OPERATING SYSTEMS

FIRST HOMEWORK, SPRING SEMESTER 2023

Julian Ewaied 215543497 Namir Ballan 326165156

Question 1

Because if the number of parameters for the function is not constant, we need access to the first parameter (which has the number of parameters that got passed).

Question 2

It is because when moving to kernel mode, we change the stack to the kernel stack, which makes the user stack inaccessible (also user's irresponsibility in terms of memory management might lead to a stack overflow, which means that the sys-call parameters are not safe in user's stack).

Question 3

there are 4 possible outputs for the program:

1. no output – nothing is printed
2. 8
3. 813
4. 183
5. 138

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, and not that its possible that the 8 is printed after 1 is printed and before 3 is printed, so its possible that the output is 138 or 813 or 183.

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, then the returned value is 0, and thus the bits that WEXITSTATUS returns are 0's, which means that value is zero, which means that value is 3, and thus we print 3 to the screen and exit, which means that the output is only 3.