Orphans and Zombies

Orphan process

- The orphan process or zombie process, these concepts are related with child processes. The lifetimes of parent and child processes are usually not the same.
- Parent execution completed and Child is still executing- here child becomes orphan. The orphaned child is adopted by init.
- After a child‘s parent terminates, a call to getppid() will return the value 1.

Zombie process

- Child execution completed, Parent is still executing- here child becomes zombie.
- A zombie is dead process which does not have any pending instruction to execute. It is de functional
- When child completes before parent, the kernel deals with this situation by turning the child into a zombie.
- The reason why zombie process is created is that *parent should collect the exit status of its child.* Once parent gets exit status of child, process manager removes child from zombie state.

Zombie process

- To avoid child becoming zombie, child should send exit status (exit() or _exit()) and parent should wait to collect the exit status (wait() or waitpid()) .

  _exit() system call

- A process may terminate in two general ways. One of these is abnormal termination, caused by the delivery of a signal whose default action is to terminate the process. Alternatively, a process can terminate normally, using the _exit() system call.

```
void  _exit( int status );
```

exit()... a library function

Programs generally don‘t call _exit() directly, but instead call the exit() library function, which performs various actions before calling _exit().

```
void exit(int status);
```

C standard specifies two constants, EXIT_SUCCESS (0) and EXIT_FAILURE (1). These are macros defined in stdlib.h. The following actions are performed by exit():

1) Exit handlers (functions registered with atexit() and on_exit()) are called, in reverse order of their registration

2) The stdio stream buffers are flushed.

3) The _exit() system call is invoked, using the value supplied in status.