Processes: Parents and Children

The normal way to create a new (child) process in Unix/Linux using PYTHON is using `fork()`

```
# creating a child process using fork
# import module os.
import os
id = os.fork()
# at this point, after execution of the fork,
# 2 processes are running, child and parent
if (id==0):
      # this is the child
      # do something here in the child process
   print('child')
else:
      # this is the parent
      # do something here in the parent process
   print('parent')
```

Processes: Parents and Children

The "normal" behaviour is for the parent process to wait (using `os.wait()`) for the termination of the child process, and retrieve its exit status. The other possibilities are

1. The parent process terminates before the child process (in which case the child becomes an *orphan*).
2. The child process terminates before the parent, but the parent does not `wait()` for the exit status of the child: In this case the child process becomes a *zombie*.

Processes: Orphans

```
1  # creating and demonstrating an orphan process
2  import os
3  import time
4  id = os.fork()
5  if (id==0):
6  # this is the child
7      parent = os.getppid()
8      for j in range(7):
9          time.sleep(1)
10         cppid=os.getppid()
11         print ("Child: My parent "\
12                "is process ID ", cppid)
13         if (parent != cppid):
14             print ("I just became an orphan")
15 else:
16 # this is the parent
17     time.sleep(3)
```

Processes: Orphans

Comments on the preceding code

- In line 4, the parent creates a child.
- In line 17 the parent goes to "sleep" for 3 seconds: Since there is no code after this, after the 3 seconds the parent process terminates.
- Lines 8 and 9 show that the child process will run for (at least) 7 seconds, so the child will outlive the parent. Because the parent process dies after (about) 3 seconds, and the child process lives for (about) 7 seconds, for (about) the last 4 seconds, the child is an **orphan**.
- `os.getppid()` returns the Parent Process ID of the current process.
- To find out when the child process becomes an orphan - we see if its current PPID has changed (in line 13): The new PPID will be different (...the orphan is adopted...)

Processes: Zombies

```
1  import os, sys, time
2  code = os.fork()
3  if code == 0:
4      for j in range(6):
5          time.sleep(1)
6          print ('child still alive, process ID: ', os.getpid()
       )
7      # after 6 seconds, terminate the child
8      sys.exit(1)
9  else:
10     # don't let the parent wait
11     for i in range(12):
12         print ('parent still alive, process ID ', os.getpid()
       )
13         time.sleep(1)
```

Processes: Zombies

Comments on the preceding code

- In line 2, the parent creates a child.
- As long as they are running, each second both child and parent processes print out their status (and process ID).
- The child is terminated after (about) 6 seconds (in line 8).
- The parent runs for at least 12 seconds, and does not `wait()` for the child.
- Therefore, for the second half of the program (approx.), the last 6 seconds, the child is a **zombie** process.

Process Hierarchies

```
1  import os, sys, time
2  def myloop(a,b):
3  # The process which we NAME b sleeps for a seconds
4      for k in range(a):
5          time.sleep(1)
6          print (b, ' still alive, process ID: ', os.getpid())
7  # Now lets create the child process:
8  code = os.fork()
9  if code == 0:
10 # Now lets create a "grandchild" process:
11     ncode = os.fork()
12     if ncode == 0:
13         myloop(5,'grandchild')
14     else:
15         myloop(10, 'child')
16 else:
17     myloop(15,'parent')
```

Process Hierarchies

Comments on the preceding code

- In lines 13, 15, 17, we tell the grandchild/child/parent to live for 5/10/15 seconds respectively.
- Note that there is no `os.wait()` command anywhere in the code.
- When the "grandchild" dies after 5 seconds (line 13), it will become a zombie.
- When the child dies after 10 seconds (line 15), it will become a zombie. **What happens to the grandchild zombie?**
- When the parent dies after 15 seconds, the zombie child gets adopted by `init()`. `init()` regularly executes `os.wait()` so that the zombie (entry in the process table) goes away.

If one was to swap the 10 and 5 in lines 15 and 13, so that the child terminates before the "grandchild", the child would become a zombie while the grandchild would become an orphan.

A different way to "kill" a Zombie

```
1  import os, sys, time
2  def myloop(a,b):
3  # The process which we NAME b sleeps for a seconds
4      for k in range(a):
5          time.sleep(1)
6          print b, ' still alive, process ID: ', os.getpid()
7  # Now lets create the child process:
8  code = os.fork()
9  if code == 0:
10     myloop(10,'child')
11 else:
12     myloop(20,'parent')
13     os.wait()
14     myloop(10,'parent')
```

The child process executes for 10 seconds: Then, since the parent is not waiting for the execution status of the child, the child becomes a zombie. But after 20 seconds, the parent does execute a `wait()` which gets rid of the zombie, leaving the parent to continue for a further 10 seconds.

Some useful OS commands for processes.....

- **ps**
  `ps aux` List all processes
  `ps aux | grep defunct` List all zombie processes (also called "defunct" processes). The vertical separator here is called a pipe: It links two different commands, sending the output from the "ps aux" command as input to the "grep defunct" command.
  `ps aux | grep Z` Since the status of Zombies is often returned as Z, we can grep (search for) Z in the output listing.
- **pstree** This command (type in `man pstree` for more details) draws to the screen in ASCII characters a tree to represent all processes on your system, and their place in the hierarchy (with `init()` at the root of the tree).
  `pstree -p` Include process IDs for all processes.
- **kill** Explicitly kill a certain process.
  `kill -9 PID` Force the termination (-9) of process with ID `PID`.