Today we will finish (?) writing our own Unix Shell. First, we will examine the three approaches for system calls, then we will change our code to execute processes in background.

You will need the code from the last tutorial.

```
tar -zxvf 457shell-20120126.tar.gz
cd 457shell
```

1. In order to show the differences more easily, first, you should create the following function:

```
int fork_1() {
    int pid;
    pid = fork();
    return pid;
}
```

Note: Function name is “fork_1” (“fork” underscore number one).

2. Create functions “fork_2” and “fork_3” (based on “fork_1”) using the other two approaches, i.e., indirect call (“syscall” function) and inline assembly.

3. (Optional) Try to execute our unix shell using this 3 approaches, in order to verify if your implementation works just as the usual “fork”, substituting “fork” for your function “fork_n” in “execute_external”.

4. Modify your “makefile” to generate the assembly code (-S) for your program. For example: “make asm”.

```
Note: For “good practices”, “make clean” should also remove the file that results from “make asm”.
```

5. Execute “make asm” and compare the three versions of fork_x. Is “syscall” independent of GNU C library?
```
(Hint: Look for “labels”, i.e., function name followed by a colon. E.g, “fork_n:”)
```

6. To execute a program in background, a user should finish the command with “&”. For example, running “ls &”.
```
So, change the function execute_external to the following:
```
function execute_external:
    background = false
    if last_argument == "&":
        background = true
        remove last argument
    child = fork
    if child < 0, print error
    if child == 0, execute child program
    if child > 0:
        if not in background: waitpid(child, status, 0)
        else: print "Executing %s [%d]", process_name, child_pid

7. Modify the “signal handler” to deal with SIGCHLD as well. If you receive a SIGCHLD, you should use “waitpid” and tell which process has ended.

   Note: A more complete implementation would tell why it ended.

8. Can your program detect when a child is finished? What is missing?
   Fix it, compile it and test it!

   NOTE: The “final” version will be available in our wiki.

Codes from Slides (for reference)

    int child = wait(&status);
    int child = waitpid(pid, &status, options);
    int child = waitpid(-1, &status, WNOHANG);

Where:
    - options=0 forces it to wait and options=WNOHANG returns if no signal is found.
    - pid=-1, waits for the next child

    pid = getpid();

    pid = syscall(SYS_getpid);

    asm volatile ("movl %%eax, %0 \n	"
                  "int $0x80 \n	"
                  "=a"(pid):
                  "a"(SYS_getpid):);