Homework 2

Monday, April 15, 2019 10:16 AM

Due 4/18/19

Daniel Yan

In Unix, the first process is called init. All the others are descendants of “init”. The init process spawns a sshd process that detects a new secure ssh requested connection (WKPort 22). Upon a new connection, sshd spawns a login process that then loads a shell on it when a user successfully logs into the system. Now, assume that the user types

who | grep <uwnetid> | wc –l

Draw a process tree from init to those three commands. Add fork, exec, wait, and pipe system calls between any two processes affecting each other.

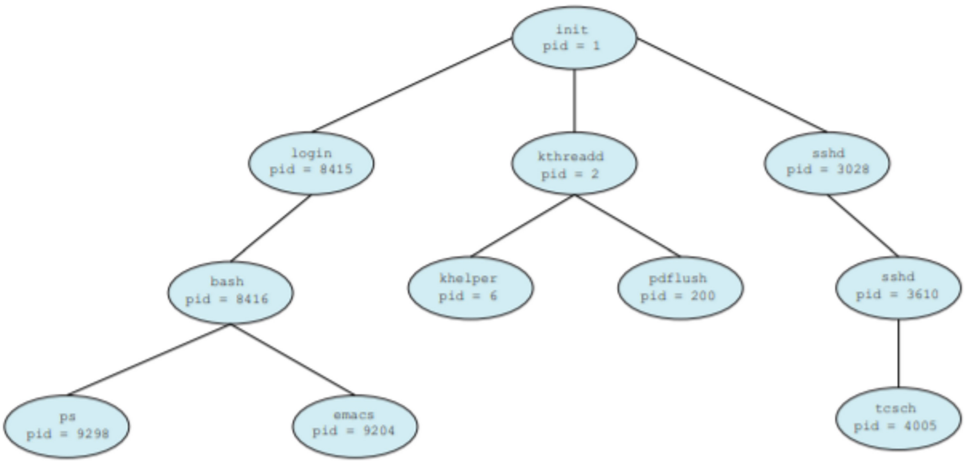
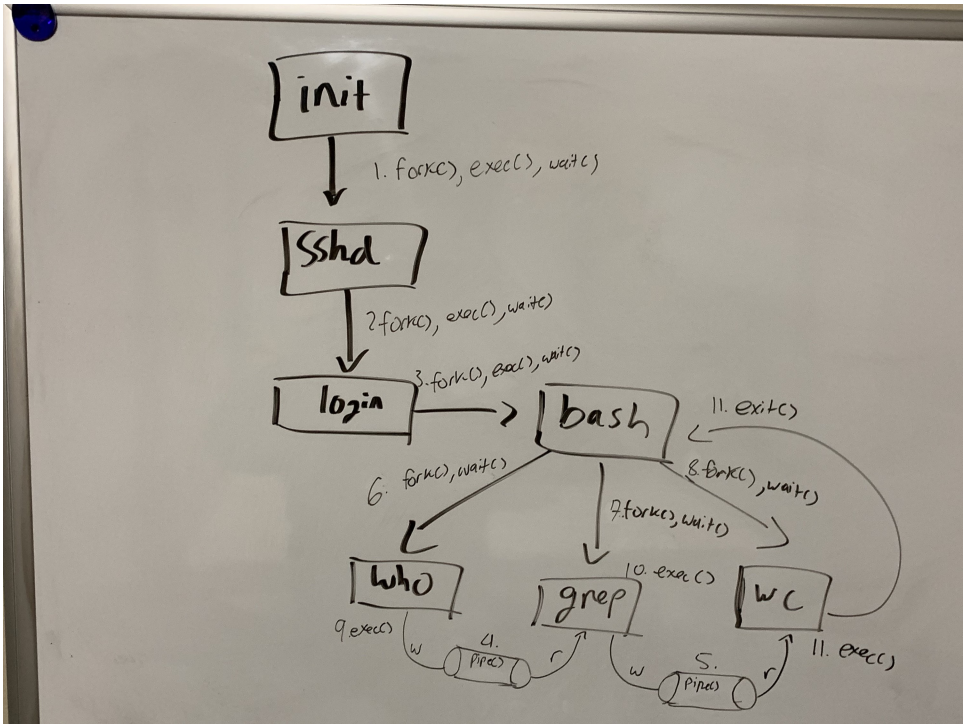


Figure 3.8 A tree of processes on a typical Linux system.

Example Process tree

Complete following problems from book (Ninth Edition): 3.2, 3.9, 3.11, 3.14

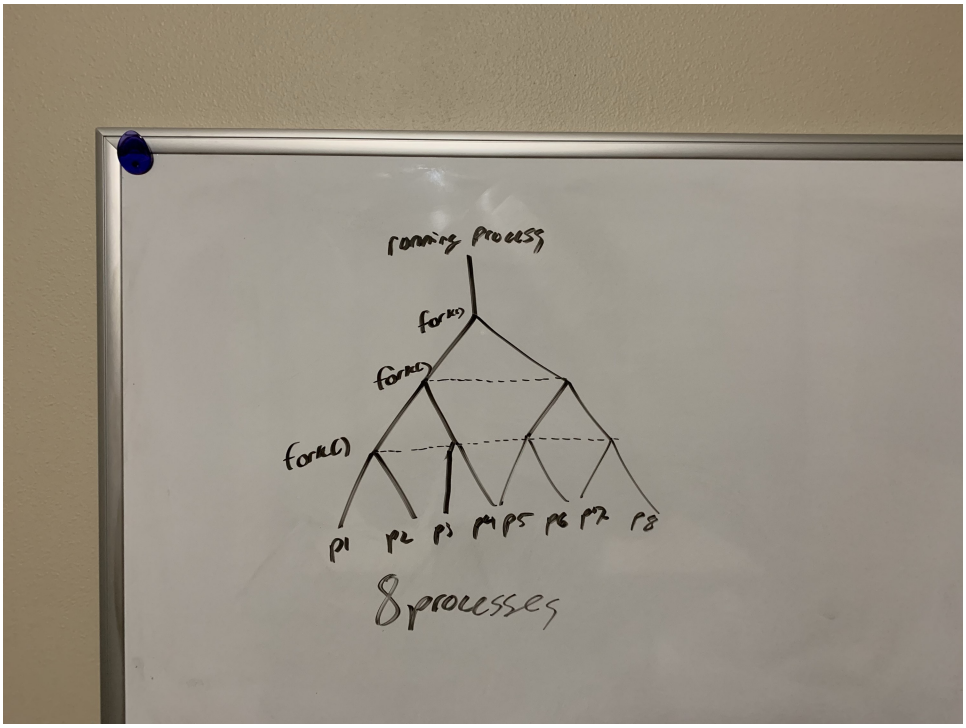
Process Tree



Exercise 3.2 Including the initial parent process, how many processes are created by the program shown in Figure 3.31?

```
#include <stdio.h>
#include <unistd.h>

int main()
{
    /* fork a child process */
    fork();

    /* fork another child process */
    fork();

    /* and fork another */
    fork();

    return 0;
}
```

- 8 Processes



Exercise 3.9 **Describe the actions taken by a kernel to context-switch between processes.**

- When switching the CPU from one process to another, you need to form a state save of the current process and a state restore of a different process. This state save and state restore formation task is the context switch. When the kernel then performs a context switch, it saves all the old process contexts inside the PCB and loads the saved context of the new process to run. To be more specific, when the kernel saves the content, it saves current register states and machines states in the PCB. Afterwards, the scheduler is invoked to determine the next process to run which is then loaded from the PCB, which is the restore state (restoring from registers).

Exercise 3.11 **Explain the role of the init process on UNIX and Linux systems in regard to process termination**

- Whenever a process is terminated, it is moved into the zombie state until the parent calls wait(). If the parent doesn't invoke wait(), the child process will remain a zombie until the parent is alive. When wait() is called, the pid and the entry in the process table is released for the child process. In the case where wait() for the parent process wasn't called, and the parent process terminates, the init process becomes the new parent. Every so often, the init process calls wait(), releasing the pid and entry in the process table for the child process.

Exercise 3.14 **Using the program in Figure 3.34, identify the values of pid at lines A, B, C, and D. (Assume that the actual pids of the parent and child are 2600 and 2603, respectively.)**

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid, pid1;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed");
       return 1;
    }
    else if (pid == 0) { /* child process */
       pid1 = getpid();
       printf("child: pid = %d",pid); /* A */
       printf("child: pid1 = %d",pid1); /* B */
    }
    else { /* parent process */
       pid1 = getpid();
       printf("parent: pid = %d",pid); /* C */
       printf("parent: pid1 = %d",pid1); /* D */
       wait(NULL);
    }

    return 0;
}
```

- Values of pid for Child A: 0
    - Given we don't actually use getpid() for pid here, so pid is just 0
- Values of pid1 for Child B: 2603
    - Given we ask for the pid of the child which is 2603 from getpid(), the pid is 2603
- Values of pid for Parent C: 2603
    - The pid is never set again, but is from the child process so it would be 2603
- Values of pid1 for Parent D: 2600
    - We set pid1 from getpid() in parent so it'd be 2600