ENSEA
Ecole Nationale Supérieure de l'Electronique et de ses Applications

FAME

Lab 1 : Fork & co...

C. BARÈS

The manipulations proposed in this practical work are to be performed under GNU/-Linux. Always refer to the manual pages of the used functions, section 2 or 3. To view these pages you can :

- search for them on the Internet. Example on `http://man7.org/linux/man-pages/`
- install them on your VM : `sudo apt install manpages-dev`, then by typing into a terminal : `man 2 xxx` for the 2nd section of the `xxx`.. man page.

The sections that will interest us in system programming are :

2. System calls
3. Functions of the standard C library

1 – Let's eat !

1.1 PCB

The following program is considered :

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

void print_PCB()
{
    // TODO: complete this function
}

int main()
{
    pid_t ret = fork();

    printf("fork() returned: %d\n", ret);
    print_PCB();
    exit(EXIT_SUCCESS);
}
```

Complete the function `print_PCB`, so that it displays for the running process the following information : PID, PPID, UID, GID ; an example of an output for a single process could be :

```
PCB | PPID: 25934
    | PID:  26451
```

```
    | UID:   1000
    | GID:   1000
```

Compile and run this program once completed. Was the result predictable ? Are all the values displayed by `print_PCB` consistent ?

1.2 Father and son

Modify the `main` function of the previous question so that the father and son display a different sentence instead of "fork() returned :".

1.3 fork ! fork ! fork !

Can you predict how many processes will be created if you run the following program ?

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

int main(int argc, char** argv)
{
    fork();fork();fork();
    printf("fork!\n");
    exit(EXIT_SUCCESS);
}
```

Draw a tree of the different processes created at each `fork()`.

2 – Rabbits ?

2.1 Proper use of the fork

Test the following program. How is your system responding to this program ? You can interrupt it with `<ctrl>+c`.

```c
#include <unistd.h>

int main()
{
    while(1) fork();
}
```

So what does this program do ? (draw a graph)

2.2 Test its own limits

Write a program that generates an infinite number of processes (only children, no grandchildren) ; use the function `sleep()` (`man 3 sleep`) so that the children don't die right away and test the return value of the `fork()` to determine if everything is going well.

Is there a limit ? If so, what is it worth ?

3 – Wait & sig

3.1

Create a child process that falls asleep for 10 seconds and then returns the hand to his father who was waiting for him with `wait`

a) What is the status returned by the son when everything is normal ?
b) What is the status returned when you kill the son prematurely with a signal of your choice ?
c) Observe what happens when the father does not wait on the son when he is dead.
d) What happens when it's the father who is killed prematurely ?

For all these questions, you can use the following commands from a 2nd terminal :

- `ps -f xxx` to get information about the pid process `xxx`
- `kill -SIGUSR1 xxx` to send the signal `USR1` to the pid process `xxx`
- `kill -l` returns the list of possible signals

3.2

Generate by a loop n processes from the same father. Put the father on hold for all his children. Display the status of children as they disappear. The children make a call to `sleep(i)` with different `i` values.

4 – Exec + argv = <3

4.1

Create a "slave" program capable of running different unix commands when given the corresponding character string on the command line. For example :

```
$ ./slave whoami
Yes, my master:
ensea
$ ./slave date +%F
Yes, my master:
2019-02-05
```

4.2

Using the arguments `argc` and `argv` of the function `main()`, create a program that executes itself 5 times in a row (without using `fork`).

Do the same thing again, but use the `environ` instead of `argv`.