MEDI-CAPS UNIVERSITY

Knowledge is Power

Operating System

PRACTICAL FILE

Enrolment No.	:	EN20CS306049
Name of Student	:	Shobhit Gupta
Department	:	Computer Science & Engineering
Faculty of	:	Engineering
Class	:	B.Tech. CSBS
Year/Sem	:	II Year / IV (Even) Sem
Course Name	:	Operating System
Course Code	:	CB3CO06
Faculty Name	:	Vineeta Rathore

1. List of Operating Systems and Their Features:

S.No.	Name of OS	Versions	Release Date	Special Features
1	MS Windows	Windows 1.01 Windows 2.01 Windows 3.0 Windows 95 Windows XP Windows Vista Windows 7 Windows 8 Windows 10 Windows 11	20-11-1985 9-12-1987 22-05-1990 24-08-1995 25-10-2001 30-01-2007 22-10-2009 26-10-2012 29-7-2015 05-10-2021	• Protected and supervisor mode. • Allows disk access and file systems • Device drivers • Networking • Security. • Program Execution. • Memory management • Virtual Memory • Multitasking.
2	Solaris	Solaris 1 Solaris 2 Solaris 7 Solaris 8 Solaris 9 Solaris 10 Solaris 11	1991-1994 June 1992 November 1998 Feb 2000 28-05-2002 January 31, 2005 November 15, 2010	• Virtualization Technology - Solaris Containers. • Virtualization Technology - Solaris ZFS. • Availability Improvement - Predictive Self-Healing. • Performance Bottleneck Resolution - Dynamic Trace (DTrace) • Security -Process Privilege Administration. • Performance improvement.

3	Linux	Debian Linux. Gentoo Linux. Ubuntu Linux. Linux Mint Desktop. RHEL Linux Distribution. CentOS Linux Distribution. Fedora Linux Distribution. Kali Linux Distribution	16-08-1993 26-07-2000 20-10-2004 27-08-2006 07-05-2019 14-05-2004 06-11-2003 13-03-2013	• Stable and Dependable • High performance. • free and an open-source operating system. • Faster and lighter than Ubuntu • Lesser Memory Usage • Easy Configuration. • One command application installation • Software Management: • Installation and Image creation • Fedora OS offers many architectures. • Fedora OS is a very reliable and stable operating system. • Multi-

				language support. • Developed in a secure environment
4	Android OS	1.0 1.1 Cupcake Donut Éclair Froyo Gingerbread Honeycomb Ice Cream JellyBean Kitkat Lollipop Marshmallow Nougat Oreo Pie Android 10 Android 11	September 23, 2008 February 9, 2009 April 27, 2009 September 15, 2009 October 26, 2009 May 20, 2010 December 6, 2010 February 22, 2011 October 18, 2011 July 9, 2012 October 31, 2013 November 12, 2014 October 5, 2015 August 22, 2016	• Near Field Communication (NFC) • Alternate Keyboards. • IR Transmission. • No-Touch Control. • Automation. • Wireless App Downloads. • Storage & Battery Swap. • Custom Home Screen. • Connectivity GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX.
5	MS DOS	• 1. MS-DOS 1.x • 2_MS-DOS 2.x • 3_MS-DOS 3.x ○ 3.1_MS-DOS 3.00 ○ 3.2_MS-DOS 3.10 ○ 3.3_MS-DOS 3.20 ○ 3.4_MS-DOS 3.21 ○ 3.5_MS-DOS 3.30 ○ 3.6_MS-DOS 3.31 • 4_MS-DOS 4.0 (IBM-		**Features of DOS** • It is a single user system. • It controls program. • It is machine independence. • It manages (computer) files. • It manages input and

		developed) • 5 MS-DOS 5.x • 6 MS-DOS 6.x ○ 6.1 MS-DOS 6.0 ○ 6.2 MS-DOS 6.20 ○ 6.3 MS-DOS 6.21 ○ 6.4 MS-DOS 6.22 • 7 MS-DOS 7.x • 8 MS-DOS 8.x • 9 Other Versions ○ 9.1 MS-DOS 4.0 (multitasking) and MS-DOS 4.1 ○ 9.2 MS-DOS Mobile 1.0 (Windows Phone)		output system. • It manages (computer) memory. • It provides command processing facilities. • It operates with Assembler.
6	UBUNTU	Ubuntu 21.10 Ubuntu **20.04.3 LTS** Ubuntu 20.04.2 LTS Ubuntu 20.04.1 LTS Ubuntu 20.04 LTS Ubuntu **18.04.6 LTS** Ubuntu 18.04.5 LTS Ubuntu 18.04.4 LTS Ubuntu 18.04.3 LTS Ubuntu 18.04.2 LTS Ubuntu 18.04.1 LTS Ubuntu 18.04 LTS Ubuntu **16.04.7 LTS** Ubuntu 16.04.6 LTS Ubuntu 16.04.5 LTS Ubuntu 16.04.4 LTS Ubuntu 16.04.3 LTS Ubuntu 16.04.2 LTS Ubuntu 16.04.1 LTS Ubuntu 16.04 LTS Ubuntu **14.04.6 LTS**	October 14, 2021 August 26, 2021 February 4, 2021 August 6, 2020 April 23, 2020 September 17.2021 August 13, 2020 February 12, 2020 August 8, 2019 February 15, 2019 July 26, 2018 April 26, 2018 August 13, 2020	• Ubuntu is free and an open-source operating system. • Ubuntu is more secure. • Ubuntu runs without install. • Ubuntu supports window tiling. • Ubuntu is more resource-friendly. • Ubuntu is completely customizable

		Ubuntu 14.04.5 LTS Ubuntu 14.04.4 LTS Ubuntu 14.04.3 LTS Ubuntu 14.04.2 LTS Ubuntu 14.04.1 LTS Ubuntu 14.04 LTS	February 28, 2019 August 2, 2018 March 1, 2018 August 3, 2017 February 16, 2017 July 21, 2016 April 21, 2016 March 7, 2019 August 4, 2016 February 18, 2016 August 6, 2015 February 20, 2015 July 24, 2014	. • A well-rounded operating system for desktop computing.
7	Symbian OS	EPOC32 Symbian OS 6.0 and 6.1 Symbian OS 6.2 Symbian OS 7.0 Symbian OS 7.0 Symbian OS 8.0 Symbian OS 9.1 Symbian OS 9.3 Symbian OS 9.5		Symbian OS contained **a browser, messaging, multimedia, communication protocol, mobile telephony, data synchronization, security, application environment, multi-tasking, robustness, flexible**.
8	Chrome OS	**Chrome OS 87 Desktop**	(January 18, 2022)	1. Speedy boot-up, as fast as three-seconds. 2. Security by default. 3. Support for both x86 and ARM architectu

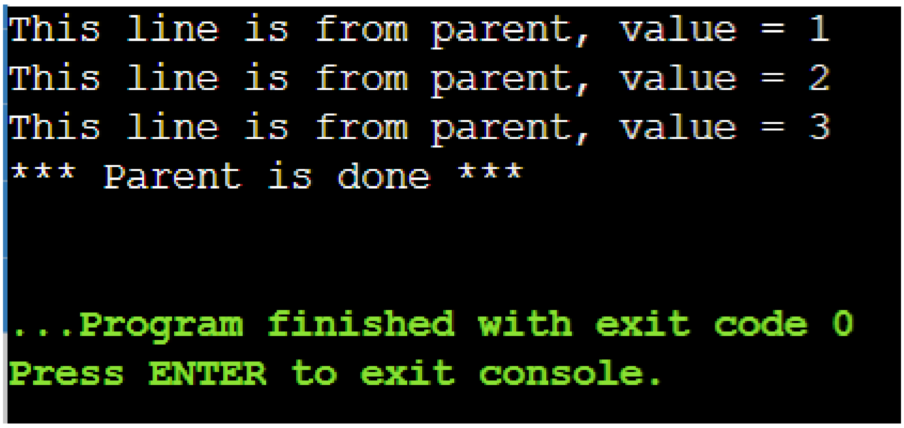
				res. 4. The application menu. 5. A surprising way to support Microsoft **Office.**
9	Fedora	1 (Yarrow) 2 (Tettnang) 3 (Heidelberg) 4 (Stentz) 5 (Bordeaux) 6 (Zod) 7 (Moonshine) 8 (Werewolf) 9 (Sulphur) 10 (Cambridge) 11 (Leonidas) 12 (Constantine) 13 (Goddard) 14 (Laughlin)	2003-11-06 2004-05-18 2004-11-08 2005-06-13 2006-03-20 2006-10-24 2007-05-31 2007-11-08 2008-05-13 2008-11-25 2009-06-09 2009-11-17	• Fedora OS offers many architectures . • Fedora OS is a very reliable and stable operating system. • It provides unique security features. • Fedora OS provides a very powerful firewall. • Fedora OS is very easy to use. • It supports a large community. • Fedora OS is actively developed.

		15 (Lovelock) 16 (Verne) 17 (Beefy Miracle) 18 (Spherical Cow) 19 (Schrödinger's Cat) 20 (Heisenbug)	2010-05-25 2010-11-02 2011-05-24 2011-11-08 2012-05-29 2013-01-15 2013-07-02 2013-12-17	
10	Cent OS	CentOS Linux 3 CentOS Linux 4 CentOS Linux 5 CentOS Linux 6 CentOS Linux 7.0-1406 CentOS Linux 8.0-1905	March 19th, 2004 March 9th, 2005 April 12th, 2007 July 10th, 2011 July 7th, 2014 September 24th, 2019	• Desktop Environment. Unlike previous CentOS versions where the default installation did not include a GUI, the CentOS 8 default desktop environment is GNOME 3.28. ... • Networking. • Cockpit Web Console. • Content

				Distribution. • Software Management

2. Write a program on fork() system call.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#define MAX_COUNT 3

void ChildProcess(void);
void ParentProcess(void);

int main(void) {
pid_t pid=fork();
if (pid == 0)
ChildProcess();

else
ParentProcess();
return 0;
}

void ChildProcess(void)
{

int i;
for (i = 1; i <= MAX_COUNT; i++)
printf(" This line is from child, value = %d\n", i);
printf(" *** Child process is done ***\n");
}

void ParentProcess(void)
{
int i;
for (i = 1; i <= MAX_COUNT; i++)
printf("This line is from parent, value = %d\n", i);
printf("*** Parent is done ***\n");
}
```



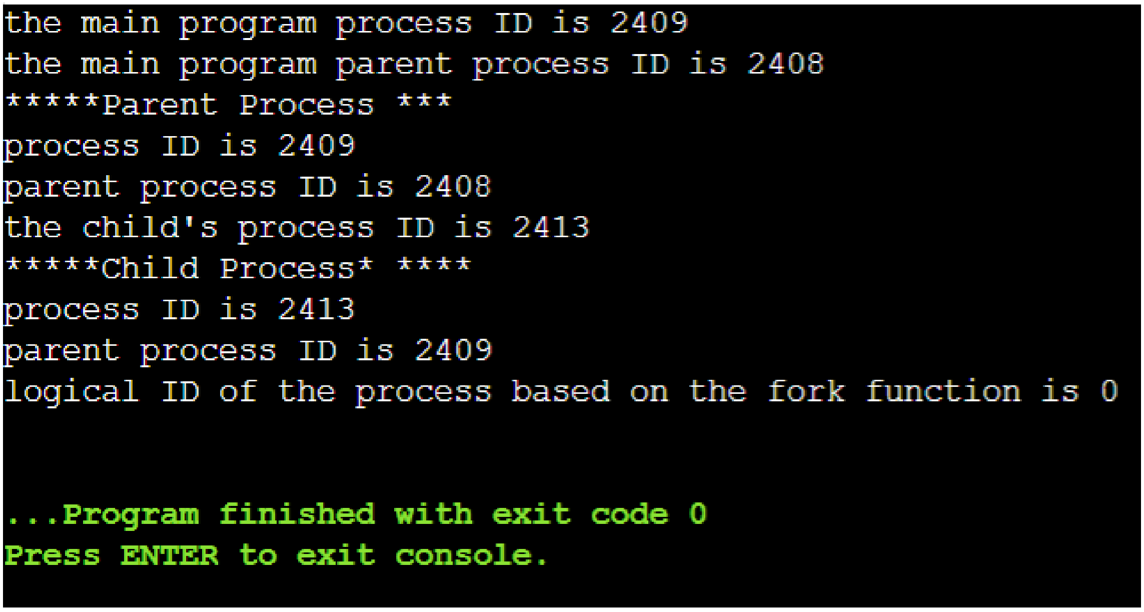
```
This line is from parent, value = 1
This line is from parent, value = 2
This line is from parent, value = 3
*** Parent is done ***


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

3. Write a program on fork() system call.

#include <stdio.h>

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

int main (void){
pid_t fork_pid;
printf ("the main program process ID is %d\n", (int)
getpid ());
printf ("the main program parent process ID is %d\n",
(int) getppid ());

fork_pid = fork ();
if (fork_pid != 0){
printf ("*****Parent Process ***\n");
printf ("process ID is %d\n", (int) getpid ());
printf ("parent process ID is %d\n", (int) getppid ());
printf ("the child's process ID is %d\n", (int)
fork_pid);

sleep(10); }

else{
wait (NULL);

printf ("*****Child Process* ****\n");
printf ("process ID is %d\n", (int) getpid ());
printf ("parent process ID is %d\n", (int) getppid ());
printf ("logical ID of the process based on the fork
function is %d\n", (int) fork_pid); }

return 0; }
```

OUTPUT :

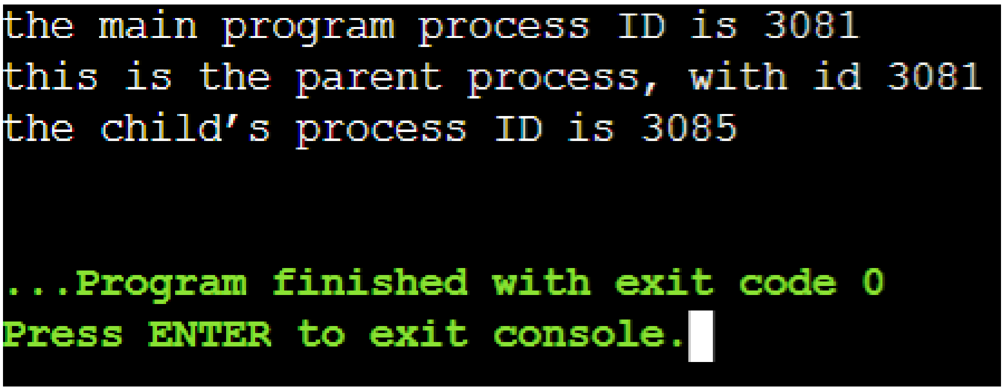


```
the main program process ID is 2409
the main program parent process ID is 2408
*****Parent Process ***
process ID is 2409
parent process ID is 2408
the child's process ID is 2413
*****Child Process* ****
process ID is 2413
parent process ID is 2409
logical ID of the process based on the fork function is 0


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

4. Write a program on fork() system call.

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int main ()
{pid_t child_pid;

printf ("the main program process ID is %d\n", (int)
getpid ());

child_pid = fork ();

if (child_pid != 0)
{
printf ("this is the parent process, with id %d\n",
(int) getpid ());
printf ("the child’s process ID is %d\n", (int)
child_pid);

}

else

printf ("this is the child process, with id %d\n",
(int) getpid ());

return 0;

}
```

OUTPUT :

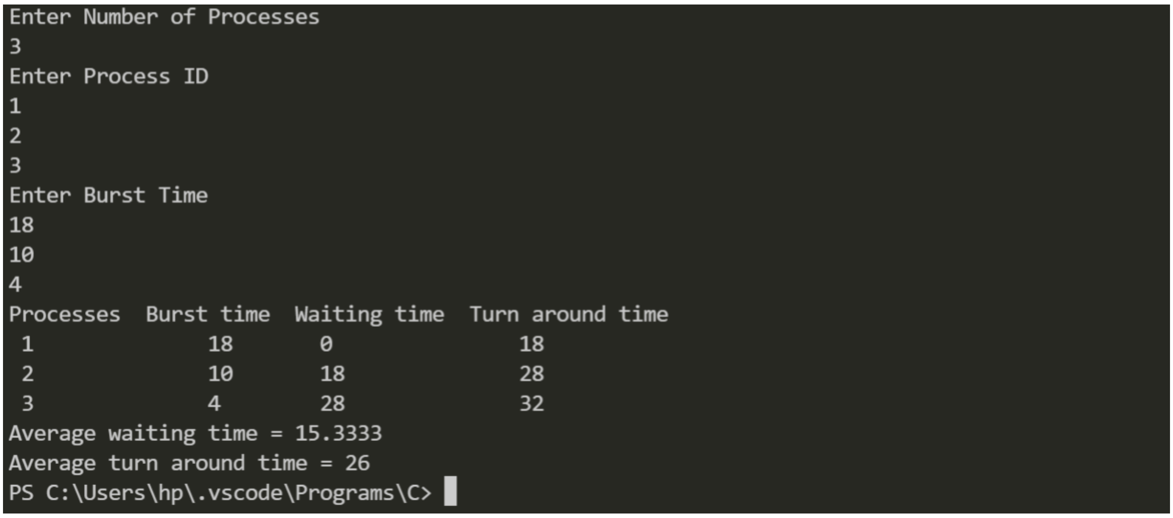


```
the main program process ID is 3081
this is the parent process, with id 3081
the child's process ID is 3085


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

5. Write a program to implement First Come First Serve Scheduling Algorithm.

```
#include <iostream>
using namespace std;
void WaitingTime(int processes[], int n,
                 int bt[], int wt[])
{
    wt[0] = 0; // waiting time for first process is 0
    for (int i = 1; i < n; i++)
        wt[i] = bt[i - 1] + wt[i - 1];
}

void TurnAround(int processes[], int n,
                int bt[], int wt[], int tart[])
{
    for (int i = 0; i < n; i++)
        tart[i] = bt[i] + wt[i];
}
void AvgTime(int processes[], int n, int bt[])
{
    int wt[n], tart[n], total_wt = 0, total_tart = 0;
    WaitingTime(processes, n, bt, wt);

    TurnAround(processes, n, bt, wt, tart);

    cout << "Processes "
         << " Burst time "
         << " Waiting time "
         << " Turn around time\n";
    for (int i = 0; i < n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tart = total_tart + tart[i];
        cout << " " << i + 1 << "\t\t" << bt[i] <<"\t "
             << wt[i] << "\t\t " << tart[i] << endl;
    }
    cout << "Average waiting time = "
         << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "
         << (float)total_tart / (float)n;
}
int main()
{
    int n;
    cout << "Enter Number of Processes";
    cin >> n;
    int processes[n], burst_time[n];
    cout << "Enter Process ID";
    for (int i = 0; i < n; i++)
    {
        cin >> processes[i];
    }
    cout << "Enter Burst Time";
    for (int m = 0; m < n; m++)
    {
        cin >> burst_time[m];
    }
    AvgTime(processes, n, burst_time);
    return 0;
}
```

OUTPUT:

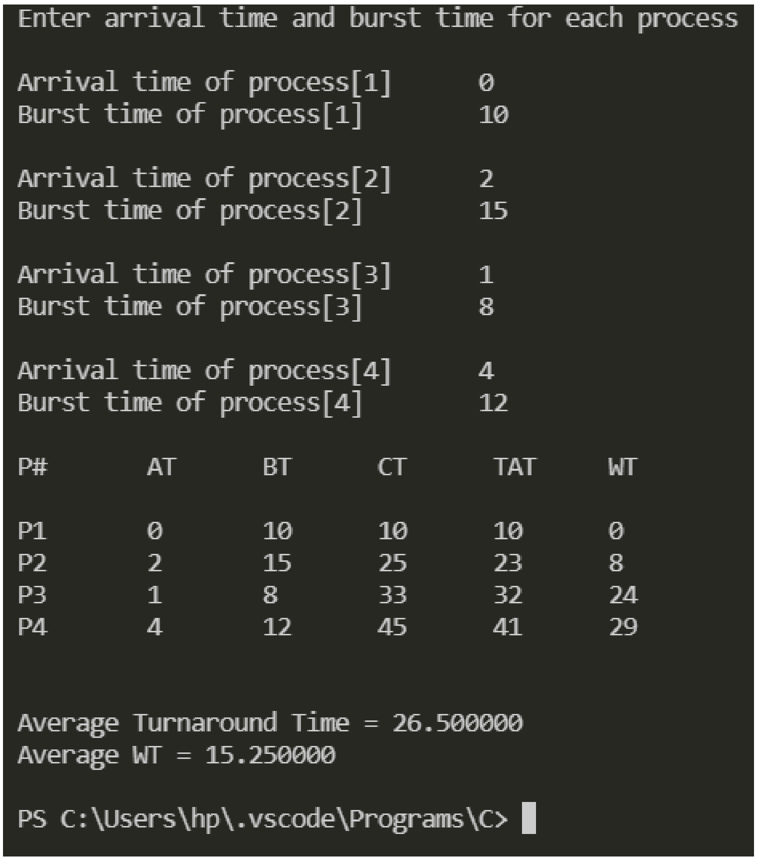


```
Enter Number of Processes
3
Enter Process ID
1
2
3
Enter Burst Time
18
10
4
Processes  Burst time  Waiting time  Turn around time
 1           18        0               18
 2           10        18              28
 3           4         28              32
Average waiting time = 15.3333
Average turn around time = 26
PS C:\Users\hp\.vscode\Programs\C>
```

6. Write a program to implement First Come First Serve Scheduling Algorithm with arrival time.

```
#include <stdio.h>
int main()
{int bt[10] = {0}, at[10] = {0}, tat[10] = {0}, wt[10] =
{0}, ct[10] = {0};

int n, sum = 0;
float totalTAT = 0, totalWT = 0;



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

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

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

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

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

printf("\n");

}

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

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

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

}
printf("P#\t AT\t BT\t CT\t TAT\t WT\t\n\n");
for (int i = 0; i < n; i++)

{
printf("P%d\t %d\t %d\t %d\t %d\t %d\n", i + 1, at[i],
bt[i], ct[i], tat[i], wt[i]);
}
printf("\n\nAverage Turnaround Time = %f\n",totalTAT / n);
printf("Average WT = %f\n\n", totalWT / n);

return 0;

}
```

OUTPUT:



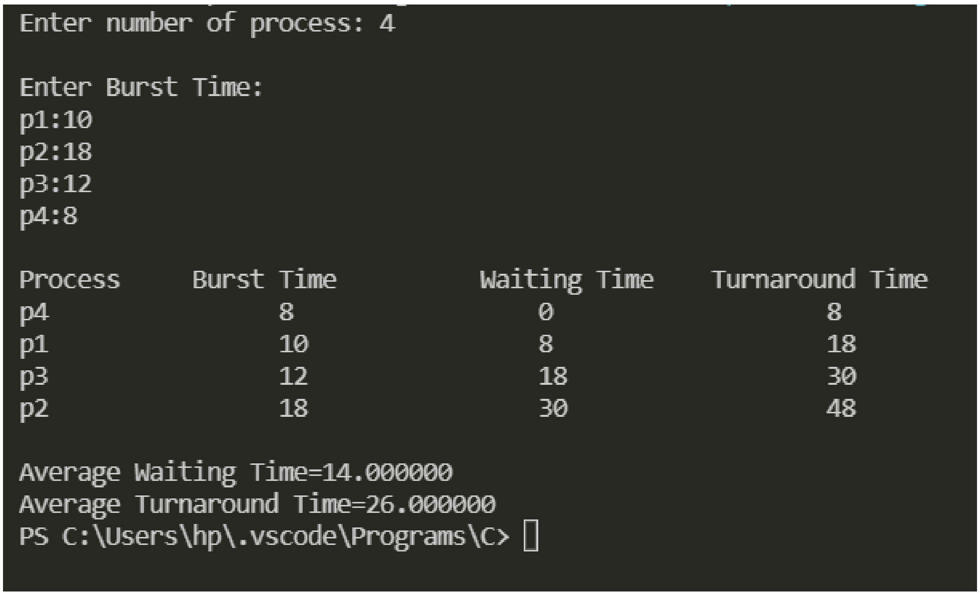
```
Enter arrival time and burst time for each process

Arrival time of process[1]     0
Burst time of process[1]       10

Arrival time of process[2]     2
Burst time of process[2]       15

Arrival time of process[3]     1
Burst time of process[3]       8

Arrival time of process[4]     4
Burst time of process[4]       12

P#       AT      BT      CT      TAT     WT

P1       0       10      10      10      0
P2       2       15      25      23      8
P3       1       8       33      32      24
P4       4       12      45      41      29


Average Turnaround Time = 26.500000
Average WT = 15.250000

PS C:\Users\hp\.vscode\Programs\C>
```

7. Write a program to implement Shortest Job First Scheduling Algorithm.

```
#include <stdio.h>
int main()
{
int bt[20], p[20], wt[20], tat[20], i, j, n, total = 0,
pos, temp;
    float avg_wt, avg_tat;
    printf("Enter number of process:");
    scanf("%d", &n);

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

    //sort burst time
    for (i = 0; i < n; i++)
    {
        pos = i;
        for (j = i + 1; j < n; j++)
        {
            if (bt[j] < bt[pos])
                pos = j;
        }

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

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

```
            p[pos] = temp;
        }

        wt[0] = 0; // waiting time for first process will
    be zero

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

            total += wt[i];
        }

        avg_wt = (float)total / n; // average waiting time
        total = 0;

        printf("\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];
            printf("\np%d\t\t  %d\t\t    %d\t\t\t%d", p[i],
    bt[i], wt[i], tat[i]);
        }

        avg_tat = (float)total / n; // average turnaround
    time
        printf("\n\nAverage Waiting Time=%f", avg_wt);
        printf("\nAverage Turnaround Time=%f\n", avg_tat);
        return 0;
    }
```

OUTPUT :



```
Enter number of process: 4

Enter Burst Time:
p1:10
p2:18
p3:12
p4:8

Process     Burst Time          Waiting Time    Turnaround Time
p4                8                  0                    8
p1                10                 8                    18
p3                12                 18                   30
p2                18                 30                   48

Average Waiting Time=14.000000
Average Turnaround Time=26.000000
PS C:\Users\hp\.vscode\Programs\C> 
```

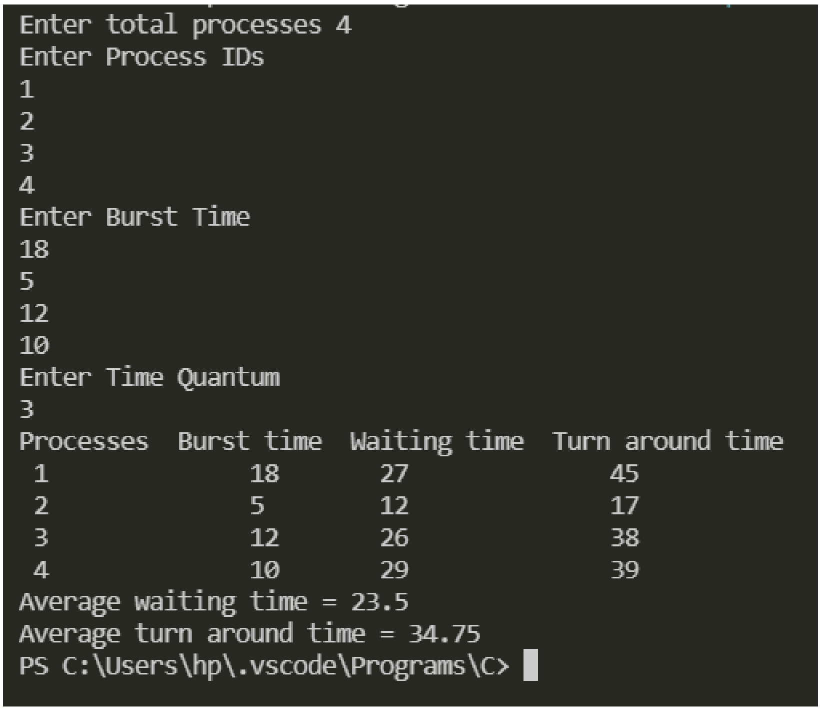
8. Write a program to implement Round Robin Scheduling Algorithm.

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

void findWaitingTime(int processes[], int n,
           int bt[], int wt[], int quantum)
{

  int rem_bt[n];
  for (int i = 0 ; i < n ; i++)
       rem_bt[i] = bt[i];
  int t = 0; // Current time

  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 processes";
    cin>>n;
  int processes[n],burst_time[n];
    cout<<"Enter Process IDs";
  for(int i=0;i<n;i++)
    {
         cin>>processes[i];
    }
  cout<<"Enter Burst Time";
    for(int i=0;i<n;i++){
    cin>>burst_time[i];
    }

  int quantum;
    cout<<"Enter Time Quantum";
    cin>>quantum;
  findavgTime(processes, n, burst_time, quantum);
  return 0;
}
```

OUTPUT :

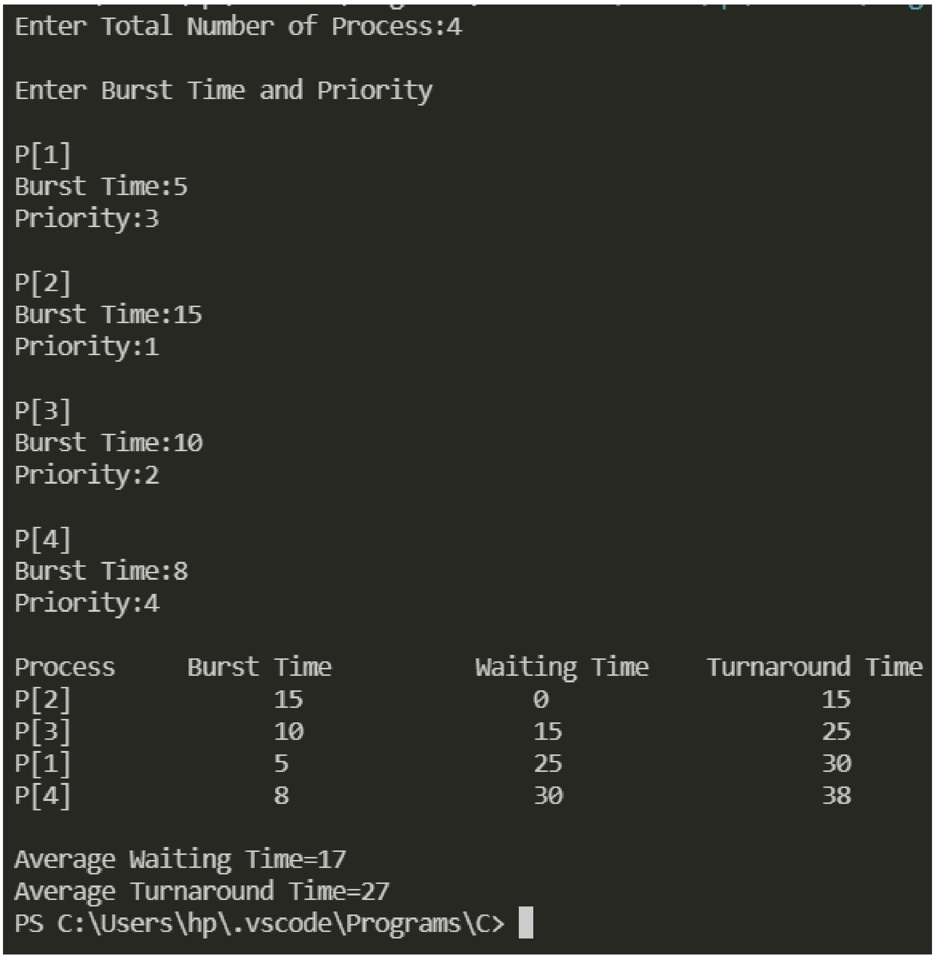


```
Enter total processes 4
Enter Process IDs
1
2
3
4
Enter Burst Time
18
5
12
10
Enter Time Quantum
3
Processes  Burst time  Waiting time  Turn around time
 1          18      27             45
 2          5       12             17
 3          12      26             38
 4          10      29             39
Average waiting time = 23.5
Average turn around time = 34.75
PS C:\Users\hp\.vscode\Programs\C>
```

9. Write a program to implement Priority Scheduling Algorithm.

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

int main()
{
int bt[20],p[20],wt[20],tart[20],pr[20];
int i,j,n,total=0,pos,temp,avg_wt,avg_tart;
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;
}

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++)
    {
        tart[i]=bt[i]+wt[i];
        total+=tart[i];
        cout<<"\nP["<<p[i]<<"]\t\t  "<<bt[i]<<"\t\t
"<<wt[i]<<"\t\t\t"<<tart[i]; }

    avg_tart=total/n;
    cout<<"\n\nAverage Waiting Time="<<avg_wt;
    cout<<"\nAverage Turnaround Time="<<avg_tart;

    return 0; }
```

OUTPUT



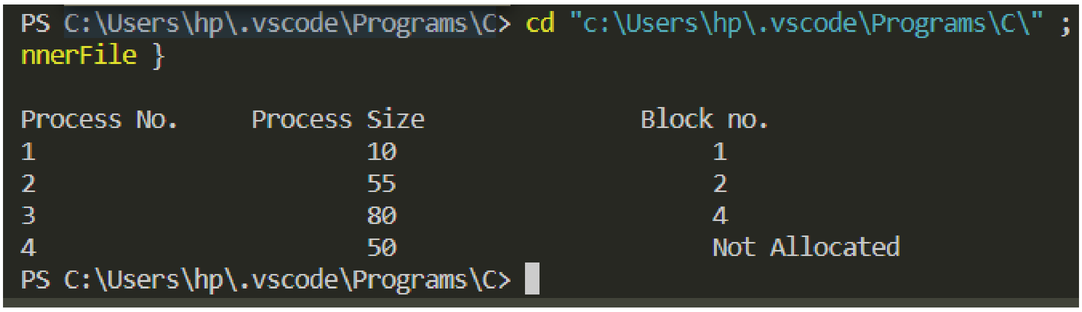
```
Enter Total Number of Process:4

Enter Burst Time and Priority

P[1]
Burst Time:5
Priority:3

P[2]
Burst Time:15
Priority:1

P[3]
Burst Time:10
Priority:2

P[4]
Burst Time:8
Priority:4

Process     Burst Time          Waiting Time    Turnaround Time
P[2]              15                 0                   15
P[3]              10                 15                  25
P[1]              5                  25                  30
P[4]              8                  30                  38

Average Waiting Time=17
Average Turnaround Time=27
PS C:\Users\hp\.vscode\Programs\C>
```

10. Write a program to implement First Fit Memory Allocation Technique.

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

void FirstFit(int blockSize[], int blocks, int
processSize[], int processes)
{
    // This will store the block id of the
allocated block to a process
    int alloc[processes];
    int occupied[blocks];

    // initially assigning -1 to all
allocation indexes
    // means nothing is allocated currently
    memset(alloc, -1, sizeof(alloc));

  for(int i = 0; i < blocks; i++){
        occupied[i] = 0;
    }

    // take each process one by one and find
    // first block that can accomodate it
    for (int i = 0; i < processes; i++)
    {
        for (int j = 0; j < blocks; j++)
        {
```

```
        if (!occupied[j] && blockSize[j] >=
processSize[i])
            {
                // alloc block j to p[i]
process
                alloc[i] = j;
                occupied[j] = 1;

                break;
            }
        }
    }

    printf("\nProcess No.\tProcess Size\t
Block no.\n");
    for (int i = 0; i < processes; i++)
    {
        cout << i + 1 << "\t\t\t" <<
processSize[i] << "\t\t\t";
        if (alloc[i] != -1)
            cout << alloc[i] + 1 << endl;
        else
            cout << "Not allocd" << endl;
    }
}

int main()
{
    int blockSize[] = {50,70,45,100,30};
    int processSize[] = {10,55,80,50};
    int m = 
sizeof(blockSize)/sizeof(blockSize[0]);
    int n = 
sizeof(processSize)/sizeof(processSize[0]);

    FirstFit(blockSize, m, processSize, n);

    return 0;
}
```

OUTPUT :



```
PS C:\Users\hp\.vscode\Programs\C> cd "c:\Users\hp\.vscode\Programs\C\" ;
nnerFile }

Process No.     Process Size             Block no.
1                       10                    1
2                       55                    2
3                       80                    4
4                       50                    Not Allocated
PS C:\Users\hp\.vscode\Programs\C> 
```

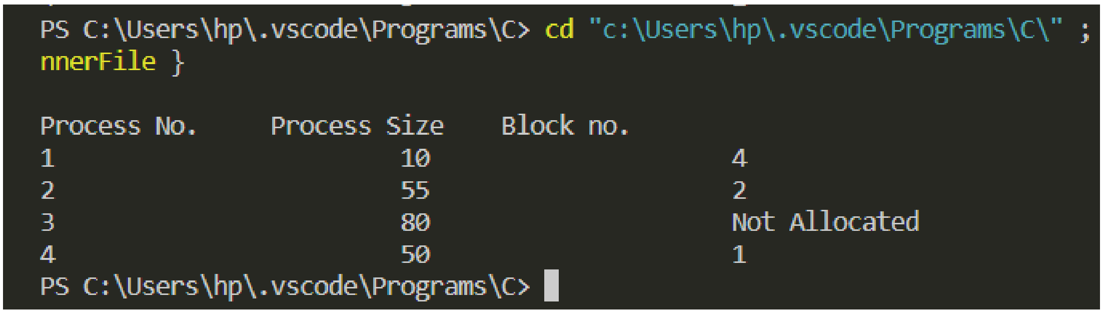
11. Write a program to implement Best Fit Memory Allocation Technique.

```
#include<stdio.h>

void BestFit(int blockSize[], int blocks, int
processSize[], int proccesses)
{
    // This will store the block id of the
allocated block to a process
    int alloc[proccesses];
    int occupied[blocks];

    // initially assigning -1 to all alloc
indexes
    // means nothing is allocated currently
    for(int i = 0; i < proccesses; i++){
        alloc[i] = -1;
    }

    for(int i = 0; i < blocks; i++){
        occupied[i] = 0;
    }

    // pick each process and find suitable
blocks
    // according to its size ad assign to it
    for (int i=0; i<proccesses; i++)
    {
        int index = -1;
        for (int j=0; j<blocks; j++)
        {
            if (blockSize[j] >= processSize[i]
&& !occupied[j])
            {
                // place it at the first block
fit to accomodate process
                if (index == -1)
                    index = j;

                // if any future block is
larger than the current block where
                // process is placed, change
the block and thus index
                else if (blockSize[index] <
blockSize[j])
                    index = j;
            }
        }

        // If we were successfully able to
find block for the process
        if (index != -1)
        {
            // allocate this block j to
process p[i]
            alloc[i] = index;

            // make the status of the block as
occupied
```

```
            occupied[index] = 1;

            // Reduce available memory for the
block
            blockSize[index] -=
processSize[i];
        }
    }

    printf("\nProcess No.\tProcess Size\tBlock
no.\n");
    for (int i = 0; i < proccesses; i++)
    {
        printf("%d \t\t\t %d \t\t\t", i+1,
processSize[i]);
        if (alloc[i] != -1)
            printf("%d\n",alloc[i] + 1);
        else
            printf("Not Allocated\n");
    }
}


int main()
{
    int blockSize[] = {50,70,45,100,30};
    int processSize[] = {10,55,80,50};
    int blocks =
sizeof(blockSize)/sizeof(blockSize[0]);
    int proccesses =
sizeof(processSize)/sizeof(processSize[0]);
```

```
   BestFit(blockSize, blocks, processSize, 
proccesses);

    return 0 ;
}
```

OUTPUT :

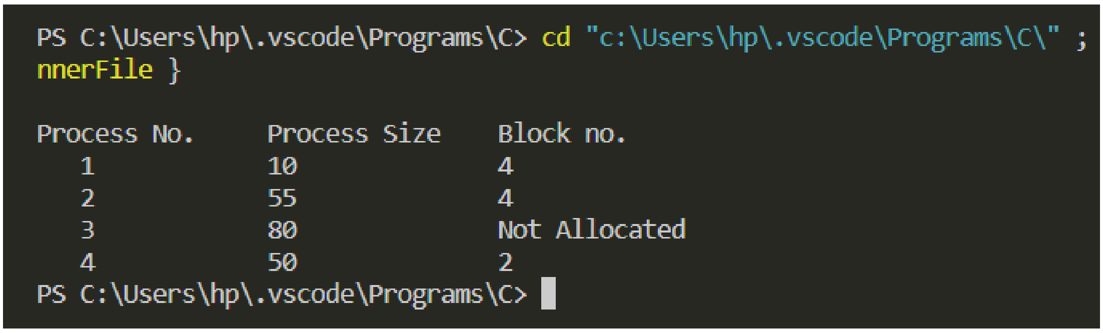


```
PS C:\Users\hp\.vscode\Programs\C> cd "c:\Users\hp\.vscode\Programs\C\" ;
nnerFile }

Process No.     Process Size    Block no.
1                        10                      4
2                        55                      2
3                        80                      Not Allocated
4                        50                      1
PS C:\Users\hp\.vscode\Programs\C> 
```

12. Write a program to implement Worst Fit Memory Allocation Technique

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

// Function to allocate memory to blocks as
per worst fit
// algorithm
void WorstFit(int blockSize[], int m, int
processSize[],int n)
{
    // Stores block id of the block allocated
to a
    // process
    int alloc[n];
    // Initially no block is assigned to any
process
    memset(alloc, -1, sizeof(alloc));
    // 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 worst = -1;
        for (int j=0; j<m; j++)
        {
            if (blockSize[j] >=
processSize[i])
            {
                if (worst == -1)
                    worst = j;
                else if (blockSize[worst] <
blockSize[j])
                    worst = j;
            }
        }

        // If we could find a block for
current process
        if (worst != -1)
        {
            // allocate block j to p[i]
process
            alloc[i] = worst;

            // Reduce available memory in this
block.
            blockSize[worst] -=
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 (alloc[i] != -1)
            cout << alloc[i] + 1;
        else
            cout << "Not Allocated";
        cout << endl;
    }
}



int main()
{
    int blockSize[] = {50,70,45,100,30};
    int processSize[] = {10,55,80,50};

    int m =
sizeof(blockSize)/sizeof(blockSize[0]);
    int n =
sizeof(processSize)/sizeof(processSize[0]);

    WorstFit(blockSize, m, processSize, n);

    return 0 ;
}
```

OUTPUT :



```
PS C:\Users\hp\.vscode\Programs\C> cd "c:\Users\hp\.vscode\Programs\C\" ;
nnerFile }

Process No.     Process Size    Block no.
   1            10              4
   2            55              4
   3            80              Not Allocated
   4            50              2
PS C:\Users\hp\.vscode\Programs\C>
```