Experiment – 5

NAME : Madhuram Brijeshkumar Modi

ROLL NO. : 21BCP102

DIV, GROUP : 2,G3

Processes

Fork System Call: The fork system call is used for creating a new process, which is called the child process, which runs concurrently with the process that makes the fork() call (parent process). After a new child process is created, both processes will execute the next instruction following the fork() system call.

It takes no parameters and returns an integer value. Below are different values returned by fork().

Negative Value: creation of a child process was unsuccessful.
Zero: Returned to the newly created child process.
Positive value: Returned to parent or caller. The value contains process ID of newly created child process.

1. Write a program to create a simple child process.

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

int main()
 {
        fork();
        printf("Hello, World\n");
        return 0;
}
```

```
Hello World!
Hello World!
```

2. Write a program to create multiple child processes.

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

int main()
{
        fork();
```

```
        fork();
        fork();
        printf("Hello, World\n");
}
```

Output:

```
Hello, World
Hello, World
Hello, World
Hello, World
Hello, World
Hello, World
Hello, World
Hello, World
```

3. Write a program to check the return status of parent and child processes.

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

void forkexample()
{
        if(fork() == 0)
        printf("Hello, from the child\n");
        else
        printf("Hello, from the parent\n");
}
int main()
 {
        forkexample();
        return 0;
}
```

Output:

```
Hello, from the parent
Hello, from the child
```

4. Write a program to get the process IDs of a child process and its parent process.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        int pid = fork();
        if(pid == 0)
        {
                printf("I am child process. My Process ID: %d\n", getpid());
                printf("My parent's Process ID: %d\n", getppid());
                printf("Child terminates\n");
                exit(0);
        }
        else {
                printf("I am parent process. My Process ID: %d\n", getpid());
                printf("My parent's Process ID: %d\n", getppid());
                printf("Parent terminates\n");
        }
        return 0;
}
```

Output:

```
I am parent process. My Process ID: 13453
My parent's Process ID: 13446
Parent terminates
I am child process. My Process ID: 13454
My parent's Process ID: 1
Child terminates
```

5. Write a program to create a zombie process.

Zombie Process:
A process which has finished the execution but still has entry in the process table to report to its parent process is known as a zombie process. A child process always first becomes a zombie before being removed from the process table. The parent process reads the exit status of the child process which reaps off the child process entry from the process table.
Zombie process is also known as ***"dead"*** process.

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

int main()
{
```

```
        int child_pid = fork();
        if(child_pid > 0)
        sleep(60);
        else exit(0);
        return 0;
}
```

6. Write a program to create an orphan process.

Orphan Process:

A process whose parent process no more exists i.e. either finished or terminated without waiting for its child process to terminate is called an orphan process.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        int pid = fork();
        if(pid == 0)
        {
                printf("I am Child process. My Process ID: %d\n", getpid());
                printf("My Parent's Process ID: %d\n", getppid());
                sleep(30);
                printf("After sleeping my Process ID: %d\n", getpid());
                printf("After sleeping my Parent's Process ID: %d\n", getppid());
                printf("Child terminates\n");
                exit(0);
        }
        else
         {
                sleep(20);
                printf("I am Parent. My Process ID: %d\n", getpid());
                printf("My Parent's Process ID: %d\n", getppid());
                printf("Parent terminates\n");
        }
        return 0;
}
```

Output:

```
I am Child process. My Process ID: 13537
My Parent's Process ID: 13536
I am Parent. My Process ID: 13536
My Parent's Process ID: 13529
Parent terminates
After sleeping my Process ID: 13537
After sleeping my Parent's Process ID: 1
Child terminates
```