**Zombie vs. Orphan Processes…**

*Hello Good Reader…*

Let’s begin with relationships between parent & child processes…

*Possible cases on parent-child process relationship...*

**case 1 :** parent process waits for the child process to be executed, by invoking the wait() system call.

**case 2 :** parent doesn’t wait for the child process, i.e. both execute parallelly.

**Let’s first we discuss about ‘zombie process’…**

*Zombie Process…*

* “ The process which has finished its execution but still has an entry in the process table to report to its parent process is known as zombie process. ”
* The parent process reads the exit status of child process (via wait() system call), which eventually reaps off the child process entry from the process table.

*What happens with the zombie processes ?*

* Zombie processes can actually be removed using the wait() system call.
* what wait() system call do is, after completion of its child process, parent process collects the exit status of the child process and the entire entry of the child process in the process table will be completely removed. In other words, the child process is now completely killed and has no trace in the system.
* Actually, freeing-up the data entry (of child process) from process table is known as — ‘Reaping the child’ or ‘Reaping’.

*why lots of zombie processes are harmful to the system ???*

Let’s answer this question… be with me Guys.

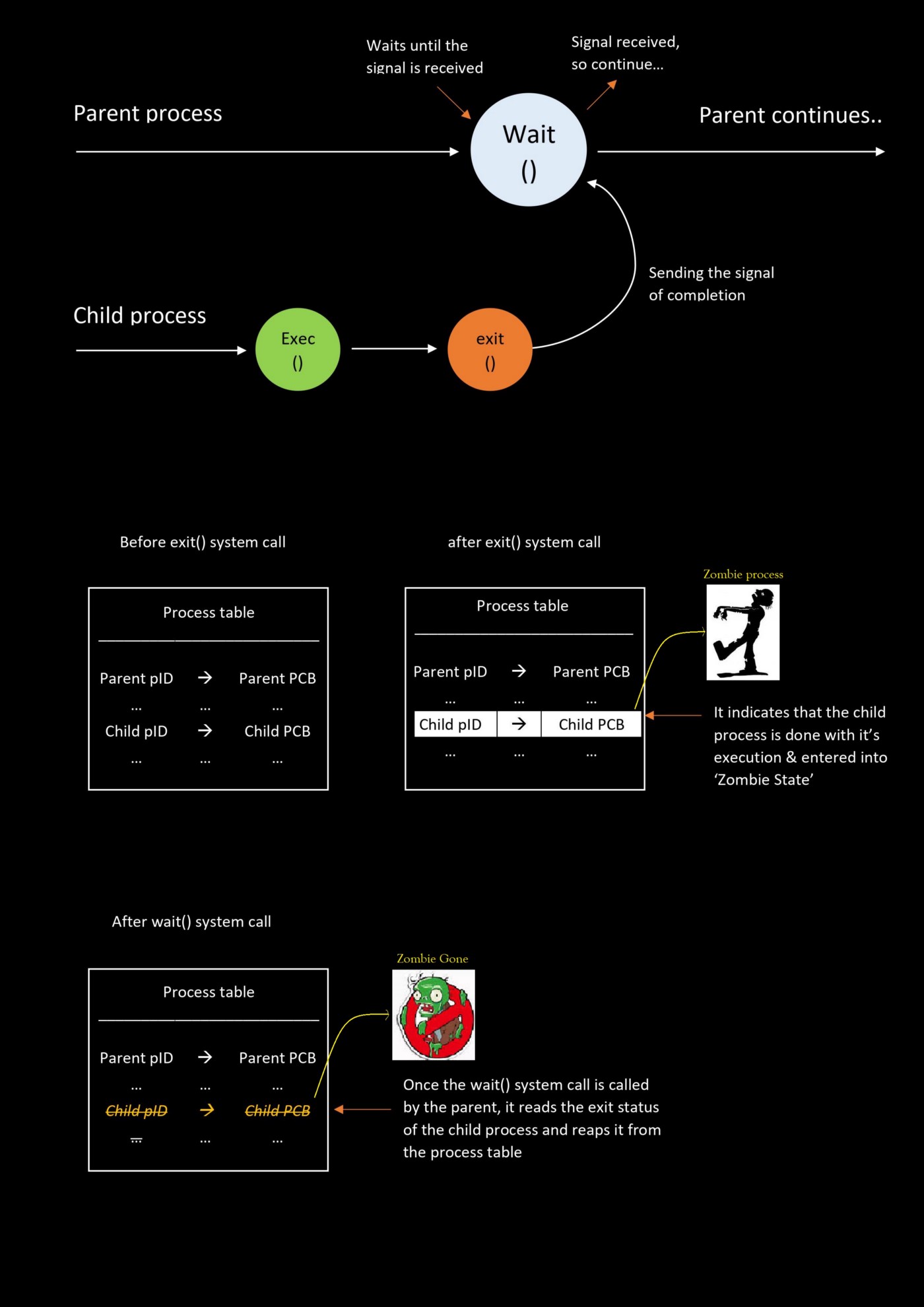
* There is one process table available per system & it’s must be of finite size, right ??
* As you can see in the figure, zombie processes occupies some space in the process table. So, too many zombie processes causes the process table to be more crowded or sometimes absolutely full.
* Due to this reason, the system is now not able to generate the new processes which may lead the system come to standstill.

Now, you know what is a ‘Zombie process’. Understanding it is the prerequisite for discussing upon the different cases.

***case 1 : parent process waits for the child process to be executed, by invoking the wait() system call.***

* wait() system call suspends the execution of a parent process until child terminates or it receives a signal.
* It will lead the parent process to be in the ‘waiting state’.
* When the child terminates, the kernel sends ‘SIGCHLD’ signal to the process.
* after termission of child process, child reports its status to parent and then parent process asks kernel to kill that child process entirely.
* meanwhile, we can say that the child will remain in ‘zombie state’ for a very short duration of time. i.e. only between exit() & wait() system call.

*Following figure will help you clear your idea…*



Akshat A. Mistry | Parent process waits for the child to be executed

***Before moving forward with the second case, we need to learn about an ‘orphan process’…***

*Orphan Process…*

* “ A process whose parent process no more exists i.e. either finished or terminated without waiting for the child process to terminate is known as ‘Orphan Process’. ”
* After termination of parent process, no one is there to look after the child process leading it to remain itself in the ‘Zombie State’.

*What will happen with the ‘orphan processes’ ??*

* ‘Re-parenting’ is a process of detecting the orphan process (by kernel) and trying to provide it a new parent. In most of the cases, INIT process becomes the new parent.
* this causes, new parent to wait for the child termination & then asking the kernel to clean-up the PCB entry in the process table.

*Why too many Orphan processes are harmful to the system ???*

Very simple answer to this question is,

* These processes are not needed any more, that’s why parent rejected them.
* These processes are not doing any useful work, they are claiming the resources but not producing the effective output.
* they have occupied the resources, they are utilizing the resources still they are none of use. so those occupied resources must be freed

Now we are ready to understand case 2…

***case 2 : parent doesn’t wait for the child process , i.e. both execute parallely.***

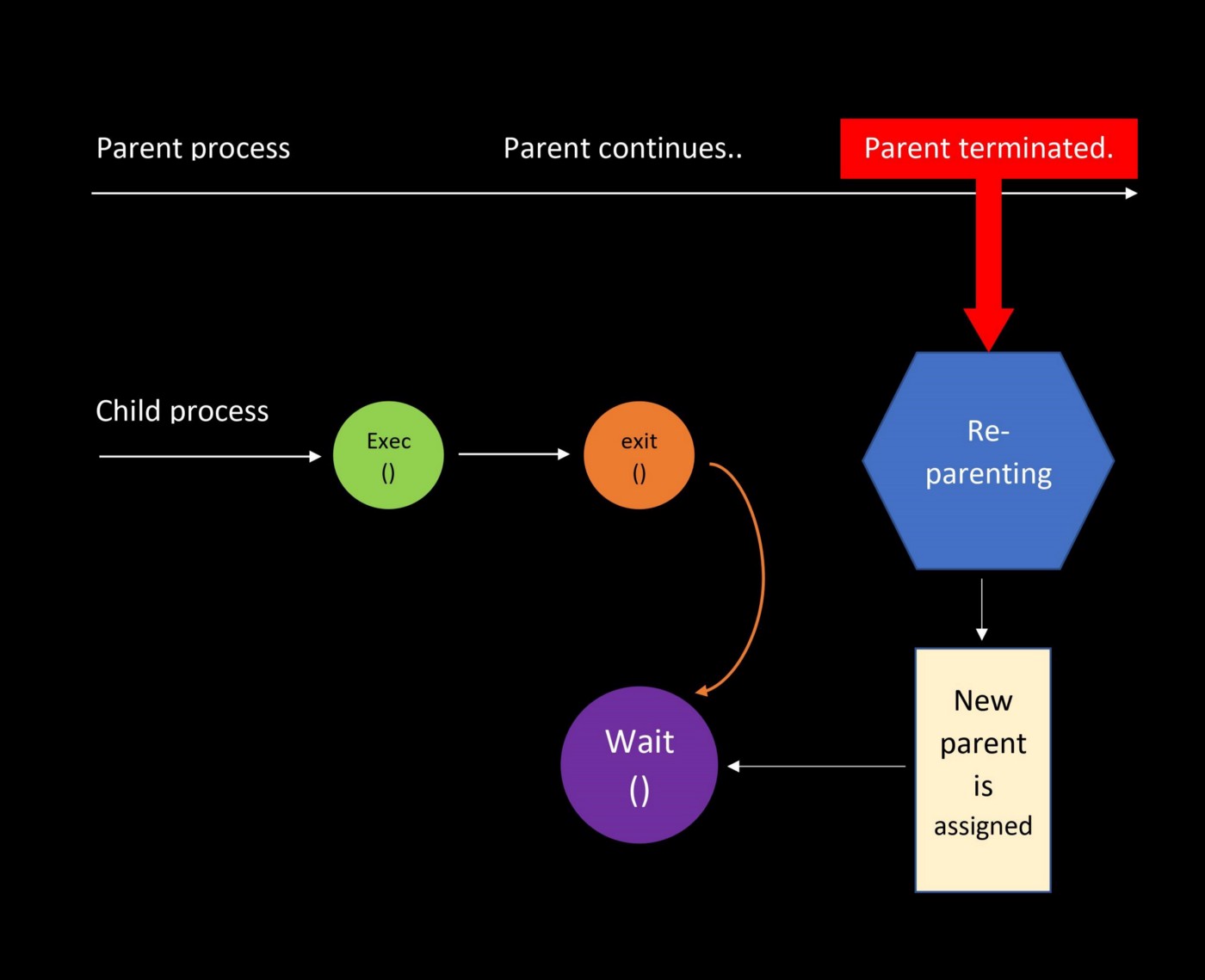
* Further this case can be sub-divided into to sub-cases.

*Sub-case 2.1 : parent process terminates after completion of it’s child process. But note that parent is not having any wait() system call…*

* In this case, what happens is that, once the child process is terminated, It enters in a zombie state but its parent is not caring about its child process. so the child entry remains in the process table making it a ‘zombie process’.
* After completion or termination of the parent process, the child has now become an orphan. And now we know that what happens with the orphan processes, right ??? are you able to recall it ? Yes, ofcourse, it will go under ‘re-parenting’ process where the new parent is allocated to this processes and it gets fully killed.
* Here you can note that, between the time period of child termination & reparenting process (which eventually calls the wait() system call for it), the child process is a ‘zombie’, wasting the precious memory space in the process table.
* refer the below diagram…

*Sub-case 2.2 : parent process terminates before completion of it’s child process. But note that parent is not having any wait() system call…*

* Understanding this case is quite simple than the above case.
* As soon as the parent terminates, the child becomes an orphan. But the child has not been terminated yet and its reparenting is done. So as soon the child terminates, new parent will call the wait() system call for it.
* It’s been similar to the very first case we have discussed, but the only difference is there the actual parent is calling wait() system call or its child & here the new parent will do the same.
* refer the below diagram…



Akshat A. Mistry | Parent process does NOT waits for the child

*Understanding ‘fork() Bomb’*

* It actually creates an infinite number of ‘Zombie processes’ without letting the parent wait() for its child execution. i.e. case 2.
* But the reparenting process needs some time to assign new parent process to it. (if it falls under subcase 2.2)
* At worst, the entire process table is acquired by these zombies, and there doesn’t exist any space for new process creation.
* The only option is to reboot the entire system to get rid of the ‘fork() Bomb’