Week 06
Deadline : Saturday, April 8th 2017

Fork

1. In this tutorial, we will learn fork process with some C programs
2. Login to your badak account first
3. Change your directory to “work” and create new directory inside “work” named “work06”
   ```
   $ cd work
   $ mkdir work06
   ```
4. Move all the required files posted on scele to your work06 directory.
   The files you need to move:
   - 04-fork.c
   - 05-fork.c
   - 06-fork.c
   - etc.
5. Change your directory to “work06” directory
6. Make a file named lab06.txt
   ```
   $ vi lab06.txt
   ```
   or
   ```
   $ nano lab06.txt
   ```
7. Write your github username on the first row of lab06.txt file, for example:
   ```
   # Github Account: myusername
   ```
   Don’t forget to save the file and exit the text editor
8. To begin, compile the C scripts first using make
   ```
   $ make
   ```
   NOTE: There will be possibly some warnings when compiling, but you can ignore them.

Learning Fork

1. Run the 04-fork program
   ```
   $ ./04-fork
   ```
2. To understand what fork does, you can look at its manual in Linux
   ```
   $ man fork
   ```
3. Learn the code from 04-fork.c script and the output of 04-fork
4. Do the same thing as you did for 04-fork to 05-fork and 06-fork
   ```
   $ ./05-fork
   $ ./06-fork
   ```
5. Learn the code and the output of both programs
6. Still don’t understand about fork? Here is some short explanation about fork.
   So basically, fork() function is a way to split down a process into 2 processes. Whenever a program calls the fork(), it will do 2 processes that runs on the same code, but with different process ID and perhaps also different output and behaviour for each process.
7. Now, answer the questions below and write the answers on lab06.txt at point number 1:
   - What are PID and PPID?
   - What do the getpid(), getppid(), sleep, and wait functions do?
   - What are the return values of fork() ?

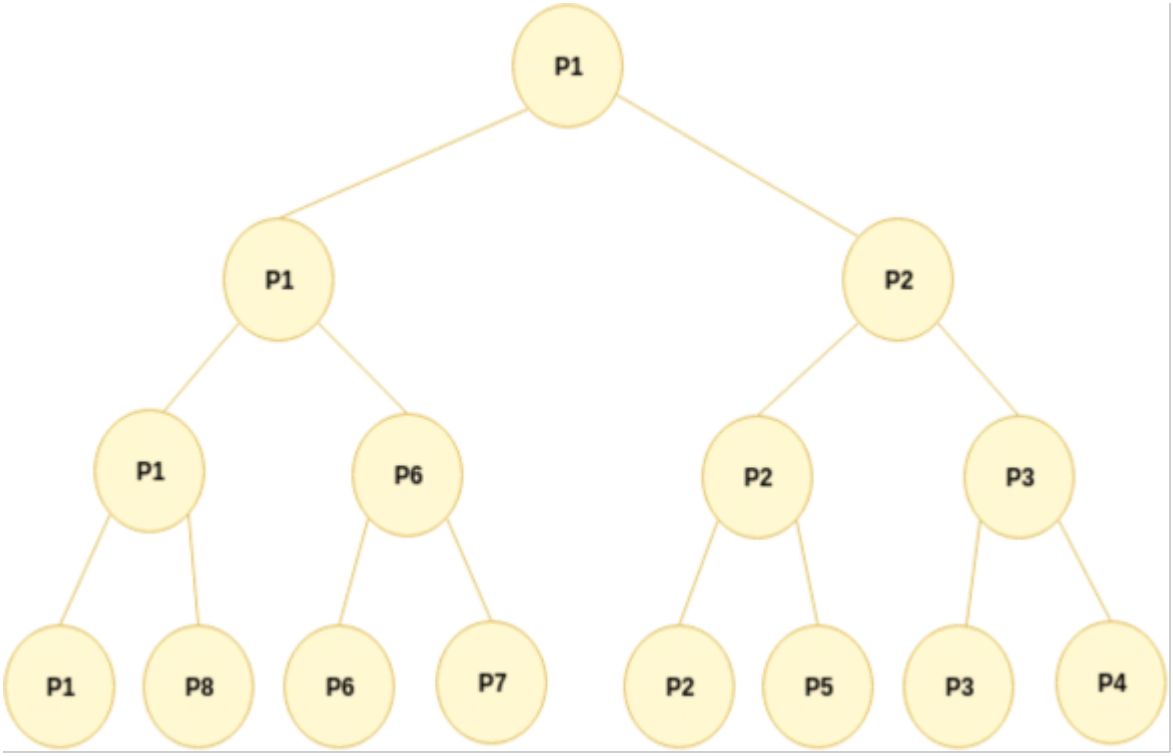
Fork Code

1. Run the 10-fork program

```
$ ./10-fork
```

2. Learn the code from 10-fork.c script and the output of 10-fork
3. Now, edit the script in your directory named 11-fork.c. Your goal is to have a program that has output structured like the fork tree below.

   Fork tree structure:



   ¬ʌ
   Output example:

```
L0: PID[1042] (PPID[965])
L1: PID[1043] (PPID[1042])
L2: PID[1044] (PPID[1043])
L3: PID[1045] (PPID[1044])
L2: PID[1044] (PPID[1043])
L1: PID[1043] (PPID[1042])
L2: PID[1046] (PPID[1043])
L1: PID[1043] (PPID[1042])
L0: PID[1042] (PPID[965])
L1: PID[1047] (PPID[1042])
L2: PID[1048] (PPID[1047])
L1: PID[1047] (PPID[1042])
L0: PID[1042] (PPID[965])
L1: PID[1049] (PPID[1042])
L0: PID[1042] (PPID[965])
```

   NOTE:
   - ¬ʌ the PID and PPID numbers can be different from example depends on the number of processes run on your OS, but the sequence of the output must be the same
   - ¬ʌ Be sure to compile the script first after editing using `$ make` command
   - ¬ʌ In the fork tree, the P1 stands for PID[1042] from the output example
   - ¬ʌ Just change the codes in main function
4. If you have finished, don't forget to compile the program and put the output to result.txt

```
$ make
$ ./11-fork > result.txt
```

5. Then, answer these questions below and write them on lab06.txt at point number 2:

- What does procStatus function do?
- What is the usage of wait(NULL) in function levelFork (not in general) ?
- What is the usage of fflush(NULL) in function procStatus (not in general) ?

Privacy Matters, Encryption, and Digital Signature using GnuPG

1. Hash and sign your works using sha1sum
   ```
   $ sha1sum * > SHA1SUM
   $ sha1sum -c SHA1SUM
   $ gpg --sign --armor --detach SHA1SUM
   ```
2. Verify your works
   ```
   $ gpg --verify SHA1SUM.asc
   ```
3. Change your directory to “work”, and create a tar ball. Tar is a way to create an archive file (like zip). You can google it for more information
   ```
   $ cd ..
   $ tar cvfj work06.tbj work06/
   ```
4. Encryption the tar file
   ```
   $ gpg --output work06.tbj.gpg --encrypt --recipient OSTEAM
   work06.tbj
   ```
5. Copy the file to your os171 directory in folder week06/
   ```
   $ cp work06.tbj.gpg ~/os171/week06/work06.tbj.gpg
   ```
6. Change your directory to os171/week06/
7. Remove “dummy” file
8. Check also whether your copy of “work06.tbj.gpg” exists or not. If you don’t find it, copy it once again
9. Push your work to GitHub
10. Week06 is done.

Review Your Work

After this week task completed, please don’t forget to check your files/folders. The structure of files/folders should be like:

os171
    key

mypublickey1.txt
log
<log_file>
…
SandBox
<some_random_name>
…
week00
report.txt
week01
lab01.txt
report.txt
myExpectation.txt
what-time-script.sh
week02
work02.tbj.gpg
*work02
*00-toc.txt
*01-public-osteam.txt
*02-ls-al.txt
*03-list-keys1.txt
*04-list-keys2.txt
*hello.c
*hello
*status.c
*status
*loop.c
*loop
*exercise.c
*exercise
*SHA1SUM
*SHA1SUM.asc
week03
work03.tbj.gpg
*work03
*01-public-osteam.txt
*.profile
*sudo-explanation.txt
*what-is-boot.txt
*SHA1SUM
*SHA1SUM.asc

week04
work04.tbj.gpg
*work04
*lab04.txt
*global-char.c
*global-char
*local-char.c
*local-char
*open-close.c
*open-close
*write.c

- *write
- *result1.txt
- *result2.txt
- *demo-file1.txt
- *demo-file2.txt
- *demo-file3.txt
- *demo-file5.txt
- *00-pointer-basic.c
- *00-step-1
- *00-step-2
- *00-step-3
- *00-step-4
- *SHA1SUM
- *SHA1SUM.asc

week05

- work05.tbj.gpg
  - *work05
    - *06-memory
    - *06-memory.c
    - *06-memory.map
    - *07-result.txt
    - *08-comments.txt
    - *Makefile
    - *SHA1SUM
    - *SHA1SUM.asc
    - *toprc

week06

- work06.tbj.gpg
  - *work06
    - *04-fork.c
    - *04-fork
    - *05-fork.c
    - *05-fork
    - *06-fork.c
    - *06-fork
    - *10-fork.c
    - *10-fork
    - *11-fork.c
    - *11-fork
    - *Makefile
    - *result.txt
    - *lab06.txt
    - *SHA1SUM
    - *SHA1SUM.asc

week07

- dummy

week08

- dummy

week09

- dummy

week10

- dummy

xtra

- dummy

Keep in mind for every files/folders with wrong name, you will get penalty point.
Note:File marked with "" mean that file should be inside the archived file.*