Operating Systems

Lab 4 Exercises – Processes, `fork()`, `wait()`, *and* `waitpid()` *system calls*

Learning goals: this laboratory activity is useful to understand how to create processes and how to synchronize parent and children processes.

Exercise 1

Make a program **`parent-child`** that forks a single child process, then it generates and **prints** a random number **`n1`**, in range **`[0-10]`**, sleeps for **`n1`** seconds, waits for its child to exit, retrieves the child's exit code, and **prints** it.

The child **prints** his PID, the parent PID, and sleeps for **`n2`** seconds, where **`n2`** is a random number in range **`[0-10]`**, then it generates a random number **`r`** in range **`[0-255]`**, and use it as its return code, then **prints `r`** and its parent PID, and exits.

Run your program several times, and compare the parent PID printed by the child before sleeping, and the one printed after sleeping. Are they always the same? If not why?

Exercise 2

Modify the range of **`n1`** and **`n2`** in **`parent-child`** to **`[0-100]`**. Remake.
Run **`3`** instances of the program **`parent-child`** in background using the command:
`parent-child & parent-child & parent-child &`
From a different shell, use **`ps`** to figure out the PID, and the status, of all the parents and children processes.
Terminate **`4`** of those processes using the command **`kill -9 PID`** , use **`ps`** again to figure out the PID, and the status of all remaining processes.

Exercise 3

Write a concurrent C program that reads from command line an argument **`n`**, dynamically allocate an array **`v`** of size **`n`**, and read n values from **`stdin`** and store the values in the array.
The main process creates **`n`** processes.
The first created child sleeps **`n`** seconds, the second child sleeps **`n-1`** seconds, and so on so forth (last created child sleeps **`1`** second). Each child after sleeping terminates returning as its status code the number of seconds it has slept.

The parent must wait for its children sequentially **from the first created to the last one**, printing their return code and the value of **`v[i]`**, where i is the identifier of the process (0 to n-1).