Operating systems

# first homework, spring semester 2023

Julian Ewaied 215543497 Namir Ballan 326165156

*Question 2*

It is because system calls are highly used, not to mention that when we move to kernel mode, we lose the multi-process feature of the CPU – meaning, all processes must wait until the system call is finished to proceed, which means that the computer is stuck. Therefore, we need a quick access to the parameters of the system call, which do not require privileges checking and/or loading through multiple levels of memory hierarchy.

*Question 3*

there are 4 possible outputs for the program:

1. no output – nothing is printed
2. 8
3. 8, 13
4. 13, 8

explanation: the first line is a fork() call. there is 2 cases:

if fork fails, then its return value would be -1. so the program will enter the first if statement, print nothing, and exit.

if the call doesn’t fail, then we have two processes: the parent process and the son process.

in the parent process, the fork() will return a positive non zero value and so the program will enter the else if statement and it will print the pid of the process, which is 8 for the pid of the parent.

in the son process, fork() returns 0 so the son will enter the else statement and it will run execv().

if execv() succeeds it wont return and the son process will print nothing. if execv() fails the son will print its pid,which is 13. In the case where the son prints nothing, the output will be 8. if the son printed 13, we don’t know who will print first, the son or parent process, so its possible that the output is 13,8 or 8,13.

*Question 4*

The code defines a local variable 'value'. Then, it calls the system call fork(). Assuming it succeeds, the parent enters the if condition, and enters a 'wait()', which means that the process status becomes TASK\_INTERRUPTIBLE, and therefore the only process currently running in the shell is the son. The son however, skips the if condition, and prints value, which is 0, then adds 4 to it (value=4 now), and returns it to its parent. Its parent exists the TASK\_INTERRUPTIBLE status, and receives a number, which by taking WEXITSTATUS() of it we can retrieve the return value of the son – 4. Now we proceed with the code of the parent: value=7, then prints value to the screen, and proceeds with its code (no more printings). Therefore we can say that the code prints:

0

7

And exits.

If the 'fork' fails, there's no son and fork returns -1. Therefore, we enter the if condition and value is not changed by wait and we do not wait, which means that value is 3, and thus we print 3 to the screen and exit, which means that the output is only 3.