



# Graphic Era

Deemed to be University

Accredited by NAAC with Grade A

NBA Accredited Program in CSE, ECE & ME  
Approved by AICTE, Ministry of HRD, Govt. of India

---

## OPERATING SYSTEM

### TERM WORK

Name: Mohd Nasir

Sec: "F"

Roll No: 14

Student Id:20021595

Uni Roll No:2016855

Subject Name: Operating System Lab

Subject Code: PCS-502

Student Sign:

Professor Sign:



# GraphicEra

Deemed to be University

Accredited by NAAC with Grade A

NBA Accredited Program in CSE, ECE & ME  
Approved by AICTE, Ministry of HRD, Govt. of India

---

## DATA BASE MANAGEMENT SYSTEM

### TERM WORK

Name: Mohd Nasir

Sec: "F"

Roll No: 14

Student Id:20021595

Uni Roll No:2016855

Subject Name: DBMS Lab

Subject Code: PCS-503

Student Sign:

Professor Sign:

**Ques. Write a program to Implement Fork() System Call.**

```
#include <stdio.h>

#include <unistd.h>

int main()

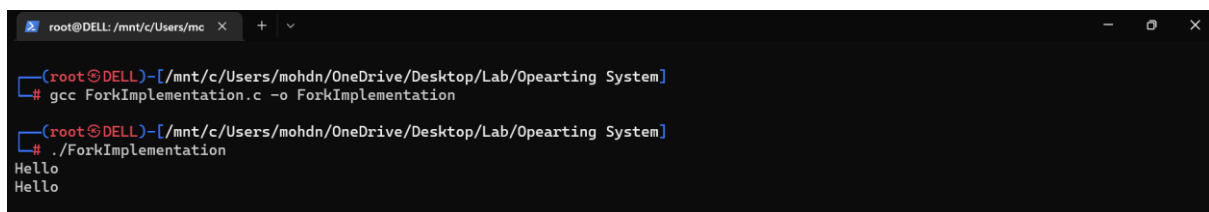
{

    fork();

    printf("Hello\n");

    return 0;

}
```

A terminal window with a dark background. The title bar shows 'root@DELL: /mnt/c/Users/mc'. The prompt is '(root@DELL)-[/mnt/c/Users/mohdn/OneDrive/Desktop/Lab/Operating System]'. The first command is '# gcc ForkImplementation.c -o ForkImplementation'. The second command is '# ./ForkImplementation'. The output shows 'Hello' on the first line and 'Hello' on the second line, indicating a successful fork.

```
root@DELL: /mnt/c/Users/mc x + v

(root@DELL)-[/mnt/c/Users/mohdn/OneDrive/Desktop/Lab/Operating System]
# gcc ForkImplementation.c -o ForkImplementation

(root@DELL)-[/mnt/c/Users/mohdn/OneDrive/Desktop/Lab/Operating System]
# ./ForkImplementation
Hello
Hello
```

**Ques. Write a Program to Implement More than Two Fork() System Call.**

```
#include <stdio.h>

#include <unistd.h>

int main()
{
    fork();

    printf("LINUX\n");

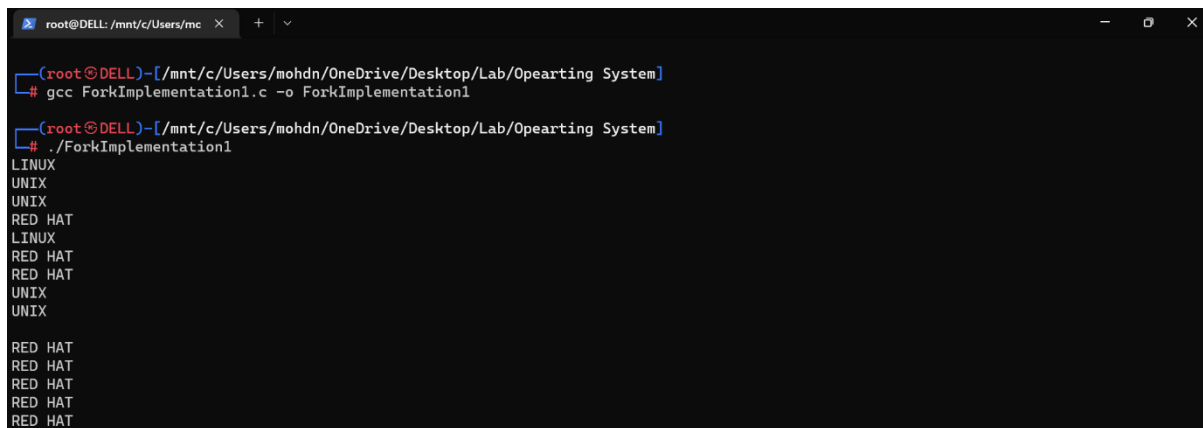
    fork();

    printf("UNIX\n");

    fork();

    printf("RED HAT\n");

    return 0;
}
```

A terminal window with a dark background and light blue text. The window title is 'root@DELL: /mnt/c/Users/mc'. The prompt is '(root@DELL)-[/mnt/c/Users/mohdn/OneDrive/Desktop/Lab/Operating System]'. The user enters '# gcc ForkImplementation1.c -o ForkImplementation1'. The prompt changes to '# ./ForkImplementation1'. The program outputs 'LINUX', 'UNIX', 'RED HAT', 'LINUX', 'RED HAT', 'RED HAT', 'UNIX', 'UNIX', 'RED HAT', 'RED HAT', 'RED HAT', 'RED HAT', and 'RED HAT' on separate lines.

```
root@DELL: /mnt/c/Users/mc
(root@DELL)-[/mnt/c/Users/mohdn/OneDrive/Desktop/Lab/Operating System]
# gcc ForkImplementation1.c -o ForkImplementation1
(root@DELL)-[/mnt/c/Users/mohdn/OneDrive/Desktop/Lab/Operating System]
# ./ForkImplementation1
LINUX
UNIX
UNIX
RED HAT
LINUX
RED HAT
RED HAT
UNIX
UNIX
RED HAT
RED HAT
RED HAT
RED HAT
RED HAT
```

**Ques. Write a program to print the sum of an array element using the fork () system call if the parent process executes the print sum of even elements if child process executes the print sum of odd elements.**

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main()
{
    int *arr, id, size, evensum = 0, oddsum = 0;

    printf("Please Enter array size :");

    scanf("%d", &size);

    arr = (int *)malloc(size * sizeof(int));

    printf("Please %d Elements :", size);

    for (int i = 0; i < size; i++)
    {
        scanf("%d", &arr[i]);

        if ((arr[i] % 2 == 0)

            evensum += arr[i];

        else

            oddsum += arr[i];

    }

    id = fork();

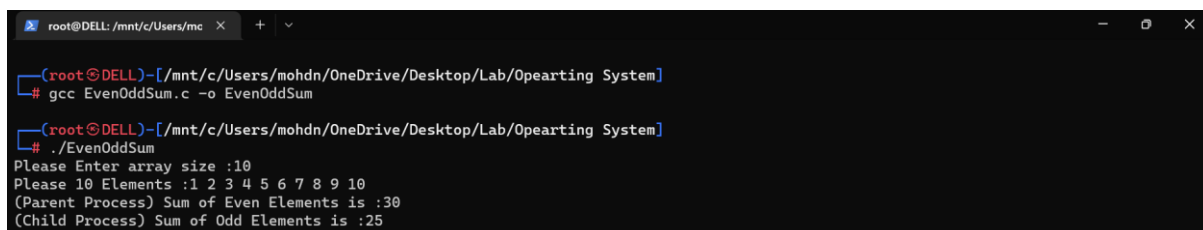
    if (id == 0)

        printf("(Child Process) Sum of Odd Elements is :%d\n", oddsum);

    else if (id > 0)

        printf("(Parent Process) Sum of Even Elements is :%d\n", evensum);

}
```



```
root@DELL: /mnt/c/Users/mc x + v
(root@DELL)~/mnt/c/Users/mohdn/OneDrive/Desktop/Lab/0pearting System
# gcc EvenOddSum.c -o EvenOddSum

(root@DELL)~/mnt/c/Users/mohdn/OneDrive/Desktop/Lab/0pearting System
# ./EvenOddSum
Please Enter array size :10
Please 10 Elements :1 2 3 4 5 6 7 8 9 10
(Parent Process) Sum of Even Elements is :30
(Child Process) Sum of Odd Elements is :25
```

**Ques. Write a program to Implement Wait () System Call.**

```
#include <stdio.h>

#include <stdio.h>

#include <unistd.h>

#include <sys/wait.h>

int main()

{

    int pid = fork();

    if (pid == 0)

    {

        sleep(5);

        printf("Child process id: %d has parent id: %d\n", getpid(), getppid());

    }

    else if (pid > 0)

    {

        wait(NULL);

        printf("Parent process id: %d has grand parent id: %d\n", getpid(), getppid());

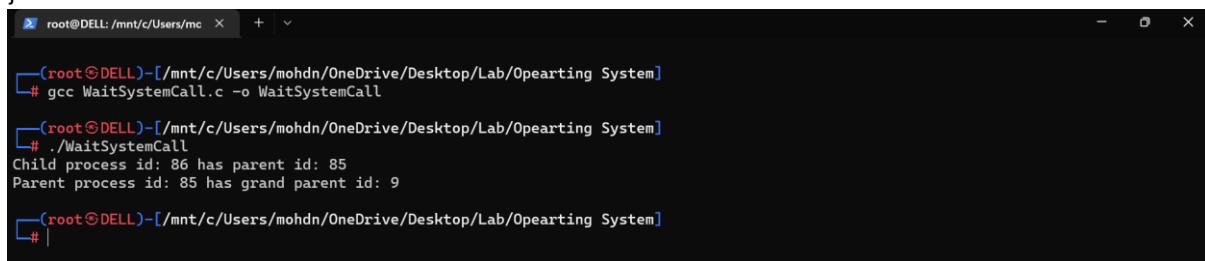
    }

    else

        printf("Process not created");

    return 0;

}
```



The terminal window shows the following commands and output:

```
root@DELL: /mnt/c/Users/mc x + - x
root@DELL:~/mnt/c/Users/mohdn/OneDrive/Desktop/Lab/Opearting System
# gcc WaitSystemCall.c -o WaitSystemCall
root@DELL:~/mnt/c/Users/mohdn/OneDrive/Desktop/Lab/Opearting System
# ./WaitSystemCall
Child process id: 86 has parent id: 85
Parent process id: 85 has grand parent id: 9
root@DELL:~/mnt/c/Users/mohdn/OneDrive/Desktop/Lab/Opearting System
#
```

### Ques. Write a Program for Orphan Process.

```
#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

int main()
{
    pid_t p;

    p = fork();

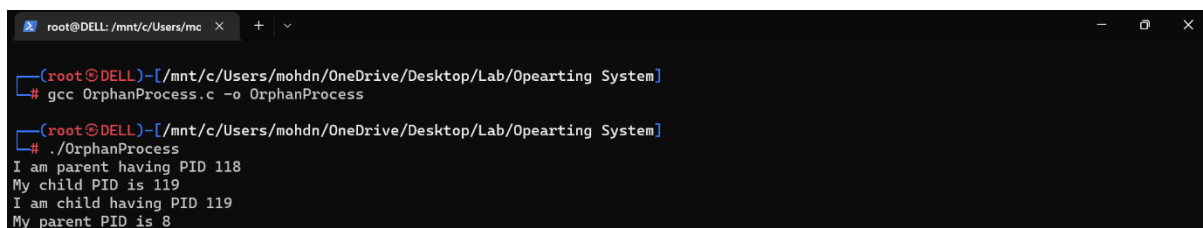
    if (p == 0) /*Child*/
    {
        sleep(50);

        printf("I am child having PID %d\n", getpid());

        printf("My parent PID is %d\n", getppid());
    }
    else /*Parent*/
    {
        printf("I am parent having PID %d\n", getpid());

        printf("My child PID is %d\n", p);
    }

    return 0;
}
```



The screenshot shows a terminal window with the following commands and output:

```
root@DELL: /mnt/c/Users/mc x + v
(root@DELL)-[/mnt/c/Users/mohdn/OneDrive/Desktop/Lab/Opearting System]
# gcc OrphanProcess.c -o OrphanProcess
(root@DELL)-[/mnt/c/Users/mohdn/OneDrive/Desktop/Lab/Opearting System]
# ./OrphanProcess
I am parent having PID 118
My child PID is 119
I am child having PID 119
My parent PID is 8
```

**Ques. Write a program for Zombie Process.**

```
#include <stdio.h>

#include <unistd.h>

#include <sys/wait.h>

int main()
{
    int pid = fork();

    if (pid == 0)
    {
        printf("Child process id: %d has Parent id: %d\n", getpid(), getppid());
    }

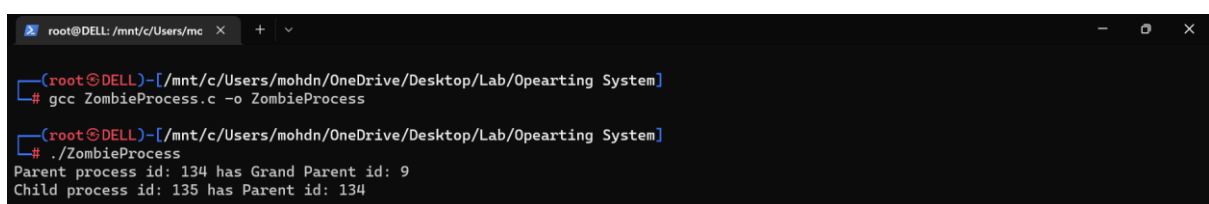
    else if (pid > 0)
    {
        wait(NULL);

        sleep(60);

        printf("Parent process id: %d has Grand Parent id: %d\n", getpid(), getppid());
    }

    else
        printf("Process not created");

    return 0;
}
```



The image shows a terminal window with the following commands and output:

```
root@DELL: /mnt/c/Users/mc x + v
(root@DELL)~/mnt/c/Users/mohdn/OneDrive/Desktop/Lab/Operating System
# gcc ZombieProcess.c -o ZombieProcess

(root@DELL)~/mnt/c/Users/mohdn/OneDrive/Desktop/Lab/Operating System
# ./ZombieProcess
Parent process id: 134 has Grand Parent id: 9
Child process id: 135 has Parent id: 134
```



**Ques. Write a program to implement FCFS (First Come First Serve) Scheduling Algorithm.**

```
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

class DataDetails
{
public:
    int ari, pno, bur;

    int ct, tat, wt;
};

bool comparator(DataDetails d1, DataDetails d2)
{
    if (d1.ari != d2.ari)
        return (d1.ari < d2.ari);
    else
        return (d1.pno < d2.pno);
}

bool comparatorPno(DataDetails d1, DataDetails d2)
{
    return (d1.pno < d2.pno);
}

int main()
{
    cout << "      FCFS SCHEDULING" << endl;

    int size, t = 0;

    float avgtat = 0, avgwt = 0;

    cout << "Enter no of process :";

    cin >> size;
```

```

cout << "AT BT" << endl;
vector<DataDetails> vrr(size);
for (int i = 0; i < size; i++)
{
    cin >> vrr[i].ari >> vrr[i].bur;
    vrr[i].pno = i + 1;
    vrr[i].ct = vrr[i].tat = vrr[i].wt = 0;
}
sort(vrr.begin(), vrr.end(), comparator);
for (int i = 0; i < size; i++)
{
    if (t >= vrr[i].ari)
    {
        t += vrr[i].bur;
        vrr[i].ct = t;
        vrr[i].tat = vrr[i].ct - vrr[i].ari;
        avgtat += vrr[i].tat;
        vrr[i].wt = vrr[i].tat - vrr[i].bur;
        avgwt += vrr[i].wt;
    }
    else
    {
        i--;
        t++;
    }
}
sort(vrr.begin(), vrr.end(), comparatorPno);
cout << "\n          SOLUTION" << endl;
cout << "PN " << "AT " << "BT " << "CT " << "TAT " << "WT " << endl;
for (int i = 0; i < size; i++)

```

```
        cout << "P" << vrr[i].pno << " " << vrr[i].ari << " " << vrr[i].bur << " " << vrr[i].ct << " " <<
vrr[i].tat << " " << vrr[i].wt << " " << endl;

    avgtat = avgtat / size;

    cout << "Average TurnAroundTime is :" << avgtat << endl;

    avgwt = avgwt / size;

    cout << "Average WaitingTime is :" << avgwt << endl;

    return 0;

}
```

Output

Clear

### FCFS SCHEDULING

Enter no of process :5

AT BT

6 7

3 3

2 1

0 3

1 9

### SOLUTION

PN	AT	BT	CT	TAT	WT
P1	6	7	23	17	10
P2	3	3	16	13	10
P3	2	1	13	11	10
P4	0	3	3	3	0
P5	1	9	12	11	2

Average TurnAroundTime is :11

Average WaitingTime is :6.4

Output

Clear

### FCFS SCHEDULING

Enter no of process :5

AT BT

3 4

5 3

0 2

5 1

4 3

### SOLUTION

PN	AT	BT	CT	TAT	WT
P1	3	4	7	4	0
P2	5	3	13	8	5
P3	0	2	2	2	0
P4	5	1	14	9	8
P5	4	3	10	6	3

Average TurnAroundTime is :5.8

Average WaitingTime is :3.2

**Ques. Write a program to implement SJF (Shortest Job First) Scheduling Algorithm.**

```
#include <iostream>
#include <vector>
#include <stdbool.h>
#include <algorithm>
using namespace std;
class DataDetails
{
public:
    int ari, pno, bur;
    int ct, tat, wt;
    int visit;
};
bool comparator(DataDetails d1, DataDetails d2)
{
    if (d1.bur != d2.bur)
        return (d1.bur < d2.bur);
    else
    {
        if (d1.ari != d2.ari)
            return (d1.ari < d2.ari);
        return (d1.pno < d2.pno);
    }
}

bool comparatorPno(DataDetails d1, DataDetails d2)
{
    return (d1.pno < d2.pno);
}

int main()
{
```

```

cout << "          SJF SCHEDULING" << endl;

int size, t = 0, ch = 0;

float avgtat = 0, avgwt = 0;

cout << "Enter no of process :";

cin >> size;

cout << "AT BT" << endl;

vector<DataDetails> vrr(size);

for (int i = 0; i < size; i++)
{
    cin >> vrr[i].ari >> vrr[i].bur;

    vrr[i].pno = i + 1;

    vrr[i].ct = vrr[i].tat = vrr[i].wt = 0;

    vrr[i].visit = 0;
}

sort(vrr.begin(), vrr.end(), comparator);

for (int k = 0; k < size; k++)
{
    int ch = 0;

    for (int i = 0; i < size; i++)
    {
        if ((vrr[i].visit == 0) && t >= vrr[i].ari)
        {
            t += vrr[i].bur;

            vrr[i].ct = t;

            vrr[i].tat = vrr[i].ct - vrr[i].ari;

            vrr[i].wt = vrr[i].tat - vrr[i].bur;

            avgtat += vrr[i].tat;

            avgwt += vrr[i].wt;

            vrr[i].visit = 1;

            ch = 1;

            break;
        }
    }
}

```

```

    }
}
if (!ch)
{
    t++;
    k--;
}
}

sort(vrr.begin(), vrr.end(), comparatorPno);

cout << endl;

cout << "          SOLUTION" << endl;

cout << "PN " << "AT " << "BT " << "CT " << "TAT " << "WT " << endl;

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

    cout << "P" << vrr[i].pno << " " << vrr[i].ari << " " << vrr[i].bur << " " << vrr[i].ct << " " <<
vrr[i].tat << " " << vrr[i].wt << " " << endl;

    avgtat = avgtat / size;

    cout << "Average TurnAroundTime is :" << avgtat << endl;

    avgwt = avgwt / size;

    cout << "Average WaitingTime is :" << avgwt << endl;

    return 0;
}

```

Output

Clear

### SJF SCHEDULING

Enter no of process :5

AT BT

2 6

5 2

1 8

0 3

4 4

### SOLUTION.

PN	AT	BT	CT	TAT	WT
P1	2	6	9	7	1
P2	5	2	11	6	4
P3	1	8	23	22	14
P4	0	3	3	3	0
P5	4	4	15	11	7

Average TurnAroundTime is :9.8

Average WaitingTime is :5.2

Output

Clear

### SJF SCHEDULING

Enter no of process :5

AT BT

5 7

3 2

9 1

1 3

2 2

### SOLUTION

PN	AT	BT	CT	TAT	WT
P1	5	7	15	10	3
P2	3	2	8	5	3
P3	9	1	16	7	6
P4	1	3	4	3	0
P5	2	2	6	4	2

Average TurnAroundTime is :5.8

Average WaitingTime is :2.8