Mini-Lab 3: Process

COMP3230, The University of Hong Kong

Sept. 2023

Total 1 point

Objective

At the end of this mini-lab, you will be able to:

- Gain hands-on experience in process-related system calls, e.g. `fork()`, `wait()`, and `exec()`.
- Understand the parental relationship between processes.

Instructions

In this mini-lab, you will implement a simple multiprocessing program and `fork()` and `exec()` to understand how the operating system manages multiple processes.

1. Open two terminals marked as Terminal 1 and Terminal 2.
2. In Terminal 1, find out and record *process id* of the current shell using `echo $$ > t1-pid.txt`.
3. Complete all TODO sections in `lab3-process.c` using `fork()`, `exec()`, and `wait()` following the inline instructions.
4. In Terminal 1, compile and execute `lab3-process.c`. If your implementation is correct, the child process will sleep 20 seconds.
5. Before the process returns in Terminal 1, in Terminal 2, execute `pstree -sp $(cat t1-pid.txt)` to monitor the tree of running processes. We can see a chain of processes like SSH > Bash (Terminal 1) > parent process (lab3-process) > child process (sleep 20).

   Explanation of their relations: Upon connecting remotely, the SSH process forks to create a new Bash shell for the user's session. When lab3-process is executed within this Bash shell, Bash forks again to run the program, which `lab3-process` forks once more to create a child process for the sleep 20 command. This sequence of fork operations establishes the chain: SSH forks Bash > Bash forks Parent (lab3-process) > Parent forks Child (sleep 20).

Submission

(1 pt) Complete all TODO sections and submit your code as `lab3-process_<your_student_id>.c`.

Appendix

```c
// file: lab3-process.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

int main(int argc, char *argv[]) {
    pid_t pid = -1;
    printf("Before fork(). variable pid = %d, getpid()=%d, getppid()=%d\n",
        pid, getpid(), getppid());
    fflush(stdout);
    // TODO: Create a child process using fork() and store the return value
        in pid (~1 line)

    if (pid < 0) {
        fprintf(stderr, "fork() Failed");
        exit(-1);
    } else if (pid == 0) {
        // Child Process
        printf("Child Process. variable pid = %d, getpid()=%d, getppid()=%d
            \n", pid, getpid(), getppid());
        // TODO: Use exec() family to replace the current process image
            with "sleep", "20" (~1 line)

    } else {
        // TODO: Make the parent process wait for the child to complete (~1
            line)

        printf("Parent Process. variable pid = %d, getpid()=%d, getppid()=%
            d\n", pid, getpid(), getppid());
    }
    return 0;
} // main
```