Practical 4

- CODES

1. **FORK.c**

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

int main() {
   pid_t pid;

   pid = fork();

   if (pid == 0) {
      // Child process
      printf("Child Process exiting\n");
   }
   else {
      // Parent process
      sleep(10);   // parent does not call wait()
      printf("Parent Process running\n");
   }

   return 0;
}
```

2. **Orphan.c**

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

int main() {
   pid_t pid;

   pid = fork();

   if (pid == 0) {
      // Child process
      sleep(5);   // parent exits before child
      printf("Child Process (Orphan)\n");
      printf("PID = %d\n", getpid());
```

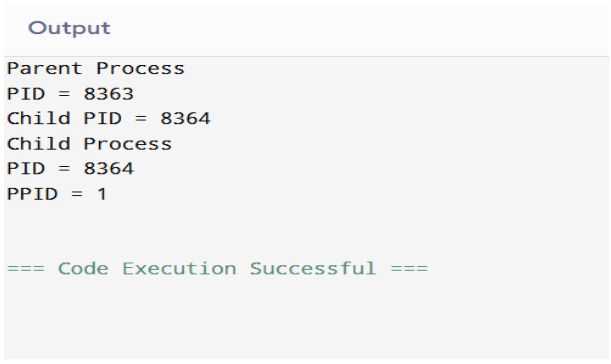
```
        printf("PPID = %d\n", getppid());
    }
    else {
        // Parent process
        printf("Parent Process exiting\n");
    }

    return 0;
}
```

3. **Zombie.c**

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

int main() {
    pid_t pid;

    pid = fork();

    if (pid == 0) {
        // Child process
        printf("Child Process exiting\n");
    }
    else {
        // Parent process
        sleep(10);   // parent does not call wait()
        printf("Parent Process running\n");
    }

    return 0;
}
```

❖ OUTPUT

1. **FORK.c**



```
Output

Parent Process
PID = 8363
Child PID = 8364
Child Process
PID = 8364
PPID = 1


=== Code Execution Successful ===
```

2. **Orphan**

```
Output

Parent Process exiting


=== Code Execution Successful ===
```

3. **Zombie**

```
Child Process exiting
Parent Process running


=== Code Execution Successful ===
```