

## **EXPERIMENT: -5 (Processes Management using System Calls)**

**Objective:** To introduce the concept of creating a new child process, performing operations on processes and working with orphan and zombie processes.

### **Lab Exercise Solution(s):**

**Q1.** Write a program using system calls for operation on process to simulate that n fork calls create  $(2^n - 1)$  child processes.

**Ans1.**

**Solution: -**

**Step1: -** nano 5\_1.c

**Step2: -**

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

```
#include<unistd.h>
```

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

```
int main(){
```

```
    int n;
```

```
    printf("# Enter the no. of times you want to run the fork system call: ");
```

```
    scanf("%d", &n);
```

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

```
        pid_t r;
```

```
        r = fork();
```

```
        if(r==0){
```

```
            printf("Current child process pid is %d \n", getpid());
```

```
        }
```

```
    }
```

```
    return 0;
```

```
}
```

Step3: - gcc 5\_1.c

Step4: - ./a.out

## nano 5\_1.c

```
GNU nano 7.2 Exp5_1.c
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>

int main(){
    int n;
    printf("# Enter the no. of times you want to run the fork system call: ");
    scanf("%d", &n);
    for(int i=0; i<n; i++){
        pid_t r;
        r = fork();
        if(r==0){
            printf("Current child process pid is %d \n", getpid());
        }
    }
    return 0;
}
```

## Output of the program

```
(shreygrg@ Linux23) - [~/Shrey_Garg]
$ nano Exp5_1.c

(shreygrg@ Linux23) - [~/Shrey_Garg]
$ gcc Exp5_1.c

(shreygrg@ Linux23) - [~/Shrey_Garg]
$ ./a.out
# Enter the no. of times you want to run the fork system call: 3
Current child process pid is 23697
Current child process pid is 23701
Current child process pid is 23700
Current child process pid is 23702
Current child process pid is 23698

Current child process pid is 23703
Current child process pid is 23699
```

**Q2. Write a program using systems for operations on processes to create a hierarchy of processes P1 → P2 → P3. Also print the id and parent id for each process.**

**Ans2.**

**Solution: -**

**Step1: - nano 5\_2.c**

**Step2: -**

```
#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include <stdlib.h>

int main()

{

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

    pid_t pid = fork();

    if(pid == 0)

    {

        printf("Child 1 PID : %d Parent PID : %d\n", (int) getpid(), (int) getppid());

        pid_t pid_1 = fork();

        if(pid_1 == 0)

        {

            printf("Child 2 PID : %d Parent PID (Child 1) : %d \n", (int) getpid(), (int) getppid());

            exit(0);

        }

        else

        {

            exit(0);

        }

    }

    else

    {

        exit(0);

    }

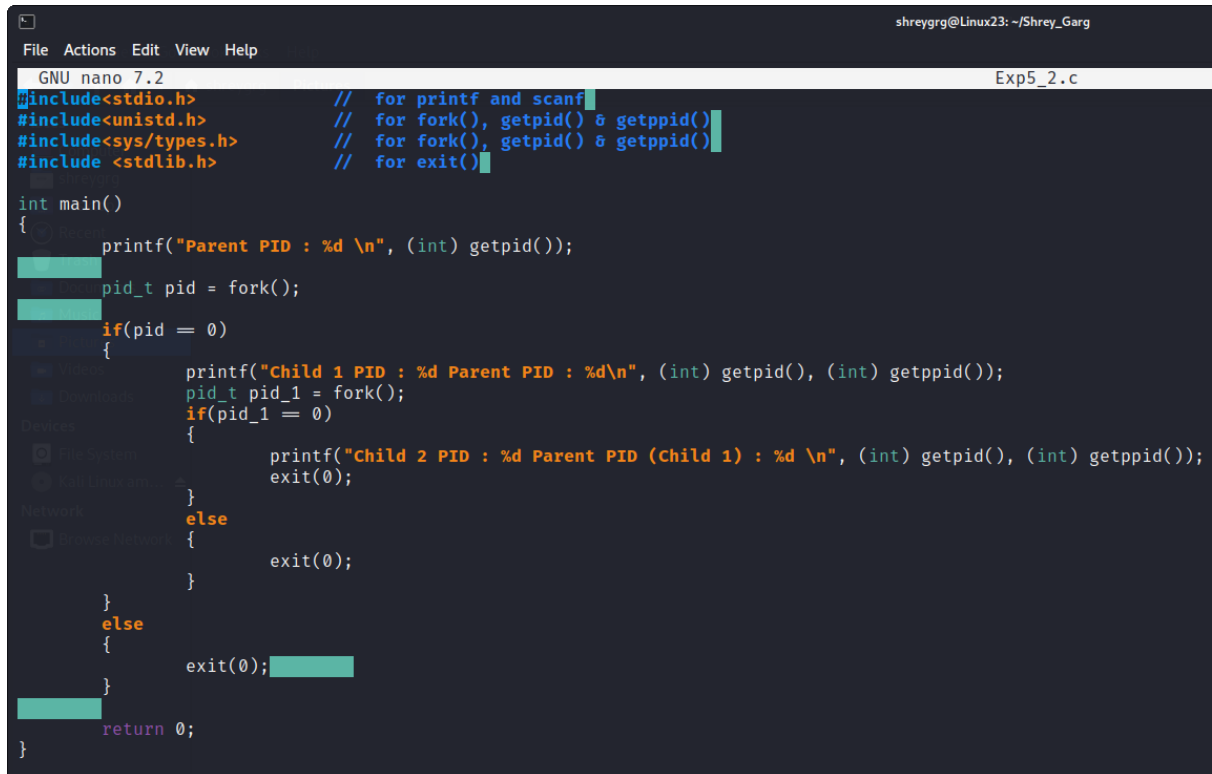
    return 0;

}
```

Step3: - gcc 5\_2.c

Step4: - ./a.out

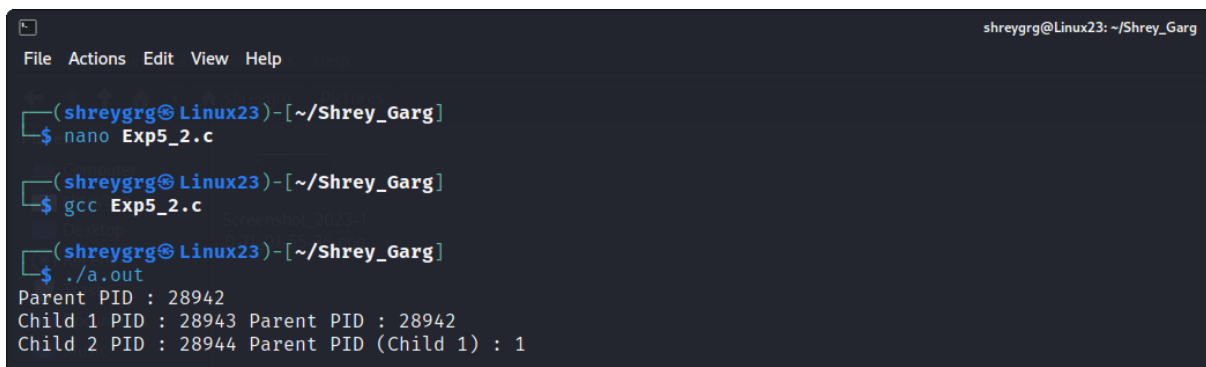
## nano 5\_2.c



```
GNU nano 7.2 Exp5_2.c
#include<stdio.h>           // for printf and scanf
#include<unistd.h>          // for fork(), getpid() & getppid()
#include<sys/types.h>       // for fork(), getpid() & getppid()
#include <stdlib.h>         // for exit()

int main()
{
    printf("Parent PID : %d \n", (int) getpid());
    pid_t pid = fork();
    if(pid == 0)
    {
        printf("Child 1 PID : %d Parent PID : %d\n", (int) getpid(), (int) getppid());
        pid_t pid_1 = fork();
        if(pid_1 == 0)
        {
            printf("Child 2 PID : %d Parent PID (Child 1) : %d \n", (int) getpid(), (int) getppid());
            exit(0);
        }
        else
        {
            exit(0);
        }
    }
    else
    {
        exit(0);
    }
    return 0;
}
```

## Output of the program



```
(shreygrg@ Linux23) - [~/Shrey_Garg]
$ nano Exp5_2.c

(shreygrg@ Linux23) - [~/Shrey_Garg]
$ gcc Exp5_2.c

(shreygrg@ Linux23) - [~/Shrey_Garg]
$ ./a.out
Parent PID : 28942
Child 1 PID : 28943 Parent PID : 28942
Child 2 PID : 28944 Parent PID (Child 1) : 1
```

**Q3. Write a program using system calls for operation on processes to create a hierarchy of processes as given in figure 1. Also, simulate process p4 as orphan and P5 as zombie.**

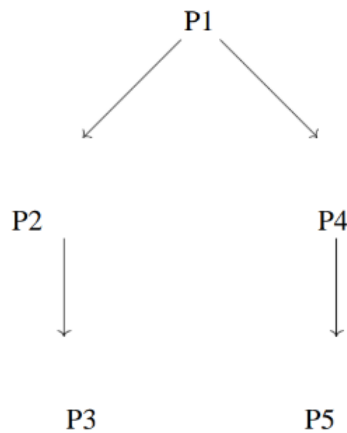


Figure 1: Hierarchy of Processes

**Ans3.**

**Solution: -**

**Step1: - nano 5\_3.c**

**Step2: -**

```

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

int main()
{
    printf("P1 PID : %d \n", (int) getpid());
    pid_t pid = fork();

    if(pid == 0)
    {
        printf("P4 PID : %d P1 PID : %d\n", (int) getpid(), (int) getppid());
        printf("Child process P4 is sleeping \n");
        pid_t pid_1 = fork();
        sleep(5);
        if(pid_1 == 0)
        {

```

```

        printf("P5 PID : %d P4 PID : %d \n", (int) getpid(), (int) getppid());
        printf("Zombie process P5's PID : %d \n", (int) getpid());
    }
    else{
        printf("Orphan child process P4's PID : %d \n", (int) getpid());
        printf("P4's New Parent PID : %d \n", (int) getppid());
    }
}
else
{
    pid = fork();
    if(pid == 0)
    {
        printf("P2 PID : %d P1 PID : %d\n", (int) getpid(), (int) getppid());
        pid_t pid_1 = fork();
        if(pid_1 == 0)
        {
            printf("P3 PID : %d P2 PID : %d \n", (int) getpid(), (int) getppid());
            exit(0);
        }
        else
        {
            exit(0);
        }
    }
    else
    {
        exit(0);
    }
}

return 0;
}

```

Step3: - gcc 5\_3.c

Step4: - ./a.out

## nano 5\_3.c

```
GNU nano 7.2 Exp5_3.c
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<stdlib.h>

int main()
{
    printf("P1 PID : %d \n", (int) getpid());
    pid_t pid = fork();
    if(pid == 0)
    {
        printf("P4 PID : %d P1 PID : %d\n", (int) getpid(), (int) getppid());
        printf("Child process P4 is sleeping \n");
        pid_t pid_1 = fork();
        sleep(5);
        if(pid_1 == 0)
        {
            printf("P5 PID : %d P4 PID : %d \n", (int) getpid(), (int) getppid());
            printf("Zombie process P5's PID : %d \n", (int) getpid());
        }
        else{
            printf("Orphan child process P4's PID : %d \n", (int) getpid());
            printf("P4's New Parent PID : %d \n", (int) getppid());
        }
    }
    else{
        pid = fork();
        if(pid == 0)
        {
            printf("P2 PID : %d P1 PID : %d\n", (int) getpid(), (int) getppid());
            pid_t pid_1 = fork();
            if(pid_1 == 0)
            {
                printf("P3 PID : %d P2 PID : %d \n", (int) getpid(), (int) getppid());
                exit(0);
            }
            else
            {
                exit(0);
            }
        }
        else
        {
            exit(0);
        }
    }
    return 0;
}
```

## Output of the program

```
shreygrg@Linux23: ~/Shrey_Garg
$ nano Exp5_3.c

shreygrg@Linux23: ~/Shrey_Garg
$ gcc Exp5_3.c

shreygrg@Linux23: ~/Shrey_Garg
$ ./a.out
P1 PID : 31646
P2 PID : 31648 P1 PID : 31646
P4 PID : 31647 P1 PID : 1
Child process P4 is sleeping

P3 PID : 31650 P2 PID : 1
Orphan child process P4's PID : 31647
P4's New Parent PID : 1
P5 PID : 31649 P4 PID : 31647
Zombie process P5's PID : 31649
```

**Q4. Write a program using system calls for operation on processes to create a hierarchy of processes 2. Also, simulate P4 as an orphan process and P7 as a zombie.**

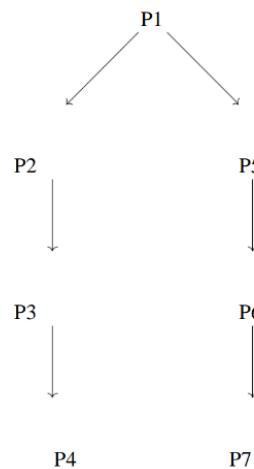


Figure 2: Hierarchy of Processes

**Ans4.**

**Solution: -**

**Step1: - nano 5\_4.c**

**Step2: -**

**#include<stdio.h>**

**#include<unistd.h>**

**#include<sys/types.h>**

**#include<stdlib.h>**

**int main()**

**{**

**printf("P1 PID : %d \n", (int) getpid());**

**pid\_t pid = fork();**

**if(pid == 0)**

**{**

**printf("P5 PID : %d Parent P1 PID : %d\n", (int) getpid(), (int) getppid());**

**pid\_t pid\_1 = fork();**



```

        if(pid_1 == 0)
        {
            printf("P6 PID : %d Parent P5 PID : %d \n", (int) getpid(), (int)
getppid());

            pid_t pid_2 = fork();
            sleep(5);

            if(pid_2 == 0)
            {
                printf("Zombie process P7's PID: %d \n", (int) getpid());
                printf("Parent P6 PID : %d \n", (int) getppid());
            }
            else
            {
                exit(0);
            }
        }
        else
        {
            exit(0);
        }
    }

else
{
    pid = fork();
    if(pid == 0)
    {

```

```

    printf("P2 PID : %d Parent P1 PID : %d\n", (int) getpid(), (int)
getppid());

    pid_t pid_1 = fork();
    if(pid_1 == 0)
    {
        printf("P3 PID : %d Parent P2 PID : %d \n", (int) getpid(), (int)
getppid());

        pid_t pid_2 = fork();
        if(pid_2 == 0)
        {
            sleep(3);
        }
        else
        {
            printf("Orphan child process P4's PID : %d \n", (int)
pid_2);

            printf("P4's New Parent PID : %d \n", (int) getppid());
        }
    }
    else
    {
        exit(0);
    }
}

return 0;
}

```

## nano 5\_4.c

```
shreygrg@Linux23: ~/Shrey_Garg
File Actions Edit View Help
GNU nano 7.2 Exp5_4.c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>

int main()
{
    printf("P1 PID : %d\n", (int) getpid());
    pid_t pid = fork();
    if(pid == 0)
    {
        printf("P5 PID : %d Parent P1 PID : %d\n", (int) getpid(), (int) getpid());
        pid_t pid_1 = fork();
        if(pid_1 == 0)
        {
            printf("P6 PID : %d Parent P5 PID : %d\n", (int) getpid(), (int) getpid());
            pid_t pid_2 = fork();
            sleep(5);
            if(pid_2 == 0)
            {
                printf("Zombie process P7's PID: %d\n", (int) getpid());
                printf("Parent P6 PID : %d\n", (int) getpid());
            }
            else
            {
                exit(0);
            }
        }
        else
        {
            exit(0);
        }
    }
    else
    {
        pid = fork();
        if(pid == 0)
        {
            printf("P2 PID : %d Parent P1 PID : %d\n", (int) getpid(), (int) getpid());
            pid_t pid_1 = fork();
            if(pid_1 == 0)
            {
                printf("P3 PID : %d Parent P2 PID : %d\n", (int) getpid(), (int) getpid());
                pid_t pid_2 = fork();
                if(pid_2 == 0)
                {
                    sleep(3);
                }
                else
                {
                    printf("Orphan child process P4's PID : %d\n", (int) pid_2);
                    printf("P4's New Parent PID : %d\n", (int) getpid());
                }
            }
            else
            {
                exit(0);
            }
        }
    }
    return 0;
}
```

## Output of the program

```
shreygrg@Linux23: ~/Shrey_Garg
File Actions Edit View Help
(shreygrg@Linux23)~-[~/Shrey_Garg]
$ nano Exp5_4.c
(shreygrg@Linux23)~-[~/Shrey_Garg]
$ gcc Exp5_4.c
(shreygrg@Linux23)~-[~/Shrey_Garg]
$ ./a.out
P1 PID : 40102
P5 PID : 40103 Parent P1 PID : 40102
P2 PID : 40104 Parent P1 PID : 1
P3 PID : 40105 Parent P2 PID : 1
Orphan child process P4's PID : 40106
P4's New Parent PID : 1

P6 PID : 40107 Parent P5 PID : 1
(shreygrg@Linux23)~-[~/Shrey_Garg]
$ Zombie process P7's PID: 40109
Parent P6 PID : 40107
```

**Submitted to: -**

**Mr. Ashish Kumar Singh**

**Section: K22QY**

**Submitted By: -**

**Shrey Garg  
(12218692)**