

Operating System 112 Fall

Homework 1 - Process Forking

Prof. 蔡文錦

TA 林浩君, 薛乃仁, 周彥昀, 許承壹

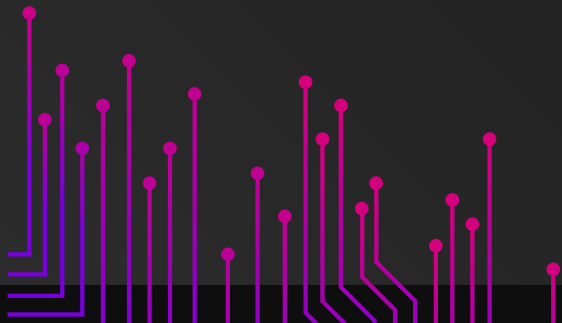
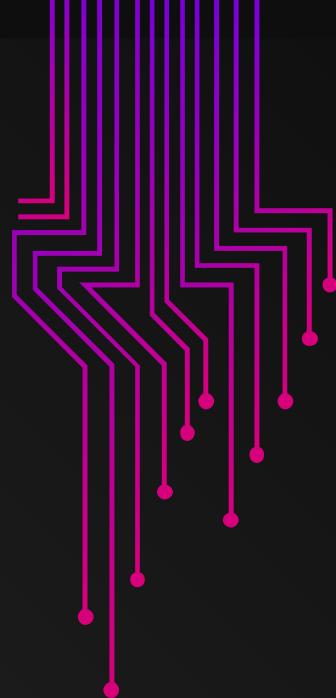


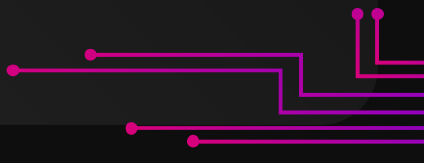
Table of Contents

1.	Objective
2.	Fork Example
3.	HW1-1(30%)
4.	HW1-2(30%)
5.	HW1-3(40% + 10% bonus)
6.	Given Files Overview
7.	Submission and Rule
8.	Reference



Objective

In this homework, we are going to learn about the concept of process, which is a running program. And also learn how to write some simple shell script.

1. Learn how to use system call to let a process communicate between user mode and kernel mode.
 2. Learn how to fork a process and understand the relationship between parent and child process.
 3. Learn how to build a subprocess to run linux commands.
 4. Learn how to write shell script to track the relationship between processes.
- 

Fork Example

For better understanding of how processes are forked, we provided “example.c” for you.

```
Main Process ID: 108204
```

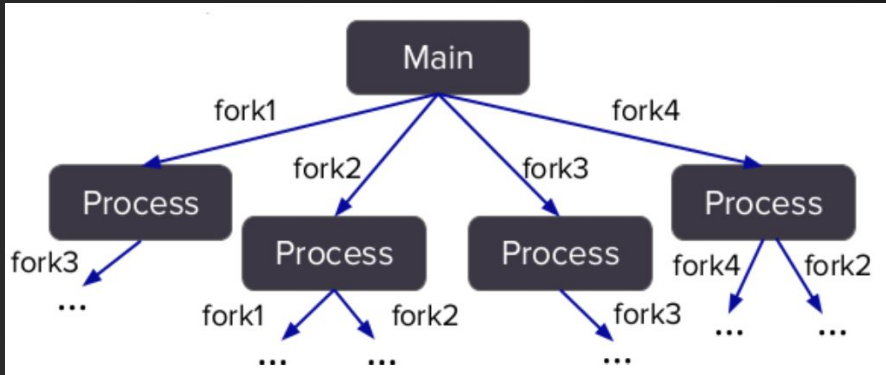
```
Fork 1. I'm the child 108205, my parent is 108204.  
Fork 3. I'm the child 108206, my parent is 108205.  
Fork 2. I'm the child 108207, my parent is 108204.  
Fork 3. I'm the child 108208, my parent is 108207.  
Fork 3. I'm the child 108209, my parent is 108204.
```

```
int main() {  
    printf("Main Process ID: %d\\n", getpid());  
  
    pid_t pid;  
    pid = fork(); // Fork 1  
    if (pid == 0) {  
        show(1);  
    } else if (pid > 0) {  
        wait(NULL);  
        pid = fork(); // Fork 2  
        if (pid == 0) {  
            show(2);  
        } else {  
            wait(NULL);  
        }  
    }  
  
    pid = fork(); // Fork 3  
    if (pid == 0) {  
        show(3);  
    } else {  
        wait(NULL);  
    }  
  
    return 0;  
}
```

Fork Example

It's highly recommended that you draw the fork tree for better recognition of forking process.

Example:



```
int main() {
    printf("Main Process ID: %d\n", getpid());

    pid_t pid;
    pid = fork(); // Fork 1
    if (pid == 0) {
        show(1);
    } else if (pid > 0) {
        wait(NULL);
        pid = fork(); // Fork 2
        if (pid == 0) {
            show(2);
        } else {
            wait(NULL);
        }
    }

    pid = fork(); // Fork 3
    if (pid == 0) {
        show(3);
    } else {
        wait(NULL);
    }

    return 0;
}
```

HW1-1 : Building a Graph of Processes (30%)

Demand:

Complete “hw1-1.c” which uses fork() to **produce the processes in the format of this screenshot**, you have to follow the output format strictly.

- Please note that **the order of fork()** of your output should be the same as provided.

```
Main Process ID: 27771
```

```
Fork 1. I'm the child 27772, my parent is 27771.  
Fork 2. I'm the child 27773, my parent is 27772.  
Fork 3. I'm the child 27774, my parent is 27772.  
Fork 4. I'm the child 27775, my parent is 27774.  
Fork 4. I'm the child 27776, my parent is 27772.  
Fork 2. I'm the child 27777, my parent is 27771.  
Fork 3. I'm the child 27778, my parent is 27771.  
Fork 4. I'm the child 27779, my parent is 27778.  
Fork 4. I'm the child 27780, my parent is 27771.
```

Hint for 1-1

Some important System Calls :

- `fork()`: Create a new child process.
- `wait()`: Wait for child process to terminate.
- `getpid()`: Return current process ID.
- `getppid()`: Return parent process ID.

HW1-2 : Simple CLI Construction (30%)

Demand:

Complete “hw1-2.c” that serves as a **shell interface** to accept user commands then execute each command in separate processes.

Your shell needs to support following commands:

- cat
- ls (-a -l)
- date
- ps (-f)
- pwd
- exit



HW1-2 : Simple CLI Construction (30%)

Annotations:

- List all the files
- List number of files and all hidden files and directories
- Another way to input parameters for ls
- Show the context inside hello.txt
- Show current running processes
- Show current path
- Exit the shell and output "Process end"
- List all the hidden files and directories
- Show the current time

```
lin > $ ./hw1-2
osh> ls
hello.txt hw1-2
osh> ls -a
. . . hello.txt hw1-2
osh> ls -a -l
total 24
drwxrwxr-x 2 lin lin 4096  9  20 22:55 .
drwxrwxr-x 3 lin lin 4096  9  20 22:55 ..
-rw-rw-r-- 1 lin lin   14  9  20 22:55 hello.txt
-rwxrwxr-x 1 lin lin 8744  9  20 22:54 hw1-2
osh> ls -al
total 24
drwxrwxr-x 2 lin lin 4096  9  20 22:55 .
drwxrwxr-x 3 lin lin 4096  9  20 22:55 ..
-rw-rw-r-- 1 lin lin   14  9  20 22:55 hello.txt
-rwxrwxr-x 1 lin lin 8744  9  20 22:54 hw1-2
osh> cat hello.txt
Hello World!!
osh> date
公曆 20廿三年 九月 廿日 週三 廿二時56分六秒
osh> ps
  PID TTY          TIME CMD
 23514 pts/3        00:00:02 zsh
 26448 pts/3        00:00:00 hw1-2
 26455 pts/3        00:00:00 ps
osh> pwd
/home/lin/Desktop/Operating-System-2023-Fall/HW1/hw1_answer/testing
osh> exit
Process end
lin >
```

Hint for 1-2

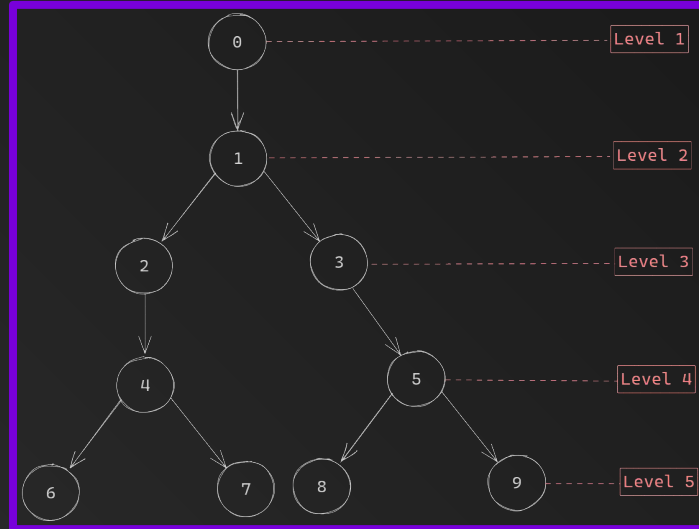
1. You need to print “**osh>**” and **a space** each time before reading command, please refer to the screenshot on the previous slide.
2. You can **only** use system calls from `execl()` / `execle()` / `execvp()` to execute the given command.
3. Create a child process first, since the mentioned calls will overwrite the current process.
4. Special handling might be required for an “exit” command.

HW1-3 :Process Traversal (40%)

Demand:

In this task, we will give you a pair of pids, and you need to answer whether the former process is the latter process' ancestor, and output “Yes” or “No”. Written in **shell script**.

- You have to implement the file “hw1-3.sh” to fulfill the mentioned function above.



HW1-3 :Process Traversal (40%)

Here is the example of what “hw1_3.sh” should do:

```
lin@lin-viplab-pc:~/Desktop/Operating-System-2023-Fall/HW1$ ./hw1-3.sh --parent 1064 --child 1890
Yes
lin@lin-viplab-pc:~/Desktop/Operating-System-2023-Fall/HW1$ ./hw1-3.sh --parent 1890 --child 1064
No
```

We also provide “test_hw1-3.c” and “test_hw1-3.sh” for checking program correctness.

```
lin@lin-viplab-pc:~/Desktop/Operating-System-2023-Fall/HW1/to_student$ ./test_hw1-3.sh
```

Pass Screen

```
Testcase 28 PASS
Testcase 29 PASS
Testcase 30 PASS
```

```
#####
#
#   You pass all testcases
#
#####
```

Fail Screen

```
Testcase 28 FAIL
command: ./hw1-3.sh --parent 18581 --child 18585
response: parent pid: 18581, child pid: 18585
```

```
Testcase 29 FAIL
command: ./hw1-3.sh --parent 18582 --child 18585
response: parent pid: 18582, child pid: 18585
```

```
Testcase 30 FAIL
command: ./hw1-3.sh --parent 18584 --child 18585
response: parent pid: 18584, child pid: 18585
```

```
You pass 0/30 testcases
```

Hint for 1-3

1. There's a template that help you read the input.
2. Study commands `ps`, `pstree` for getting process information.
3. Don't think too hard, it just takes about ten lines of codes to finish.
4. You can use any shell (`sh` / `bash` / `zsh` / etc...), just rewrite the first line.

```
hw1-3.sh x
to_student > hw1-3.sh
1  #!/bin/bash
2
3  # Read parent pid, child pid
4  while [ "$#" -gt 0 ]; do
5      case "$1" in
6          --parent)
7              parent="$2"
8              shift 2
9              ;;
10         --child)
11             child="$2"
12             shift 2
13             ;;
14         *)
15             echo "Unknown parameter passed: $1"
16             exit 1
17             ;;
18     esac
19 done
```

```
$ ./hw1-3.sh --parent 14215 --child 69177
parent pid: 14215, child pid: 69177
```

Template output result

HW1-3 :Process Traversal - Bonus (10%)

Bonus Demand:

- Design an additional parameter “--path” that decide whether to print out the path.
 - If two process is ancestor relationship, print out all pids along the path.
 - If two process is not ancestor relationship, just print out “No”.
 - If no given parameter “--path”, you should not print out the path.

Example:

```
lin@lin-viplab-pc:~/Desktop/Operating-System-2023-Fall/HW1$ ./hw1-3.sh --parent 1064 --child 1890 --path
Yes
1064 -> 1684 -> 1791 -> 1811 -> 1890
```

```
lin@lin-viplab-pc:~/Desktop/Operating-System-2023-Fall/HW1$ ./hw1-3.sh --parent 1890 --child 1064 --path
No
```

Given Files Overview

C example.c

-> for you to understand fork() and wait()

>_ hw1-3.sh

-> given template from T.A.

C test_hw1-3.c

-> for building the process graph that you need to track

>_ test_hw1-3.sh

-> for testing your implemented function in hw1-3.sh

Submission and Rule

Submission:

Please upload your homework in such format:

- HW1_studentID.zip (e.g. HW1_312551014.zip)
 - hw1-1.c
 - hw1-2.c
 - hw1-3.sh

Rule:

- No **plagiarism** is allowed, since the grade of this course is critical for **graduate program application in CS related field**, we will not pardon such behavior at all, so please be responsible to yourself. You can discuss with your classmates, but don't copy and paste.
- Incorrect filename / file format will get -10% point.
- Delayed submission will get -20% point per day.

```
lin > $ zip -r HW1_312552014.zip hw1-1.c hw1-2.c hw1-3.sh
updating: hw1-1.c (deflated 63%)
updating: hw1-2.c (deflated 56%)
updating: hw1-3.sh (deflated 54%)
lin > $ unzip -l HW1_312552014.zip
Archive: HW1_312552014.zip
  Length      Date    Time    Name
-----
  762  2023-09-20  16:36  hw1-1.c
  832  2023-09-20  22:53  hw1-2.c
  886  2023-09-20  19:34  hw1-3.sh
-----
 2480                      3 files
```


Reference

- [System Call Introduction](#)
- [Shell Script Tutorial](#)
- [Happy coding and be patient](#)

