

HW-6-Basic-Processes

CS 503 Systems Basics

100 Points

The following exercises come from CSAPP3. (20 points each)

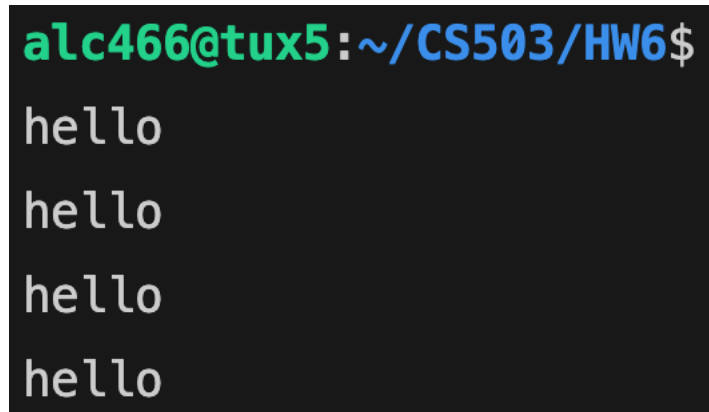
File csapp.h is in FILES/PROGRAMS. It is only relevant here because it contains functions Fork() and Wait(). The first letter is capitalized.

Problem 8.11

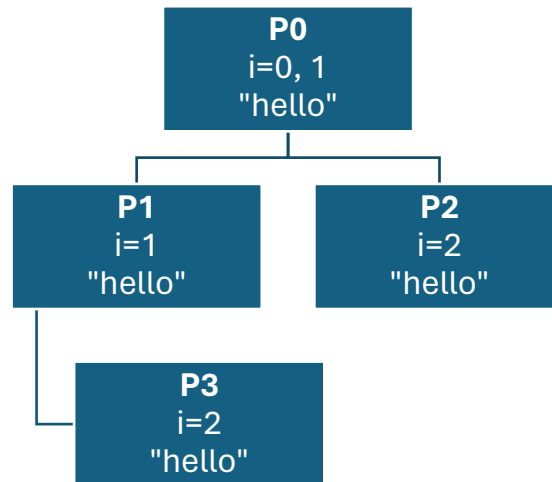
How many "hello" output lines does this program print?

```
#include "csapp.h"
int main()
{
    int i;
    for (i = 0; i < 2; i++)
        Fork();
    printf("hello\n");
    exit(0);
}
```

This program will print out "hello" 4 times.

A terminal window with a black background. The prompt is 'alc466@tux5:~/CS503/HW6\$' in green and blue. Below the prompt, the word 'hello' is printed four times, one on each line, in white. This demonstrates the output of the program where a single loop with two iterations, each calling Fork(), results in four 'hello' messages being printed (two from the parent and two from the children).

The Fork() function creates a duplicate child process every time it is called. After creation, execution starts from after the fork call. Here, Fork() is being called on every iteration of the loop and the loop runs twice, so 2 calls are made in total. The following is the process tree for this function where P represents each process. *i* represents the iteration variable in the loop after the process was created and run.



1. Loop starts in the parent P0 at $i = 0$ (first iteration)
2. Parent creates child process P1
3. Variable i becomes 1 for both P0 and P1 (second iteration)
4. Processes P2 and P3 will be created by the second fork call
5. Variable i becomes 2 for both processes P2 and P3, which means the loop is done
6. All processes have been created and each one will print "hello."

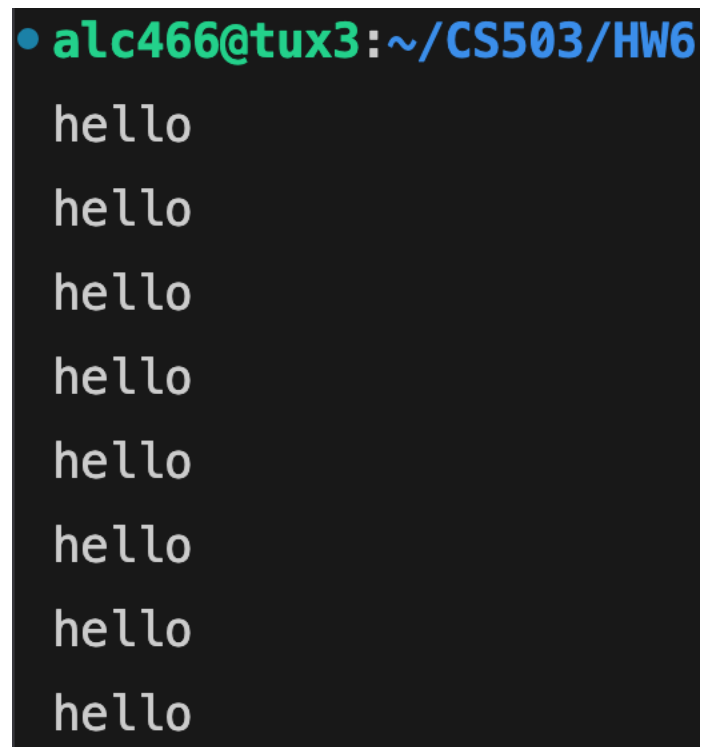
Problem 8.12

How many "hello" output lines does this program print?

```
#include "csapp.h"
void doit() {
    Fork();
    Fork();
    printf("hello\n");
    return;
}

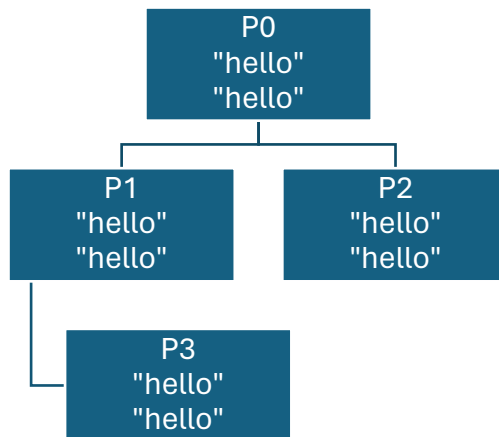
int main(){
    doit();
    printf("hello\n");
    exit(0);
}
```

This program will output "hello" 8 times.

A terminal window with a black background and white text. The prompt is '• a1c466@tux3:~/CS503/HW6'. Below the prompt, the word 'hello' is printed eight times, one on each line.

```
• a1c466@tux3:~/CS503/HW6
hello
hello
hello
hello
hello
hello
hello
hello
```

There are 2 calls to fork inside of the doit() function. However, the execution is then wrapped to main with another print("hello") at the end. Here is what the process tree for this program looks like.



1. Main function calls doit()
2. Parent process creates child process P1 (first fork call)
3. Both child and parent processes are on the second fork call now
4. Child process P1 and parent process P0 create processes P2 and P3 (second fork call)
5. All 4 processes are now created and print "hello" from within doit()
6. There is one more print("hello") statement in main() and each created process must execute this line as well, so "hello" gets printed again 4 more times

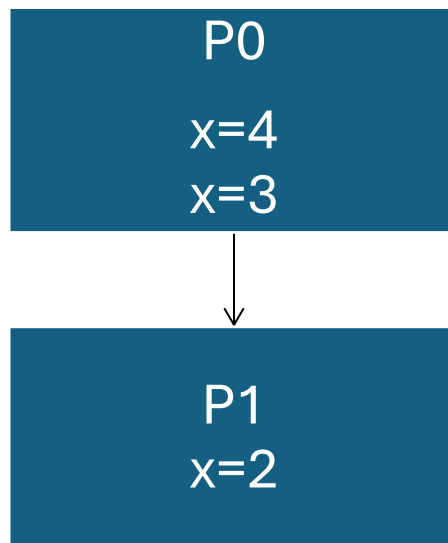
Problem 8.13

What is one possible output of the following program?

```
#include "csapp.h"
int main() {
    int x = 3;
    if (Fork() != 0)
        printf("x=%d\n", ++x);
    printf("x=%d\n", --x);
    exit(0);
}
```

```
• alc466@tux5:~/CS503/HW6$
x=4
x=3
x=2
```

The above output could be one possibility. Here is what the fork process tree looks like for this program.



1. Parent P0 creates child process P1 (both processes have $x = 3$) and there are no more fork calls

2. In this scenario, the parent process runs first so the if condition (`pid != 0`) is true. The current value of `x` (3) is first incremented and then printed (`x=4`). Afterwards, the next statement runs. It decrements `x` back and prints it (`x=3`)
3. In the child process P1, the if condition is false, so only the second print runs. It decrements the current value of `x` down and prints it (`x=2`)

The output depends on which process runs first after the fork call and their concurrent nature throughout the whole execution of the program.

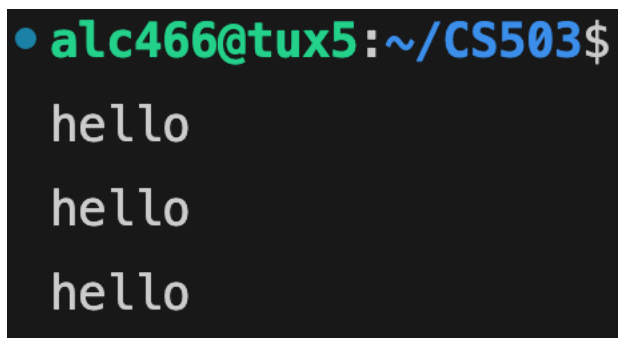
Problem 8.14

How many "hello" output lines does this program print?

```
#include "csapp.h"
void doit() {
    if (Fork() == 0) {
        Fork();
        printf("hello\n");
        exit(0);
    }
    return;
}

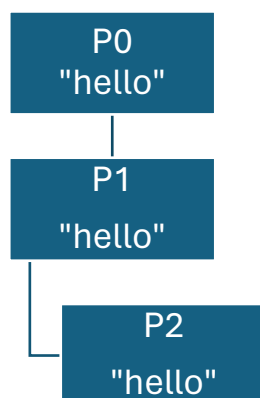
int main() {
    doit();
    printf("hello\n");
    exit(0);
}
```

This program will print "hello" 3 times.



A terminal window with a black background and green text. The prompt is `• alc466@tux5:~/CS503$`. Below the prompt, the word "hello" is printed three times, one on each line.

There are 2 fork calls in this program and they are called from main through another function `doit()`. Below is the process tree for all the forks!



1. Parent P0 creates child process P1
2. In P0, nothing else happens in terms of forking because the pid is not 0 (if condition is false) so only the "hello" in the main function is printed. After that, parent process is terminated
3. In P1 however, another fork call is made creating process P2
4. Both P1 and P2 then print "hello" because the pid is 0 for both (the if condition is true). Then, they immediately terminate.

Problem 8.16

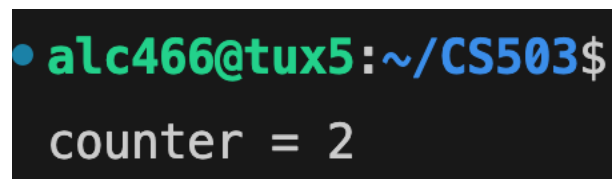
What is the output of the following program?

```
#include "csapp.h"

int counter = 1;

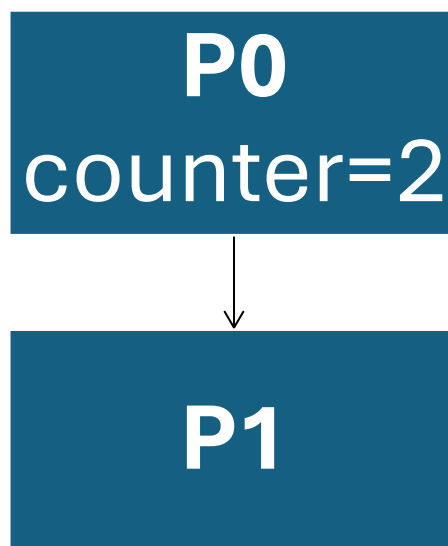
int main() {
    if (Fork() == 0) {
        counter--;
        exit(0);
    }
    else {
        Wait(NULL);
        printf("counter = %d\n", ++counter);
    }
    exit(0);
}
```

This program will only output "counter = 2".



A terminal window with a dark background. The prompt is 'a1c466@tux5:~/CS503\$'. The output is 'counter = 2'.

This program has only one fork call. The fork process tree with the outputs for each process therefore looks like this.



1. For each process, the program checks the pid of each process created by the fork call in the if condition
2. When the parent process P0 runs (pid == 0 is false) it first must wait for all its child processes to complete (indicated in the else block). In this case, there is only 1 child process P1, so it will wait when P1 finishes. If P1 completes first before the wait call, this wait call will automatically return 0 (the pid of the child) and the parent process will continue onward.
3. In the child process P1 (pid == 0 is true), its copy of the counter variable is first decremented to 0 and then the child process terminates
4. After P1 has been terminated, P0 increments its own copy of counter and prints it (counter = 2). After this, the whole program is terminated.