

EXPERIMENT NO 2

Date of Completion: - 09th February 2022

Name: - Tambe Manish Mangesh

Roll No: - 221080

Gr No: - 22120186, **Batch:** - 'A3'

AIM: - Implement following programs to exhibit UNIX Process Control “Program where parent process sorts array elements in ascending order and child process sorts array elements in descending order”. Show the demonstration of walk() and zombie process”

Theory Related Questions: -

1. What is Process: - Process is an program in execution. & OS time shares CPU across multiple processes & visualizes the CPU.

2. How to create a new process: - A new process can be created by the fork() system call. The new process consists of a copy of the address space of the original process. fork() creates new process from existing process. Existing process is called the parent process and the process is created newly is called child process.

3. Fork(): - By using an fork() system call an new process is created by making a copy of parents memory image. The new process is added to the OS process list and schedule. Then parent and child process start executing just after fork(). Parent and child executes & modify the memory & data independently.

4. Exec(): - After implementing an fork(), parent and child are running same code. It's not useful if parent wants to perform some different operations by child in that case we are using an exec() system call & when child process calls exec() system call its memory image would be replaced by another executable it will no longer execute the code of its parents that it has.

5. Orphan Processes: - Orphan processes are those processes that are still running even though their parent process has terminated or finished. A process can be orphaned intentionally or unintentionally. An intentionally orphaned process runs in the background without any manual support.

6. Zombie Processes: - A zombie process is a process whose execution is completed but it still has an entry in the process table. Zombie processes usually occur for child processes, as the parent process still needs to read its child's exit status.

Programs: -

1. Demonstration of fork, wait System Call

Code: -

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

void CreateArray(int *Arr, int iSize)
{
    int iElement = 0;
    int p = 0;
    int i = 0;
    int j = 0;
    int temp = 0;

    for(i = 0; i < iSize; i++)
    {
```

```

        printf("Enter the %d element in the array = \n", i + 1);
        scanf("%d",&iElement);
        Arr[i] = iElement;
    }

    printf("The Entered Array Elements Are = \n");

    for(i = 0; i < iSize; i++)
    {
        printf("%d\n",Arr[i]);
    }

    p = fork();

    if(p == 0)
    {
        printf("Child => PPID: %d PID: %d\n", getppid(), getpid());
        for(i = 0; i < iSize; i++)
        {
            for(j = i+1; j < iSize; j++)
            {
                if(Arr[i] < Arr[j])
                {
                    temp = Arr[i];
                    Arr[i] = Arr[j];
                    Arr[j] = temp;
                }
            }
        }

        printf("The Entered Array Elements In Descending Order Are = \n");
        for(i = 0; i < iSize; i++)
        {

```

```

        printf("%d\n",Arr[i]);
    }
}
else
if(p > 0)
{

    printf("Parent => PID: %d\n", getpid());

    for(int i = 0; i < iSize; i++)
    {
        for(int j = i+1; j < iSize; j++)
        {
            if(Arr[i] > Arr[j])
            {
                temp = Arr[i];
                Arr[i] = Arr[j];
                Arr[j] = temp;
            }
        }
    }

    printf("The Entered Array Elements In Ascending Order Are = \n");
    for(i = 0; i < iSize; i++)
    {
        printf("%d\n",Arr[i]);
    }

    wait(NULL);
}
else
{

```

```

        printf("Child process is created successfully but error while creating an child
process !\n");
    }
}

int main()
{

    int iSize = 0;
    int *Arr = NULL;

    printf("Enter the number of elements you want to add in the array = \n");
    scanf("%d",&iSize);

    Arr = (int*)malloc(sizeof(int)*iSize);

    CreateArray(Arr,iSize);

    return 0;
}

```

Output: -

```

manish@manish-HP-15-Notebook-PC: ~/OS Programs/Process system calls implementation using 'C'/Demonstration of fork, wait System Call
manish@manish-HP-15-Notebook-PC:~/OS Programs/Process system calls implementation using 'C'/Demonstration of fork, wait System Call$ gcc Array.c -o Myexe
manish@manish-HP-15-Notebook-PC:~/OS Programs/Process system calls implementation using 'C'/Demonstration of fork, wait System Call$ ./Myexe
Enter the number of elements you want to add in the array =
4
Enter the 1 element in the array =
2
Enter the 2 element in the array =
0
Enter the 3 element in the array =
1
Enter the 4 element in the array =
0
The Entered Array Elements Are =
2
0
1
0
Parent => PID: 10689
The Entered Array Elements In Ascending Order Are =
0
1
2
0
Child => PPID: 10689 PID: 10692
The Entered Array Elements In Descending Order Are =
0
2
1
0
manish@manish-HP-15-Notebook-PC:~/OS Programs/Process system calls implementation using 'C'/Demonstration of fork, wait System Call$

```

2. Demonstration of Zombie Process: -

Code: -

```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/wait.h>


void Demo();


int main()
{
    Demo();
    return 0;
}


void Demo()
{
    int p = 0;

    p = fork();

    if(p == 0)
    {
        printf("Child => PPID: %d PID: %d\n", getppid(), getpid());
```

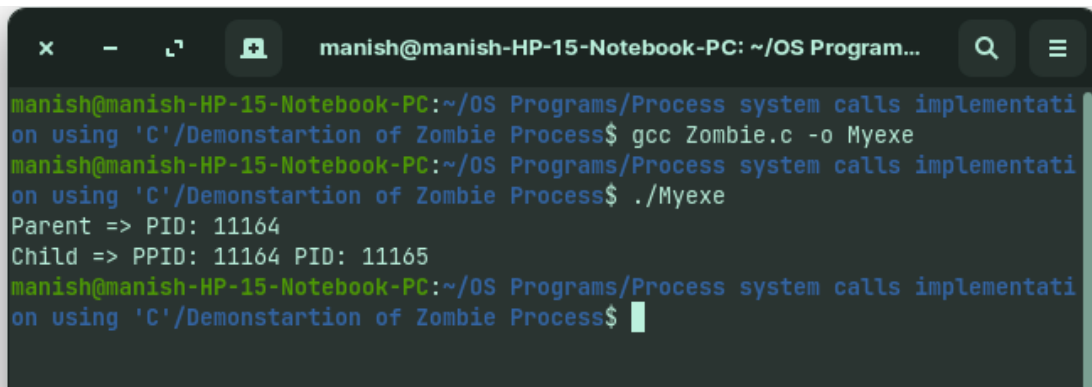
```

        exit(0);
    }
    else if(p > 0)
    {
        printf("Parent => PID: %d\n", getpid());

        sleep(10);
    }
    else
    {
        printf("Unable to create an child process !\n");
    }
}

```

Output: -



```

manish@manish-HP-15-Notebook-PC: ~/OS Program...
manish@manish-HP-15-Notebook-PC:~/OS Programs/Process system calls implementati
on using 'C'/Demonstartion of Zombie Process$ gcc Zombie.c -o Myexe
manish@manish-HP-15-Notebook-PC:~/OS Programs/Process system calls implementati
on using 'C'/Demonstartion of Zombie Process$ ./Myexe
Parent => PID: 11164
Child => PPID: 11164 PID: 11165
manish@manish-HP-15-Notebook-PC:~/OS Programs/Process system calls implementati
on using 'C'/Demonstartion of Zombie Process$

```

3. Demonstration of Orphan Process: -

Code: -

```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

void Demo();

int main()

{

    Demo();

    return 0;

}

void Demo()

{

    int p = 0;

    p = fork();

    if(p > 0)

    {

        printf("Parent Process : \n");

        printf("Parent => PID: %d\n", getpid());

    }

    else if(p == 0)

    {
```



```

    printf("Child Process Before Parent Die = \n");

    printf("Child => PPID: %d PID: %d\n", getppid(), getpid());

    sleep(1);

    printf("Child Process After Parent Die = \n");

    printf("Child => PPID: %d PID: %d\n", getppid(), getpid());

}

else

{

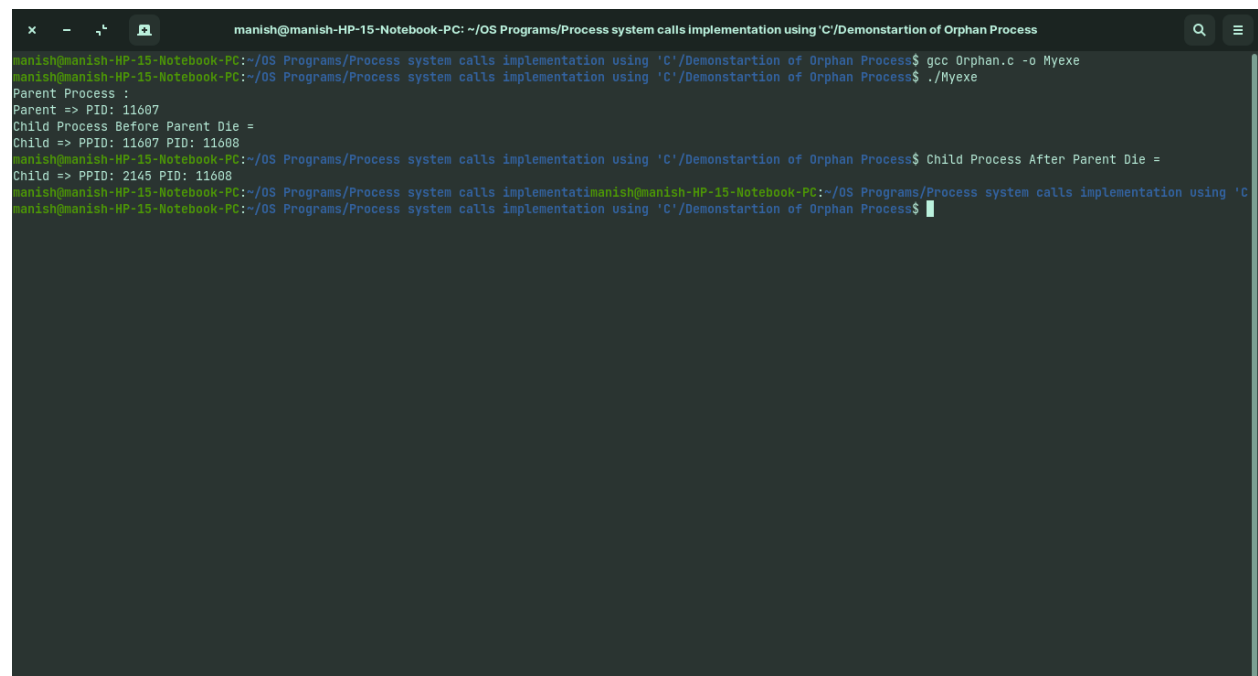
    printf("Unable to create an child process !\n");

}

}

```

Output: -



```

manish@manish-HP-15-Notebook-PC: ~/OS Programs/Process system calls implementation using 'C'/Demonstartion of Orphan Process
manish@manish-HP-15-Notebook-PC:~/OS Programs/Process system calls implementation using 'C'/Demonstartion of Orphan Process$ gcc Orphan.c -o Myexe
manish@manish-HP-15-Notebook-PC:~/OS Programs/Process system calls implementation using 'C'/Demonstartion of Orphan Process$ ./Myexe
Parent Process :
Parent => PID: 11607
Child Process Before Parent Die =
Child => PPID: 11607 PID: 11608
manish@manish-HP-15-Notebook-PC:~/OS Programs/Process system calls implementation using 'C'/Demonstartion of Orphan Process$ Child Process After Parent Die =
Child => PPID: 2145 PID: 11608
manish@manish-HP-15-Notebook-PC:~/OS Programs/Process system calls implementation using 'C'/Demonstartion of Orphan Process$
manish@manish-HP-15-Notebook-PC:~/OS Programs/Process system calls implementation using 'C'/Demonstartion of Orphan Process$

```

4. Demonstration of exec System call: -

Code: -

1. Exec.c

```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

    char *Args[] = {"Hello",NULL};

    printf("Process id of Exec.c = %d\n",getpid());

    execv("./Hello",Args);

    return 0;

}
```

2. Hello.c

```
#include<stdio.h>

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

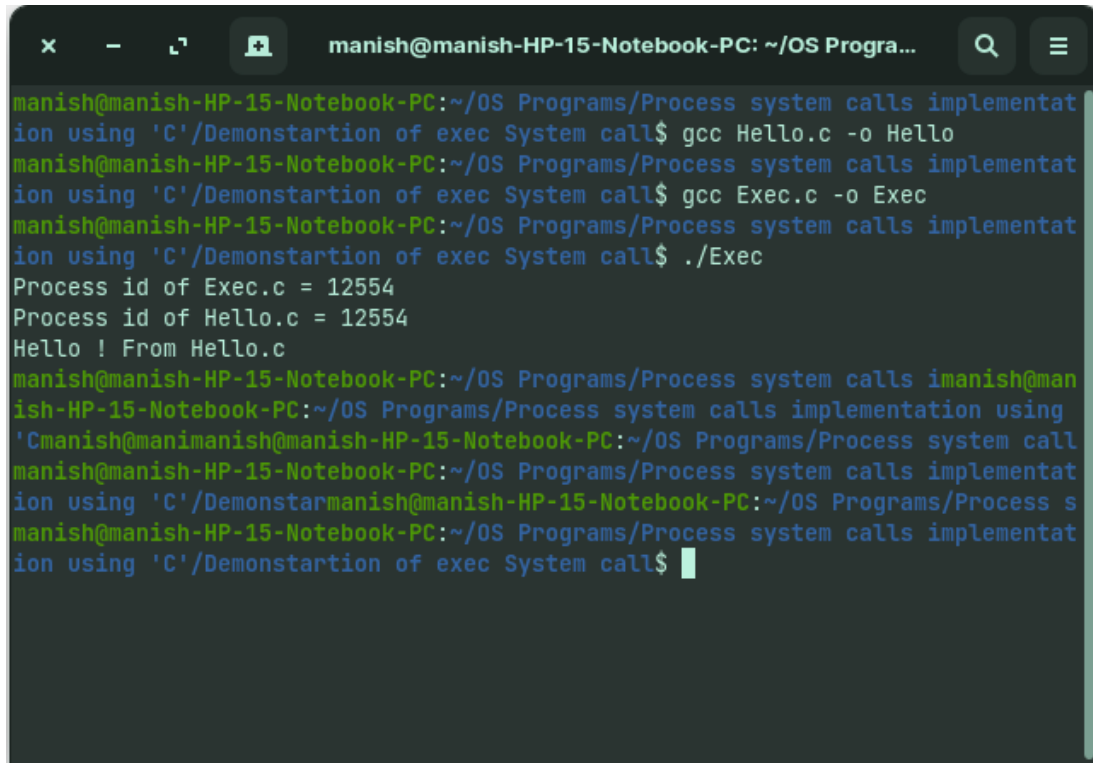
    printf("Process id of Hello.c = %d\n",getpid());

    printf("Hello ! From Hello.c\n");

    return 0;

}
```

Output: -



```
manish@manish-HP-15-Notebook-PC: ~/OS Progra...
manish@manish-HP-15-Notebook-PC:~/OS Programs/Process system calls implementat
ion using 'C'/Demonstartion of exec System call$ gcc Hello.c -o Hello
manish@manish-HP-15-Notebook-PC:~/OS Programs/Process system calls implementat
ion using 'C'/Demonstartion of exec System call$ gcc Exec.c -o Exec
manish@manish-HP-15-Notebook-PC:~/OS Programs/Process system calls implementat
ion using 'C'/Demonstartion of exec System call$ ./Exec
Process id of Exec.c = 12554
Process id of Hello.c = 12554
Hello ! From Hello.c
manish@manish-HP-15-Notebook-PC:~/OS Programs/Process system calls immanish@man
ish-HP-15-Notebook-PC:~/OS Programs/Process system calls implementation using
'C'manish@manish-HP-15-Notebook-PC:~/OS Programs/Process system call
manish@manish-HP-15-Notebook-PC:~/OS Programs/Process system calls implementat
ion using 'C'/Demonstarmanish@manish-HP-15-Notebook-PC:~/OS Programs/Process s
manish@manish-HP-15-Notebook-PC:~/OS Programs/Process system calls implementat
ion using 'C'/Demonstartion of exec System call$
```