Heaven's Light is Our Guide

Rajshahi University of Engineering and Technology, Bangladesh

Department of Computer Science and Engineering

Course No: CSE 3202
Course Title: Sessional Based on CSE 3201

Submitted To:

Mohiuddin Ahmed
Lecturer
Department of Computer Science and Engineering
Rajshahi University of Engineering and Technology.

Submitted By:

Ashraf-Ul-Alam
Roll: 1803070,
Section: B,
Department of Computer Science and Engineering,
Rajshahi University of Engineering and Technology.

Date of Submission: 11 December 2022

Experiment No: 4

Experiment Name: Process creation.

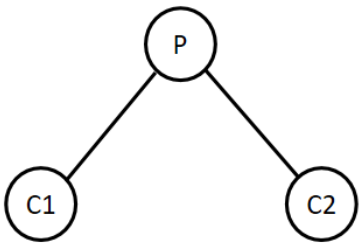
Introduction:

Process creation is achieved through the fork() system call.After the fork() system call, there are two processes, the existing one is called the parent process and the newly created one is called the child process. The fork() system call returns either of the three values −

- Negative value to indicate an error (unsuccessful in creating the child process).
- Returns a zero for child process.
- Returns a positive value for the parent process. This value is the process ID of the newly created child process.

Command:

Tree-1



Code:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    int n1 = fork();

    if (n1 > 0) //parent copy
    {
        printf("\n[%d] This is Root parent p\n", getpid());
        int n2 = fork();
        sleep(20);
        if (n2 == 0) //c2
```

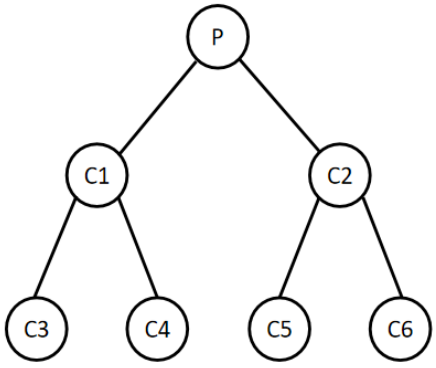
```
        {
            printf("[%d] This is Child 2 having parent %d\n",
getpid(), getppid());
        }

    }
    if (n1 == 0) //c1
    {
        printf("[%d] This is Child 1 having parent %d\n",
getpid(), getppid());
    }
    return 0;
}
```

Output:

```
amit@DESKTOP-V5UJJLP:/mnt/f/32/01 OS/Lab/Lab4$ gcc fork.c -o fork.out
[1]+  Done                    ./nfork.out
amit@DESKTOP-V5UJJLP:/mnt/f/32/01 OS/Lab/Lab4$ ./fork.out &
[1] 227
amit@DESKTOP-V5UJJLP:/mnt/f/32/01 OS/Lab/Lab4$
[227] This is Root parent p
[228] This is Child 1 having parent 227
pstree -p
init(1)─┬─init(10)───init(11)───bash(12)─┬─fork.out(227)─┬─fork.out(228)
        │                                │               └─fork.out(229)
        │                                └─pstree(230)
        └─{init}(9)
amit@DESKTOP-V5UJJLP:/mnt/f/32/01 OS/Lab/Lab4$ [229] This is Child 2 having parent 227
```

Tree-2



Code:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    int n1 = fork(); // parent p
    if (n1 > 0) // parent p copy
    {
        printf("\n[%d] This is Root parent p\n", getpid());

        int n2 = fork();
        if (n2 == 0) // child c2
        {
            // sleep(10);
            printf("[%d] This is Child 2 having parent %d\n",
getpid(), getppid());

            int n3 = fork();
            if (n3 == 0) // child c5
            {
                printf("[%d] This is Child 5 having parent
%d\n", getpid(), getppid());
            }
            if (n3 > 0) // child c2 copy
            {
                int n4 = fork();
                if (n4 == 0) // child 6
                {
                    printf("[%d] This is Child 6 having parent
%d\n", getpid(), getppid());
                }
            }
        }
        sleep(20);
    }

    if (n1 == 0) // c1
    {
        printf("[%d] This is Child 1 having parent %d\n",
getpid(), getppid());

        int n5 = fork();

        if (n5 > 0) // child c1 copy
        {
```

```
            int n6 = fork();
            if (n6 == 0) //child c4
            {
                printf("[%d] This is Child 4 having parent
%d\n", getpid(), getppid());
            }
        }
        if (n5 == 0) // child c3
        {
            printf("[%d] This is Child 3 having parent %d\n",
getpid(), getppid());
        }
        sleep(10);
    }
    return 0;
}
```

Output:

```
amit@DESKTOP-V5UJJLP:/mnt/f/32/01 OS/Lab/Lab4$ gcc nfork.c -o nfork.out
amit@DESKTOP-V5UJJLP:/mnt/f/32/01 OS/Lab/Lab4$ ./nfork.out &
[1] 214
amit@DESKTOP-V5UJJLP:/mnt/f/32/01 OS/Lab/Lab4$
[214] This is Root parent p
[215] This is Child 1 having parent 214
[216] This is Child 2 having parent 214
[217] This is Child 3 having parent 215
[218] This is Child 5 having parent 216
[219] This is Child 4 having parent 215
[220] This is Child 6 having parent 216
pstree -p
init(1)─┬─init(10)───init(11)───bash(12)─┬─nfork.out(214)─┬─nfork.out(215)─┬─nfork.out(217)
        │                                │                │                └─nfork.out(219)
        │                                │                └─nfork.out(216)─┬─nfork.out(218)
        │                                │                                 └─nfork.out(220)
        │                                └─pstree(221)
        └─{init}(9)
```

Discussion:

Here, for some processes in tree-1 and tree-2, we use sleep(). sleep() causes the current thread to suspend execution for a specified period. Other operations of the CPU will function properly but the sleep() function will sleep the present executable for the specified time by the thread. Such as, in case of tree-1, parent sleeps for 20 seconds and newly created child C2 executes. This also helps to create the pstree. Proper use of sleep() might help to prevent Zombie and Orphan processes.