Section 5.2
Process Management

1. Overview
2. Managing processes
3. System calls

5.2.1 Overview

- What is a *process*?
  - it’s a running executable

- What is *process management*?
  - it’s controlling the execution of a process
  - examples of process management:
    - *spawning* a new process (launching its execution, starting it up)
    - *killing* a process (stopping its execution)
    - pausing a process
    - modifying the behaviour of a process by sending it a *signal*

Overview (cont.)

- Processes can be managed:
  - by a user
    - using shell commands
  - by other programs or processes
    - using system calls

Overview (cont.)

- Each process has:
  - a unique process identifier (PID)
  - a parent process
    - this is the process that spawned it
    - the parent process id is PPID
  - its own address space and virtual memory
    - it has its own code segment, data segment, function call stack, heap
  - its own control flow(s)
    - it may be multi-threaded

5.2.2 Managing Processes

- From a shell, a user can:
  - start a process
    - in the foreground
    - in the background
  - send a signal to a process
    - to suspend the process
    - to stop the process
    - … and other stuff, more on this later …
- **`coding example <p1>`**

Managing Processes (cont.)

- A process can start a new process:
  - by cloning itself
    - using the **`fork()`** system call
  - by morphing itself
    - using the **`exec()`** family of system calls

5.2.3 System Calls

- Some system calls related to process management
  - **`fork()`**
  - **`exec()`**
  - **`wait()`**
  - **`system()`**

Forking a Clone Process

`pid_t fork(void)`

- Description
  - this system call creates a **clone** of the current process
    - the current process is the *parent*
    - the new process is the *child*
    - the child process gets a copy of the parent's address space
  - the return value of the **`fork()`** system call
    - in child process:
      - zero
    - in parent process:
      - child process id, if successful
      - -1 in case of error
- **`coding example <p2>`**

Forking a Clone Process (cont.)

- Multiple child processes can be spawned
  - each child process gets a copy of the parent’s code
  - multiple **`fork`**s in the parent mean multiple **`fork`**s in the children
- Watch for **`fork`** bombs ☠
  - the OS keeps process table
  - all tables have finite capacity
  - **`coding example <p3>`**

Morphing Into Another Process

- Use the **`exec()`** family of system calls
  - this *replaces* the current process code with a *different* program
    - it has the same PID
    - it has different instructions
  - examples: **`execl()`**, **`execlp()`**, **`execle()`**, **`execv()`**, **`execvp()`**
  - differences are in the parameters and the environment variables
  - if **`exec()`** call fails, the original program continues
  - **`coding example <p4>`**

Waiting for a Child Process

```
pid_t wait(int *status)
```

- Description
    - this pauses the execution of the parent until any child process terminates
    - return value:
        - child PID, if successful
        - -1 in case of error

Waiting for a Child Process (cont.)

`pid_t waitpid(pid_t` ***`pid`*****`, int *`*****`status`*****`, int`** ***`options`*****`)`**

- Description
  - pauses execution of parent until specified child process terminates
  - return value:
    - child PID, if successful
    - -1 in case of error
- **`coding example <p5>`**

Invoking a Shell Command

```
int system(const char *command)
```

- Description
  - this spawns a child shell, and runs the specified ***command***
  - the process blocks until the ***command*** execution has completed
  - return value:
    - shell termination status, if successful
    - -1 in case of error
- **coding example <p6>**