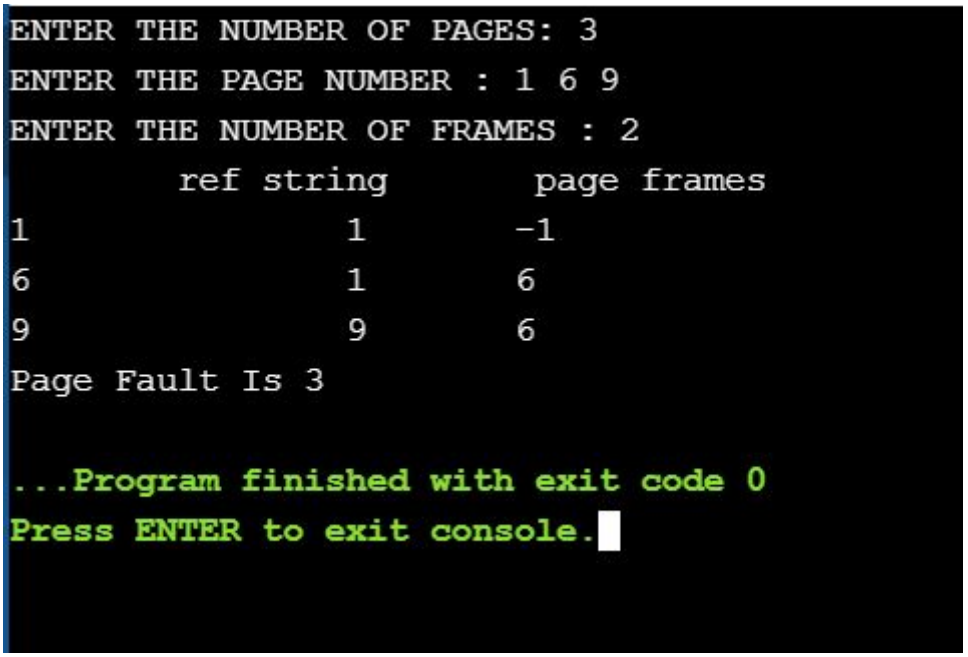
OUTPUT

```
Enter the array size of blockSize: 3
Enter the array elements of blockSize: 2 3 4
Enter the number of processes: 2
Enter the processes: 4 5

Process No.     Process Size    Block no.
   1            4               3
   2            5               Not Allocated


...Program finished with exit code 0
Press ENTER to exit console.
```

FIFO PAGE REPLACEMENT ALGORITHM

```c
#include <stdio.h>
int main()
{
    int i, j, n, a[50], frame[10], no, k, avail, count = 0;
    printf("ENTER THE NUMBER OF PAGES: ");
    scanf("%d", &n);
    printf("ENTER THE PAGE NUMBER : ");
    for (i = 1; i <= n; i++)
        scanf("%d", &a[i]);
    printf("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


ENTER THE NUMBER OF PAGES: 3
ENTER THE PAGE NUMBER : 1 6 9
ENTER THE NUMBER OF FRAMES : 2
ref string page frames
1 1 -1
6 1 6
9 9 6
Page Fault Is 3
...Program finished with exit code 0
Press ENTER to exit console.

OPTIMAL PAGE REPLACEMENT ALGORITHM

```c
#include <stdio.h>
int main()
{
    int no_of_frames, no_of_pages, frames[10], pages[30],
        temp[10], flag1, flag2, flag3, i, j, k, pos, max, faults = 0;
    printf("Enter number of frames: ");
    scanf("%d", &no_of_frames);

    printf("Enter number of pages: ");
    scanf("%d", &no_of_pages);

    printf("Enter page reference string: ");

    for (i = 0; i < no_of_pages; ++i)
    {
        scanf("%d", &pages[i]);
    }

    for (i = 0; i < no_of_frames; ++i)
    {
        frames[i] = -1;
    }

    for (i = 0; i < no_of_pages; ++i)
    {
        flag1 = flag2 = 0;

        for (j = 0; j < no_of_frames; ++j)
        {
            if (frames[j] == pages[i])
            {
                flag1 = flag2 = 1;
                break;
            }
        }

        if (flag1 == 0)
        {
            for (j = 0; j < no_of_frames; ++j)
            {
                if (frames[j] == -1)
                {
                    faults++;
                    frames[j] = pages[i];
```

```
                flag2 = 1;
                break;
            }
        }
    }

    if (flag2 == 0)
    {
        flag3 = 0;

        for (j = 0; j < no_of_frames; ++j)
        {
            temp[j] = -1;

            for (k = i + 1; k < no_of_pages; ++k)
            {
                if (frames[j] == pages[k])
                {
                    temp[j] = k;
                    break;
                }
            }
        }

        for (j = 0; j < no_of_frames; ++j)
        {
            if (temp[j] == -1)
            {
                pos = j;
                flag3 = 1;
                break;
            }
        }

        if (flag3 == 0)
        {
            max = temp[0];
            pos = 0;

            for (j = 1; j < no_of_frames; ++j)
            {
                if (temp[j] > max)
                {
                    max = temp[j];
                    pos = j;
```

```
                    }
                }
            }
            frames[pos] = pages[i];
            faults++;
        }

        printf("\n");

        for (j = 0; j < no_of_frames; ++j)
        {
            printf("%d\t", frames[j]);
        }
    }

    printf("\n\nTotal Page Faults = %d", faults);

    return 0;
}
```

OUTPUT

```
Enter number of frames: 3
Enter number of pages: 3
Enter page reference string: 1 2 3

1       -1      -1
1       2       -1
1       2       3

Total Page Faults = 3

...Program finished with exit code 0
Press ENTER to exit console.
```

LRU PAGE REPLACEMENT ALGORITHM

```c
#include <stdio.h>
int findLRU(int time[], int n)
{
    int i, minimum = time[0], pos = 0;
    for (i = 1; i < n; ++i)
    {
        if (time[i] < minimum)
        {
            minimum = time[i];
            pos = i;
        }
    }
    return pos;
}
int main()
{
    int no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10],
flag1, flag2, i, j, pos, faults = 0;
    printf("Enter number of frames: ");
    scanf("%d", &no_of_frames);
    printf("Enter number of pages: ");
    scanf("%d", &no_of_pages);
    printf("Enter reference string: ");
    for (i = 0; i < no_of_pages; ++i)
        scanf("%d", &pages[i]);
    for (i = 0; i < no_of_frames; ++i)
        frames[i] = -1;
    for (i = 0; i < no_of_pages; ++i)
    {
        flag1 = flag2 = 0;
        for (j = 0; j < no_of_frames; ++j)
        {
            if (frames[j] == pages[i])
            {
                counter++;
                time[j] = counter;
                flag1 = flag2 = 1;
                break;
            }
        }

        if (flag1 == 0)
        {
            for (j = 0; j < no_of_frames; ++j)
```

```
            {
                if (frames[j] == -1)
                {
                    counter++;
                    faults++;
                    frames[j] = pages[i];
                    time[j] = counter;
                    flag2 = 1;
                    break;
                }
            }
        }
        if (flag2 == 0)
        {
            pos = findLRU(time, no_of_frames);
            counter++;
            faults++;
            frames[pos] = pages[i];
            time[pos] = counter;
        }
        printf("\n");
        for (j = 0; j < no_of_frames; ++j)
            printf("%d\t", frames[j]);
    }
    printf("\n\nTotal Page Faults = %d", faults);
    return 0;
}
```

OUTPUT

```
Enter number of frames: 3
Enter number of pages: 3
Enter reference string: 5 7 3

5       -1      -1
5       7       -1
5       7       3

Total Page Faults = 3

...Program finished with exit code 0
Press ENTER to exit console.
```

MRU PAGE REPLACEMENT ALGORITHM

```
#include <stdio.h>
int findMRU(int time[], int n)
{
    int i, maximum = time[0], pos = 0;
    for (i = 1; i < n; ++i)
    {
        if (time[i] > maximum)
        {
            maximum = time[i];
            pos = i;
        }
    }
    return pos;
}
int main()
{
    int no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10],
flag1, flag2, i, j, pos, faults = 0;
    printf("Enter number of frames: ");
    scanf("%d", &no_of_frames);
    printf("Enter number of pages: ");
    scanf("%d", &no_of_pages);
    printf("Enter reference string: ");
    for (i = 0; i < no_of_pages; ++i)
        scanf("%d", &pages[i]);
    for (i = 0; i < no_of_frames; ++i)
        frames[i] = -1;
    for (i = 0; i < no_of_pages; ++i)
    {
        flag1 = flag2 = 0;
        for (j = 0; j < no_of_frames; ++j)
        {
            if (frames[j] == pages[i])
            {
                counter++;
                time[j] = counter;
                flag1 = flag2 = 1;
                break;
            }
        }
        if (flag1 == 0)
        {
            for (j = 0; j < no_of_frames; ++j)
            {
```

```
                if (frames[j] == -1)
                {
                    counter++;
                    faults++;
                    frames[j] = pages[i];
                    time[j] = counter;
                    flag2 = 1;
                    break;
                }
            }
        }
        if (flag2 == 0)
        {
            pos = findMRU(time, no_of_frames);
            counter++;
            faults++;
            frames[pos] = pages[i];
            time[pos] = counter;
        }
        printf("\n");
        for (j = 0; j < no_of_frames; ++j)
            printf("%d\t", frames[j]);
    }
    printf("\n\nTotal Page Faults = %d", faults);
    return 0;
}
```

OUTPUT

```
Enter number of frames: 3
Enter number of pages: 3
Enter reference string: 4 2 1 6

4       -1      -1
4       2       -1
4       2       1

Total Page Faults = 3

...Program finished with exit code 0
Press ENTER to exit console.
```

PRODUCER CONSUMER PROBLEM

```c
#include <stdio.h>
#include <stdlib.h>

int mutex = 1, full = 0, empty = 3, x = 0;

int main()
{
    int n;
    void producer();
    void consumer();
    int wait(int);
    int signal(int);
    printf("\n1.Producer\n2.Consumer\n3.Exit");
    while (1)
    {
        printf("\nEnter your choice:");
        scanf("%d", &n);
        switch (n)
        {
        case 1:
            if ((mutex == 1) && (empty != 0))
                producer();
            else
                printf("Buffer is full!!");
            break;
        case 2:
            if ((mutex == 1) && (full != 0))
                consumer();
            else
                printf("Buffer is empty!!");
            break;
        case 3:
            exit(0);
            break;
        }
    }

    return 0;
}

int wait(int s)
{
    return (--s);
}
```

```
int signal(int s)
{
    return (++s);
}
void producer()
{
    mutex = wait(mutex);
    full = signal(full);
    empty = wait(empty);
    x++;
    printf("\nProducer produces the item %d", x);
    mutex = signal(mutex);
}
void consumer()
{
    mutex = wait(mutex);
    full = wait(full);
    empty = signal(empty);
    printf("\nConsumer consumes item %d", x);
    x--;
    mutex = signal(mutex);
}
```

OUTPUT

```
1.Producer
2.Consumer
3.Exit
Enter your choice:2
Buffer is empty!!
Enter your choice:1

Producer produces the item 1
Enter your choice:1

Producer produces the item 2
Enter your choice:3


...Program finished with exit code 0
Press ENTER to exit console.
```

SSTF DISK SCHEDULING ALGORITHM

```c
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <stdlib.h>
int main()
{
    int queue[100], t[100], head, seek = 0, n, i, j, temp;
    float avg;
    printf("*** SSTF Disk Scheduling Algorithm ***\n");
    printf("Enter the size of Queue\t");
    scanf("%d", &n);
    printf("Enter the Queue\t");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &queue[i]);
    }
    printf("Enter the initial head position\t");
    scanf("%d", &head);
    for (i = 1; i < n; i++)
        t[i] = abs(head - queue[i]);
    for (i = 0; i < n; i++)
    {
        for (j = i + 1; j < n; j++)
        {
            if (t[i] > t[j])
            {
                temp = t[i];
                t[i] = t[j];
                t[j] = temp;
                temp = queue[i];
                queue[i] = queue[j];
                queue[j] = temp;
            }
        }
    }
    for (i = 1; i < n - 1; i++)
    {
        seek = seek + abs(head - queue[i]);
        head = queue[i];
    }
    printf("\nTotal Seek Time is%d\t", seek);
    avg = seek / (float)n;
    printf("\nAverage Seek Time is %f\t", avg);
    getch();
```

```
}
```

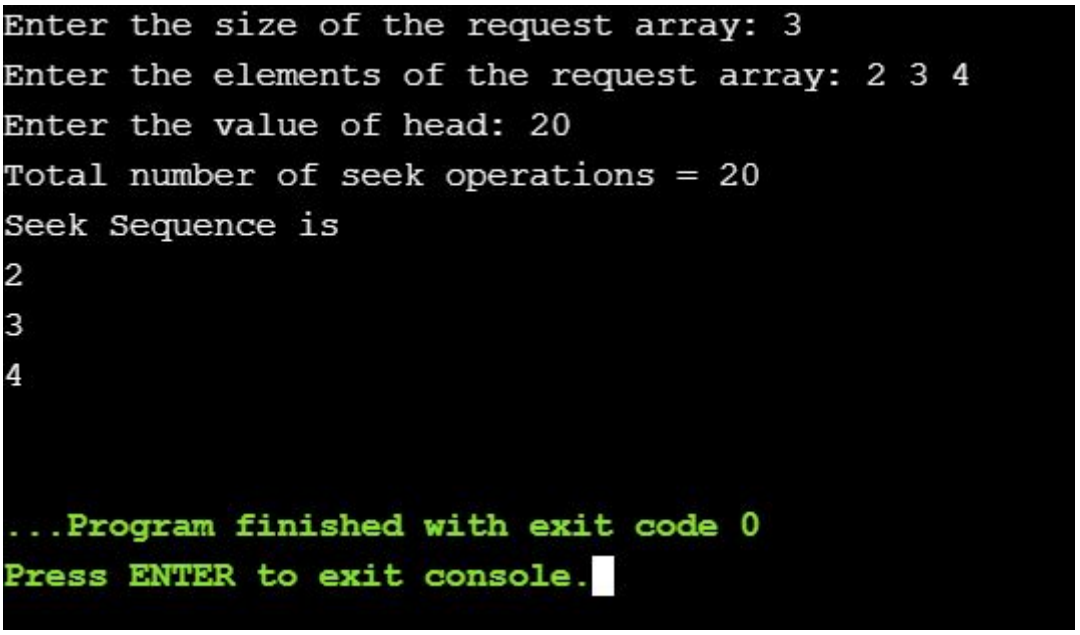
OUTPUT

```
*** SSTF Disk Scheduling Algorithm ***
Enter the size of Queue 3
Enter the Queue 4 6 8
Enter the initial head position 32

Total Seek Time is26
Average Seek Time is 8.666667

...Program finished with exit code 0
Press ENTER to exit console.
```

FCFS DISK SCHEDULING ALGORITHM

```
#include <bits/stdc++.h>
using namespace std;
void FCFS(int arr[], int head, int size)
{
    int seek_count = 0;
    int distance, cur_track;
    for (int i = 0; i < size; i++)
    {
        cur_track = arr[i];
        distance = abs(cur_track - head);
        seek_count += distance;
        head = cur_track;
    }
    cout << "Total number of seek operations = "
         << seek_count << endl;
    cout << "Seek Sequence is" << endl;
    for (int i = 0; i < size; i++)
        cout << arr[i] << endl;
}
int main()
{
    int size;
cout<<”Enter the size of the request array: “<<endl;
cin>>size;
    // request array
    int arr[size];
for(int i=0;i<size;i++)
cin>>arr[i];
    int head;
cout<<”Enter the value of head: “<<endl;
cin>>head;
    FCFS(arr, head, size);
    return 0;
}
```

OUTPUT


Enter the size of the request array: 3
Enter the elements of the request array: 2 3 4
Enter the value of head: 20
Total number of seek operations = 20
Seek Sequence is
2
3
4
...Program finished with exit code 0
Press ENTER to exit console.

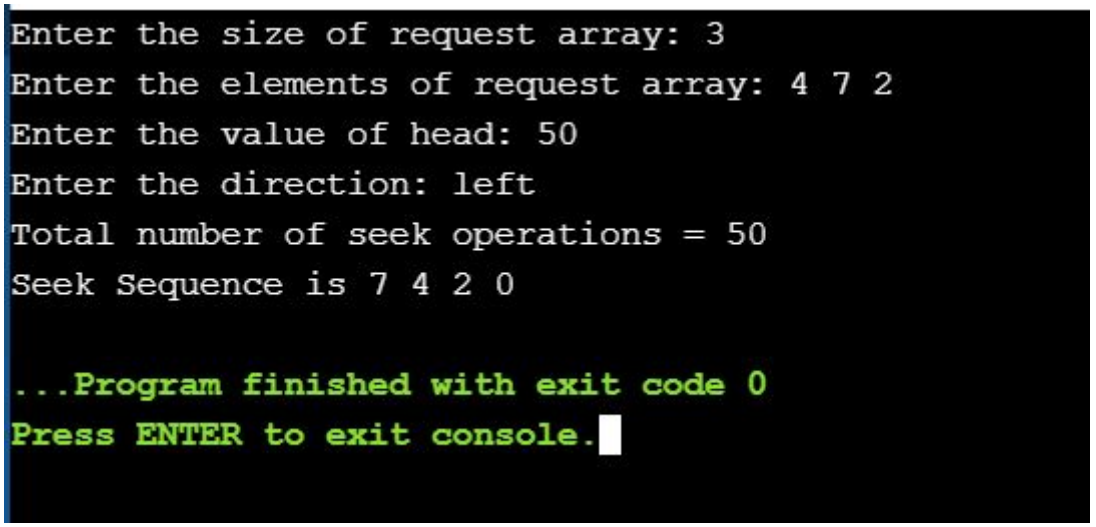
SCAN Disk Scheduling Algorithm

```
#include <bits/stdc++.h>
using namespace std;
int disk_size = 200;
void SCAN(int *arr, int size, int head, string direction)
{
    int seek_count = 0;
    int distance, cur_track;
    vector<int> left, right;
    vector<int> seek_sequence;

    // appending end values
    // which has to be visited
    // before reversing the direction
    if (direction == "left")
        left.push_back(0);
    else if (direction == "right")
        right.push_back(disk_size - 1);

    for (int i = 0; i < size; i++)
    {
        if (arr[i] < head)
            left.push_back(arr[i]);
        if (arr[i] > head)
            right.push_back(arr[i]);
    }

    // sorting left and right vectors
    std::sort(left.begin(), left.end());
    std::sort(right.begin(), right.end());

    // run the while loop two times.
    // one by one scanning right
    // and left of the head
    int run = 2;
    while (run--)
    {
        if (direction == "left")
        {
            for (int i = left.size() - 1; i >= 0; i--)
            {
                cur_track = left[i];
                // appending current track to seek sequence
                seek_sequence.push_back(cur_track);
```

```
                // calculate absolute distance
                distance = abs(cur_track - head);
                // increase the total count
                seek_count += distance;
                // accessed track is now the new head
                head = cur_track;
            }
            direction = "right";
        }
        else if (direction == "right")
        {
            for (int i = 0; i < right.size(); i++)
            {
                cur_track = right[i];
                // appending current track to seek sequence
                seek_sequence.push_back(cur_track);
                // calculate absolute distance
                distance = abs(cur_track - head);
                // increase the total count
                seek_count += distance;
                // accessed track is now new head
                head = cur_track;
            }
            direction = "left";
        }
    }
    cout << "Total number of seek operations = "
         << seek_count << endl;
    cout << "Seek Sequence is ";
    for (int i = 0; i < seek_sequence.size(); i++)
        cout << seek_sequence[i] << " ";
}
int main()
{
    int size;
    cout << "Enter the size of request array: ";
    cin >> size;
    int arr[size];
    cout << "Enter the elements of request array: ";
    for (int i = 0; i < size; i++)
        cin >> arr[i];
    cout << "Enter the value of head: ";
    int head;
    cin >> head;
    string direction;
```

```
    cout << "Enter the direction: ";
    cin >> direction;
    SCAN(arr, size, head, direction);
    return 0;
}
```

OUTPUT



C-SCAN Disk Scheduling Algorithm

```cpp
#include <bits/stdc++.h>
using namespace std;
int disk_size = 200;
void CSCAN(int arr[], int size, int head)
{
   int seek_count = 0;
   int distance, cur_track;
   vector<int> left, right;
   vector<int> seek_sequence;

   left.push_back(0);
   right.push_back(disk_size - 1);

   for (int i = 0; i < size; i++)
   {
      if (arr[i] < head)
         left.push_back(arr[i]);
      if (arr[i] > head)
         right.push_back(arr[i]);
   }
   std::sort(left.begin(), left.end());
   std::sort(right.begin(), right.end());
   for (int i = 0; i < right.size(); i++)
   {
      cur_track = right[i];
      // appending current track to seek sequence
      seek_sequence.push_back(cur_track);
      // calculate absolute distance
      distance = abs(cur_track - head);
      // increase the total count
      seek_count += distance;
      // accessed track is now new head
      head = cur_track;
   }
   head = 0;
   for (int i = 0; i < left.size(); i++)
   {
      cur_track = left[i];
      // appending current track to seek sequence
      seek_sequence.push_back(cur_track);
      // calculate absolute distance
      distance = abs(cur_track - head);
      // increase the total count
      seek_count += distance;
```

```
      // accessed track is now the new head
      head = cur_track;
   }
   cout << "Total number of seek operations = "
        << seek_count << endl;
   cout << "Seek Sequence is" << endl;
   for (int i = 0; i < seek_sequence.size(); i++)
      cout << seek_sequence[i] << " ";
}
int main()
{
   int size, head;
   cout << "Enter the size of request array: ";
   cin >> size;
   int arr[size];
   cout << "Enter the elements of request array: ";
   for (int i = 0; i < size; i++)
      cin >> arr[i];
   cout << "Enter the value of head: ";
   cin >> head;
   CSCAN(arr, size, head);
   return 0;
}
```

OUTPUT

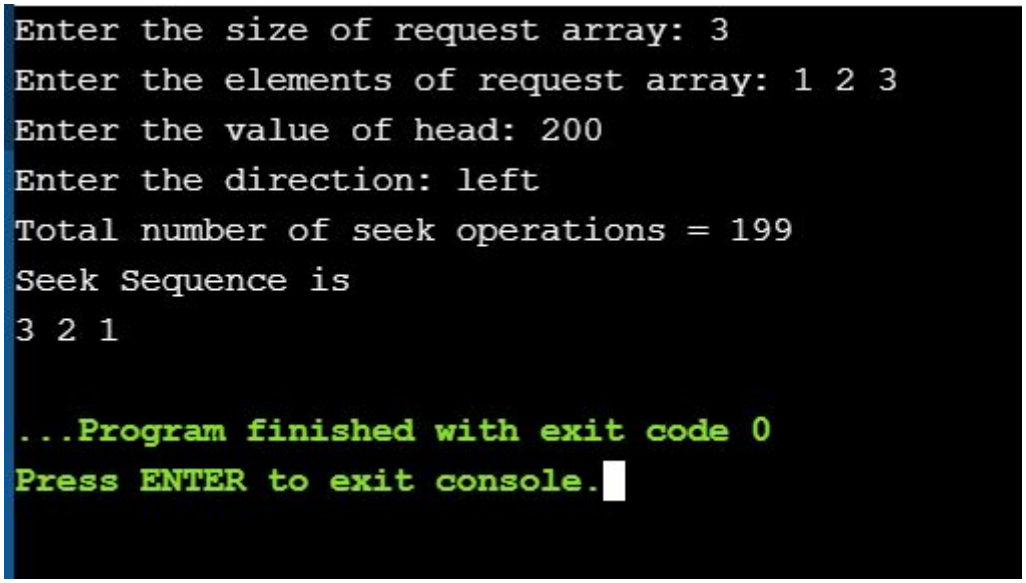
```
Enter the size of request array: 3
Enter the elements of request array: 9 87 6
Enter the value of head: 300
Total number of seek operations = 188
Seek Sequence is
199 0 6 9 87

...Program finished with exit code 0
Press ENTER to exit console.
```

LOOK Disk Scheduling Algorithm

```cpp
#include <bits/stdc++.h>
using namespace std;
int disk_size = 200;
void LOOK(int arr[], int size, int head, string direction)
{
   int seek_count = 0;
   int distance, cur_track;
   vector<int> left, right;
   vector<int> seek_sequence;

   for (int i = 0; i < size; i++)
   {
      if (arr[i] < head)
         left.push_back(arr[i]);
      if (arr[i] > head)
         right.push_back(arr[i]);
   }

   std::sort(left.begin(), left.end());
   std::sort(right.begin(), right.end());

   int run = 2;
   while (run--)
   {
      if (direction == "left")
      {
         for (int i = left.size() - 1; i >= 0; i--)
         {
            cur_track = left[i];

            // appending current track to seek sequence
            seek_sequence.push_back(cur_track);

            // calculate absolute distance
            distance = abs(cur_track - head);

            // increase the total count
            seek_count += distance;

            // accessed track is now the new head
            head = cur_track;
         }
         // reversing the direction
         direction = "right";
```

```
      }
      else if (direction == "right")
      {
         for (int i = 0; i < right.size(); i++)
         {
            cur_track = right[i];
            // appending current track to seek sequence
            seek_sequence.push_back(cur_track);

            // calculate absolute distance
            distance = abs(cur_track - head);

            // increase the total count
            seek_count += distance;

            // accessed track is now new head
            head = cur_track;
         }
         // reversing the direction
         direction = "left";
      }
   }
   cout << "Total number of seek operations = "
        << seek_count << endl;
   cout << "Seek Sequence is" << endl;
   for (int i = 0; i < seek_sequence.size(); i++)
      cout << seek_sequence[i] << " ";
}
int main()
{
   int size, head;
   cout << "Enter the size of request array: ";
   cin >> size;
   int arr[size];
   cout << "Enter the elements of request array: ";
   for (int i = 0; i < size; i++)
      cin >> arr[i];
   cout << "Enter the value of head: ";
   cin >> head;
   cout << "Enter the direction: ";
   string direction;
   cin >> direction;
   LOOK(arr, size, head, direction);
   return 0;
}
```

OUTPUT


Enter the size of request array: 3
Enter the elements of request array: 1 2 3
Enter the value of head: 200
Enter the direction: left
Total number of seek operations = 199
Seek Sequence is
3 2 1
...Program finished with exit code 0
Press ENTER to exit console.

C-LOOK Disk Scheduling Algorithm

```cpp
// C++ implementation of the approach
#include <bits/stdc++.h>
using namespace std;

int disk_size = 200;

// Function to perform C-LOOK on the request
// array starting from the given head
void CLOOK(int arr[], int head, int size)
{
    int seek_count = 0;
    int distance, cur_track;
    vector<int> left, right;
    vector<int> seek_sequence;

    for (int i = 0; i < size; i++)
    {
        if (arr[i] < head)
            left.push_back(arr[i]);
        if (arr[i] > head)
            right.push_back(arr[i]);
    }
    std::sort(left.begin(), left.end());
    std::sort(right.begin(), right.end());

    for (int i = 0; i < right.size(); i++)
    {
        cur_track = right[i];
        // Appending current track to seek sequence
        seek_sequence.push_back(cur_track);
        // Calculate absolute distance
        distance = abs(cur_track - head);
        // Increase the total count
        seek_count += distance;
        // Accessed track is now new head
        head = cur_track;
    }
    seek_count += abs(head - left[0]);
    head = left[0];
    for (int i = 0; i < left.size(); i++)
    {
        cur_track = left[i];
        // Appending current track to seek sequence
        seek_sequence.push_back(cur_track);
```

```
        // Calculate absolute distance
        distance = abs(cur_track - head);
        // Increase the total count
        seek_count += distance;
        // Accessed track is now the new head
        head = cur_track;
    }
    cout << "Total number of seek operations = "
         << seek_count << endl;
    cout << "Seek Sequence is" << endl;
    for (int i = 0; i < seek_sequence.size(); i++)
        cout << seek_sequence[i] << endl;
}
int main()
{
    cout<<”Enter the size of the request array: “<<endl;
    int size;
    cin>>size;
    int arr[size];
    for(int i=0;i<size;i++)
    cin>>arr[i];
    int head;
    cout<<”Enter the value of head” <<endl;
    cin>>head;
    cout << "Initial position of head: " << head << endl;
    CLOOK(arr, head, size);
    return 0;
}
```

OUTPUT

```
Enter the size of the request array: 3
Enter the elements of the request array: 1 2 3
Enter the value of head 50
Initial position of head: 50
Total number of seek operations = 51
Seek Sequence is
1
2
3


...Program finished with exit code 0
Press ENTER to exit console.
```

Dining Philosophers Problem

```c
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include<
#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum + 4) % N
#define RIGHT (phnum + 1) % N

int state[N];
int phil[N] = {0, 1, 2, 3, 4};

sem_t mutex;
sem_t S[N];

void test(int phnum)
{
    if (state[phnum] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING)
    {
        // state that eating
        state[phnum] = EATING;

        sleep(2);

        printf("Philosopher %d takes fork %d and %d\n",
               phnum + 1, LEFT + 1, phnum + 1);

        printf("Philosopher %d is Eating\n", phnum + 1);

        // sem_post(&S[phnum]) has no effect
        // during takefork
        // used to wake up hungry philosophers
        // during putfork
        sem_post(&S[phnum]);
    }
}

// take up chopsticks
void take_fork(int phnum)
{

    sem_wait(&mutex);
```

```
    // state that hungry
    state[phnum] = HUNGRY;

    printf("Philosopher %d is Hungry\n", phnum + 1);

    // eat if neighbours are not eating
    test(phnum);

    sem_post(&mutex);

    // if unable to eat wait to be signalled
    sem_wait(&S[phnum]);

    sleep(1);
}

// put down chopsticks
void put_fork(int phnum)
{

    sem_wait(&mutex);

    // state that thinking
    state[phnum] = THINKING;

    printf("Philosopher %d putting fork %d and %d down\n",
           phnum + 1, LEFT + 1, phnum + 1);
    printf("Philosopher %d is thinking\n", phnum + 1);

    test(LEFT);
    test(RIGHT);

    sem_post(&mutex);
}

void *philospher(void *num)
{

    while (1)
    {

        int *i = num;

        sleep(1);
```

```
        take_fork(*i);

        sleep(0);

        put_fork(*i);
    }
}

int main()
{

    int i;
    pthread_t thread_id[N];

    // initialize the semaphores
    sem_init(&mutex, 0, 1);

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

        sem_init(&S[i], 0, 0);

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

        // create philosopher processes
        pthread_create(&thread_id[i], NULL,
                       philospher, &phil[i]);

        printf("Philosopher %d is thinking\n", i + 1);
    }

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

        pthread_join(thread_id[i], NULL);
}
```

OUTPUT

```
Philosopher 4 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 is Hungry
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 5 is Hungry
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 2 is Hungry
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
```

UNIX SYSTEM CALLS

System calls in Unix are used for file system control, process control, interprocess communication etc. Access to the Unix kernel is only available through these system calls. Generally, system calls are similar to function calls, the only difference is that they remove the control from the user process.

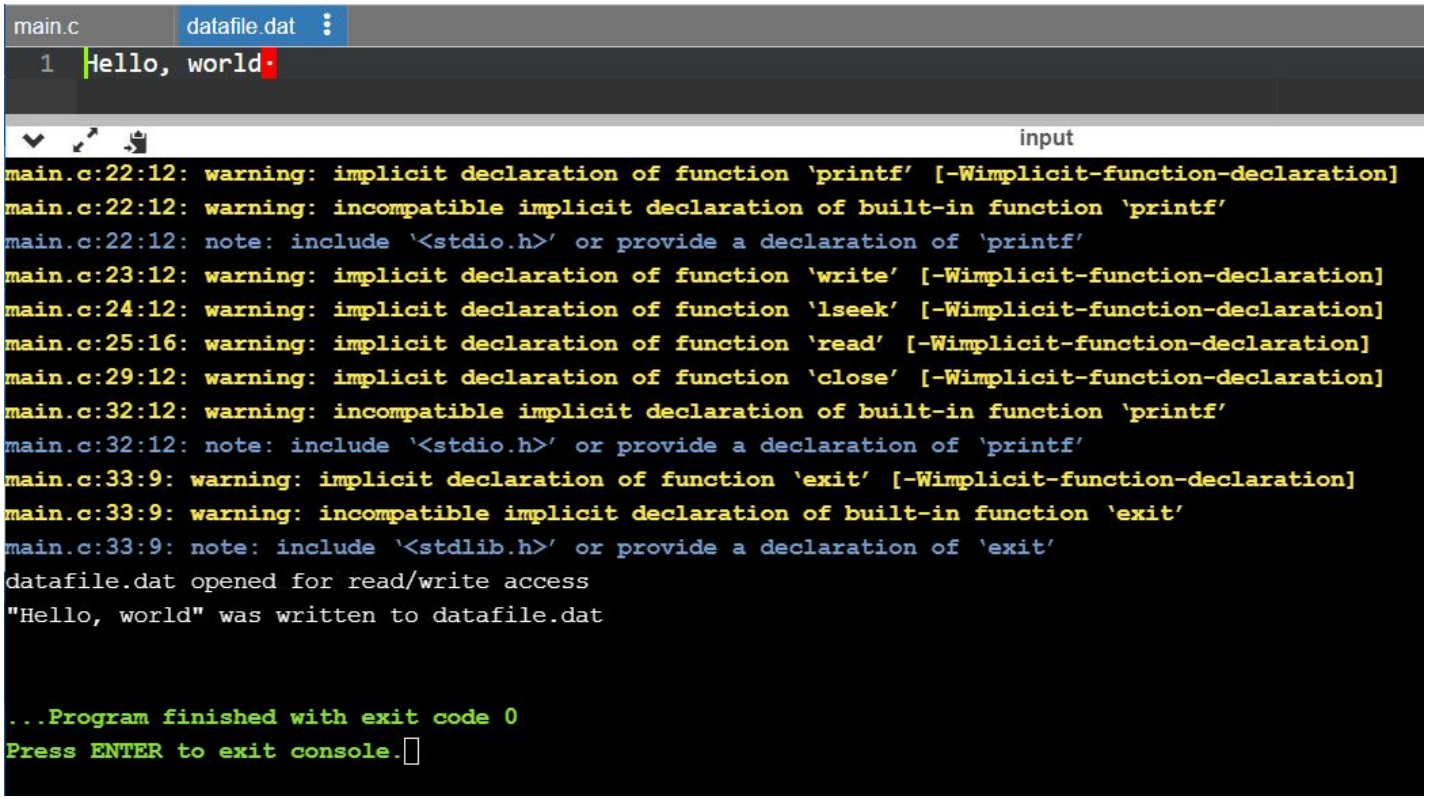
SystemCall Description

access()	This checks if a calling process has access to the required file
chdir()	The chdir command changes the current directory of the system
chmod()	The mode of a file can be changed using this command
chown()	This changes the ownership of a particular file
kill()	This system call sends kill signal to one or more processes
link()	A new file name is linked to an existing file using link system call.
open()	This opens a file for the reading or writing process
pause()	The pause call suspends a file until a particular signal occurs.
stime()	This system call sets the correct time.
times()	Gets the parent and child process times
alarm()	The alarm system call sets the alarm clock of a process
fork()	A new process is created using this command
chroot()	This changes the root directory of a file.
exit()	The exit system call is used to exit a process.

```c
/*  open.c */

  #include <fcntl.h>          /* defines options flags */
  #include <sys/types.h>      /* defines types used by sys/stat.h */
  #include <sys/stat.h>       /* defines S_IREAD & S_IWRITE  */

  static char message[] = "Hello, world";
  int main()
  {
```

```
    int fd;
    char buffer[80];
fd = open("datafile.dat",O_RDWR | O_CREAT | O_EXCL, S_IREAD | S_IWRITE);
    if (fd != -1)
       {
       printf("datafile.dat opened for read/write access\n");
       write(fd, message, sizeof(message));
       lseek(fd, 0L, 0);      /* go back to the beginning of the file */
       if (read(fd, buffer, sizeof(message)) == sizeof(message))
          printf("\"%s\" was written to datafile.dat\n", buffer);
       else
          printf("*** error reading datafile.dat ***\n");
       close (fd);
       }
    else
       printf("*** datafile.dat already exists ***\n");
    exit (0);
}
```

OUTPUT




```
#include <stdio.h>
#include <time.h>       /* may need to be  #include <sys/time.h> instead */

int main()
{
    long now, time();
```

```
        char *ctime();

        time (&now);
        printf("It is now %s\n", ctime (&now));

        exit (0);
    }
```

OUTPUT

```
main.c:17:9: warning: implicit declaration of function 'exit' [-Wimplicit-function-declaration]
main.c:17:9: warning: incompatible implicit declaration of built-in function 'exit'
main.c:17:9: note: include '<stdlib.h>' or provide a declaration of 'exit'
It is now Wed Apr 29 12:14:16 2020



...Program finished with exit code 0
Press ENTER to exit console.
```