~~~Either Orphaned or Zombied~~~

If it is left running, it might be orphaned or zombified. I am talking about the outcomes of the fork() function call. When fork() is called, a child process would be created. The concepts of the orphan process and the zombie process are related to the child process. The child and the parent processes are initially not timed, so one of the following situations might happen.

The Parent process completes its execution while the Child process is still executing; this would lead to the child process becoming an orphan process. Once the parent completes its execution and exits, the PPID of the child is handed over to init.

The second scenario would be, the child completed its execution but the parent is still executing. The child process enters a state where it is no longer able to execute instructions, becoming a dead or zombie process. This occurs because the parent process needs to gather the exit status of its child. Once the exit status is acquired, the process manager removes the child from the zombie state. In cases where the parent process terminates before collecting the child's exit status, the kernel considers it an orphan process that is then adopted by the "init" process.

There is not much problem in the occurrence of the first scenario, but in the second scenario, If a large number of zombie child processes are created, their entry would be maintained in the kernel’s process table, and many occurrences would fill up the process table that would prevent the creation of new processes. Zombies cannot be killed by a signal, so the only way to remove them is by killing the parent process of the zombies. As the zombies have ppid as the init pid(1), the entire system needs to be shut in order to kill the zombies.

To avoid the child from becoming orphan, the parent should wait until the child process is completed. To avoid the child from becoming zombie, the child should send an exit status to the parent and the parent should wait until the exit status is received from the child.

```c
// Illustration of Orphan Child Process                                                                    2
   #include<stdio.h>
   #include<sys/types.h>
   #include<unistd.h>

   void main()
   {
       if(fork()==0)
      {
          // Child process Execution
          printf("In Child Process\n");
          printf("PID: %d\tPPID: %d\n",getpid(),getppid());
          sleep(7);   // Child process turns into orphan process as the parent will be exiting after 3 seconds
          printf("Child Process terminated\n");

      }
      else
      {
          //Parent Process Execution

          printf("In Parent Process\n");
          printf("PID: %d\tPPID: %d\n",getpid(),getppid());
          sleep(3);   // Exit after 3 seconds
          printf("Parent Process Terminated\n");

      }
  }
```

Output:

```
In Parent Process
In Child Process
PID: 287        PPID: 286
PID: 286        PPID: 53
Parent Process Terminated
root@DESKTOP-5CRS5B4:/home/yash/programs# Child Process terminated
```

Executed by CHILD

Executed by PARENT

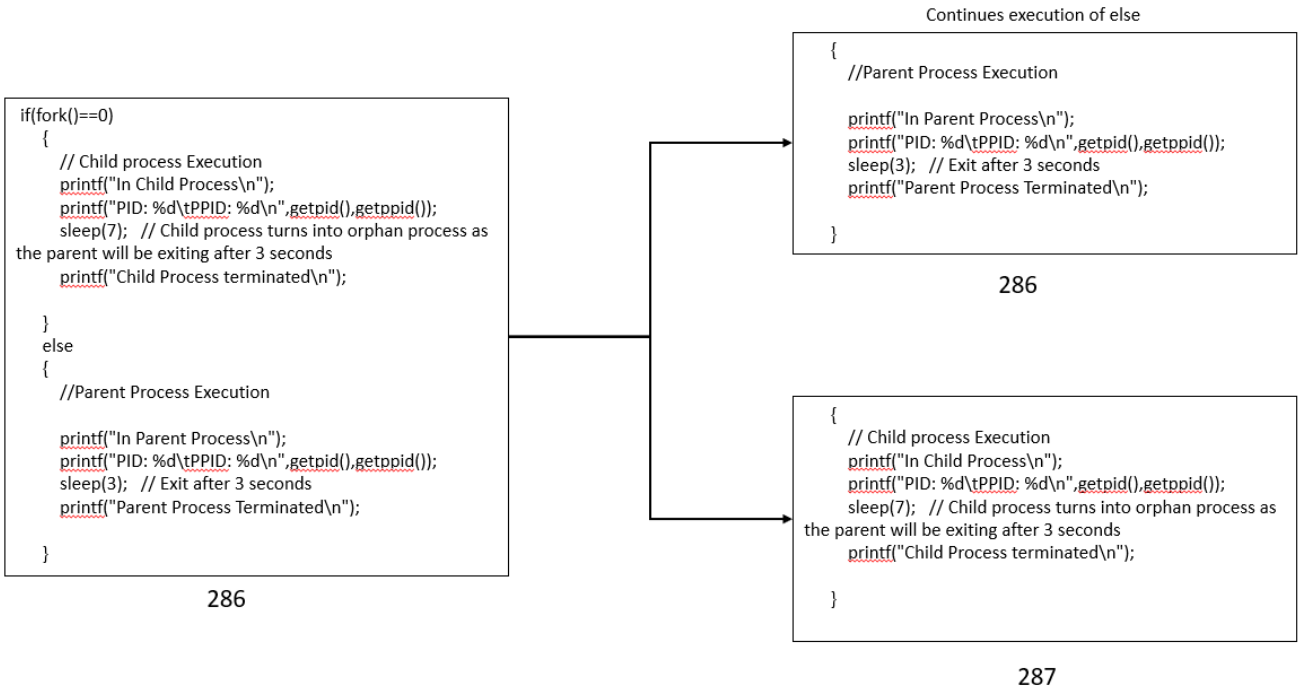
With the ps command:

```
root         91     1  0  2555   132    0 12:36 tty2     00:00:00 /init
yash         92    91  0  3561  1536    0 12:36 tty2     00:00:00 -bash
root        286    53  0  2703   656    0 13:58 pts/0    00:00:00 ./child_zombie
root        287   286  0  2703   260    0 13:58 pts/0    00:00:00 ./child_zombie
```

After the parent exits:

```
root         91     1  0  2555   132    0 12:36 tty2     00:00:00 /init
yash         92    91  0  3561  1536    0 12:36 tty2     00:00:00 -bash
root        287     1  0  2703   260    0 13:58 pts/0    00:00:00 ./child_zombie
```

The condition if(fork()==0) is executed both in parent and child process. In the parent process, the else condition is executed as fork() returns PID of the child upon successful creation and returns 0 to the child.



Zombie Process:

```c
// Illustration of transformation of Child process Zombie Process
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>

void main()
{
        printf(" Current process PID: %d\tPPID:%d\n",getpid(),getppid());

        if(fork()==0)
        {
                printf("Enetered Child Process\n");
                //Child execution
                printf("Executing in Child PID:%d\tPPID:%d\n",getpid(),getppid());
                sleep(2);
                printf("Exited Child Process\n");

        }
        else{
                //Parent execution

                printf("Entered Parent Process\n");
                printf("Executing in Parent PID:%d\tPPID:%d\n",getpid(),getppid());
                sleep(7);
                printf("Exited Parent Process\n");
        }
}
```

Output: The child is exited while the parent is still executing.

```
 Current process PID: 479        PPID:53
Entered Parent Process
Executing in Parent PID:479     PPID:53
Enetered Child Process
Executing in Child PID:480      PPID:479
Exited Child Process
Exited Parent Process
```

Ps:

Child Created

```
0 S root       479    53  0  80   0 -  2703 -        656    0 16:42 pts/0    00:00:00 ./zombie
0 S root       480   479  0  80   0 -  2703 -        204    0 16:42 pts/0    00:00:00 ./zombie
```

Child Defunctional

```
0 S root       479    53  0  80   0 -  2703 -        656    0 16:42 pts/0    00:00:00 ./zombie
0 Z root       480   479  0  80   0 -     0 -          0    0 16:42 pts/0    00:00:00 [zombie] <defunct>
```