

Please read Sections 3.1 – 3.4 and 3.6 – 3.7 of the textbook and then answer the following, trying not to look at your notes or at the textbook. Quiz #3, on Thursday 5th October, will consist exclusively of questions taken from the Part I of this homework.

Part I — Short Questions

Question 1

Explain in a few word what a process is.

Question 2

What is the state of the process being executed on the CPU?

Question 3

Describe the actions taken by the kernel to context-switch between processes.

Question 4

Give a reason why a process could make a transition from the waiting (a.k.a. blocking) state to the ready state.

Question 5

What happen to a process when it is swapped out? What is the benefit of swapping out processes?

Question 6

What is the name of the part of the O.S. that picks, among the processes in the ready queue, the one to execute on the CPU?

Question 7

How many parents do a process have? Generally, can a process have more than two children?

Question 8

What is a pid?

Question 9

What does the `getppid()` function return?

Question 10

What is an orphan? What is a zombie?

Question 11

Does message-passing involve the kernel?

Question 12

Name two ways of sending a message from a process P_1 to several processes P_2, \dots, P_n .

Question 13

What is a rendez-vous between two processes?

Question 14

Why does a system have many ports?

Question 15

What are the differences between named pipes and ordinary pipes?



Part II — Problems

This time, only one problem requires a computer, while the other relies on your capacity to analyze some code. I'll assume that you will have successfully completed them by the time Homework #4 is released (Thursday 5th October), so don't wait and let me know if you had difficulties doing them.

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

int main(){
    pid_t pid;
    pid = fork();
    if (pid == 0)
    {
        fork();
    }
    else
    {
        printf("Where am I?\n");
        execlp("ls", "ls", NULL);
        printf("Hi Mom!\n");
    }
    printf("Hi Dad!\n");
    exit(0);
}
```

Listing 1: fork.c

Problem 1

Look at the code displayed in Listing 1, and answer the following:

- What is the datatype `pid_t` used for?
- If you execute this program how many processes would be created? How many children would have the process you created by executing the program?
- Will “Where am I?” ever be printed? If yes, how many times, and by who: the parent, a child, some other process?
- Will “Hi Dad!” ever be printed? If yes, how many times, and by who: the parent, a child, some other process?
- Will “Hi Mom!” ever be printed? If yes, how many times, and by who: the parent, a child, some other process?
- If you want to use the `execl` function instead of `execlp`, what would you need to change?
- If you want to load `cd` with the arguments `.. /` instead of `ls` without arguments, what do you need to change in that program?

Problem 2

Run your virtual machine, created a “03” folder in your “Desktop/HW/” folder, and copy the code located at “Desktop/osc9e-src/ch3/unix_pipe.c” in it. Optionally, you may want to install a nicer IDE using

```
sudo apt-get install geany
```

and typing “osc” as password.

- (a) Examine the code `unix_pipe.c`, compile it, and run it. What do you observe?
- (b) Now, let us look at the code again:
 - i. What does `pipe(fd)` do? What value does it return? Use `man pipe` to answer those questions.
 - ii. Change the value of the constant `BUFFER_SIZE` to 6, compile and execute the program. What do you observe? Change it back to 25.
 - iii. Switch the values of `READ_END` and `WRITE_END`. Compile and execute the program: what happened?
 - iv. Try to modify your program, so that it actually checks that the pipe is still open before writing in it or reading from it. You can get inspiration from this question: <https://stackoverflow.com/a/19020926/>.

