Mantri Krishna Sri Ipsit
180070032
180070032@iitb.ac.in

Assignment 1
CS 347: Operating Systems

2022/01/16

Part A: Understanding Linux Processes

1. (a) My machine has **6** CPU cores. Each core has **2** threads. Hence the command `more /proc/cpuinfo` shows **12** different processors.
   (b) Frequency of each CPU is **2**.
   (c) My system has a total of **16335724 kB**. Out of this, **6286200 kB** is free. This is found using the `more /proc/meminfo` command.
   (d) The total number of forks since bootup is **24541**. And the total number of context switches since bootup is **87459116**. This is found using the `more /proc/stat` command.

2. (a) The PID of the process running the `cpu` command is **24831**.
   (b) This process is consuming **100%** of CPU and **0%** of memory.
   (c) The current state of the process is **R**. It is in **running** state.

3. (a) The PID of the process spawned by the shell to run the `cpu-print` executable is **27038**. This is found using the `ps -C cpu-print` command.
   (b) The PID of the parent of the `cpu-print` process is **11168**. This is found using the `ps -f -C cpu-print` command. The PIDs of all ancestors are:

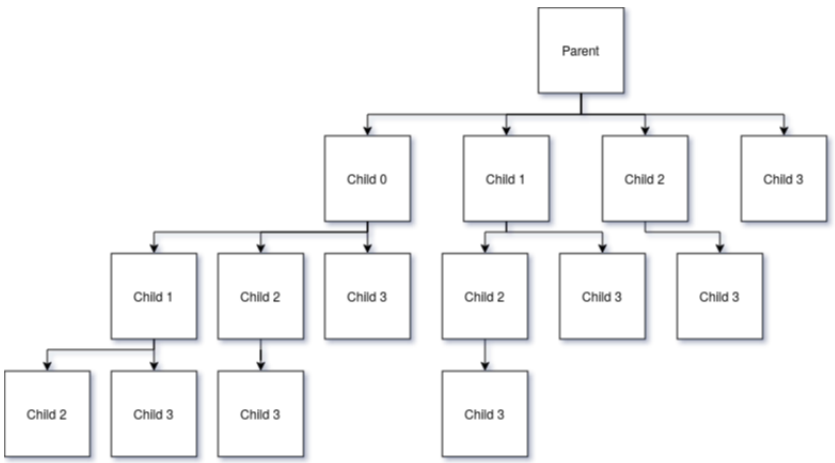
   ```
   systemd(1)---systemd(6069)---gnome-terminal-(11158)---bash(11168)---cpu-print(27038)
   ```

   (c) Using the `ls -l /proc/32077/fd` (here 32077 is the PID of the output redirection process) command, we get the following:

   ```
   total 0
   lrwx------ 1 ipsit ipsit 64 Jan 12 13:54 0 -> /dev/pts/1
   l-wx------ 1 ipsit ipsit 64 Jan 12 13:54 1 -> /tmp/tmp.txt
   lrwx------ 1 ipsit ipsit 64 Jan 12 13:54 2 -> /dev/pts/1
   ```

   Here we can see that the file descriptor for standard output is being pointed to `/tmp/tmp.txt` while the other descriptors are pointing towards a pseudo-terminal. Hence we can say that the I/O redirection happens in the following way: ***First, based on the redirection type (<, >, or 2 >), a file is opened (if it already exists then that will be used) with the given filename and then in the process, the file descriptor will point to the new file opened. In that way, whenever a system call is made to print to a screen or throw an error or take input, the data is sent to the file to which the file descriptor points to.***

4. (a) In the program, there is a loop that runs 4 times. Inside each loop, `fork()` is called. Hence the processes created can be explained using the following flow chart:



As we can see here, the output of `fork4.c` program should contain **1** line of child 0, **2** lines of child 1, **4** lines of child 2 and **8** lines of child 3. The order in which these lines are printed is based on the scheduling algorithm the OS is using. But theoritically, whenever a `fork()` is called, the parent and the created child execute concurrently.

(b) The program has been modified as follows:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

void main(int argc, void *argv) {
    for(int i=0; i<4; i++) {
        int ret = fork();
        if(ret == 0) {
            int pid = getpid();
            int ppid = getppid();
            printf("child: i = %d, PID = %d, PPID = %d\n", i, pid, ppid);
        }
        else {
            int res = wait(NULL);
            printf("reaped: PID = %d\n", res);
        }

  }
}
```

This is nothing but the parent waits for the child to finish its process. So as expected, rge children will be reaped in a depth-first order, i.e., the most recently created child will be reaped first. Here is the output of the modified program above:

```
child: i = 0, PID = 27581, PPID = 27580
child: i = 1, PID = 27582, PPID = 27581
child: i = 2, PID = 27583, PPID = 27582
child: i = 3, PID = 27584, PPID = 27583
reaped: PID = 27584
reaped: PID = 27583
child: i = 3, PID = 27585, PPID = 27582
reaped: PID = 27585
reaped: PID = 27582
child: i = 2, PID = 27586, PPID = 27581
child: i = 3, PID = 27587, PPID = 27586
reaped: PID = 27587
reaped: PID = 27586
child: i = 3, PID = 27588, PPID = 27581
reaped: PID = 27588
reaped: PID = 27581
child: i = 1, PID = 27589, PPID = 27580
child: i = 2, PID = 27590, PPID = 27589
child: i = 3, PID = 27591, PPID = 27590
reaped: PID = 27591
reaped: PID = 27590
child: i = 3, PID = 27592, PPID = 27589
reaped: PID = 27592
reaped: PID = 27589
child: i = 2, PID = 27593, PPID = 27580
child: i = 3, PID = 27594, PPID = 27593
reaped: PID = 27594
reaped: PID = 27593
child: i = 3, PID = 27595, PPID = 27580
reaped: PID = 27595
```

Part B: A Simple Shell