ICS Answer Sheet #11

Sakar Gopal Gurubacharya
s.gurubacharya@jacobs-university.de

Problem 11.1: fork system call

a. **Assume the program has been compiled into cnt and that all system calls succeed at runtime. How many child processes are created for the following invocations of the program?**

   Explain how you arrived at your answer

   At first, I compiled them using the gcc compiler and got all the individual child processes for each execution process.

   i. **./cnt**

      Child processes: **0**

      Nothing, because it doesn't enter the loop.

   ii. **./cnt 1**

      Child processes: **2**
      (1,1)
      (1,0)

      Goes to the for loop, calls the function and inside the function, child processes are created. However, the parent process cannot be made and so only 2 child processes are created.

   iii. **./cnt 2**

      Child processes: **3**
      (2,2)
      (2,1)
      (2,0)

      Similar to having 1, having 2 would make it loop through more times and would only create one child every time since it cannot satisfy fork () == 0 and hence no room for recursion.

iv. ./cnt 1 2 3

Child processes: **9**

```
(1,1)
(2,2)
(1,0)
(3,3)
(2,1)
(3,2)
(2,0)
(3,1)
(3,0)
```

With more arguments, the parent processes can be created alongside the child processes.

b. Remove the line exit(0) and compile the program again. What is printed to the terminal and How many child processes are created for the following invocations of the program?

Explain how you arrived at your answer.

v. ./cnt 1

Child processes: **2**

```
(1,1)
(1,0)
```

Same as before, goes to the for loop, calls the function and inside the function, child processes are created.

vi. ./cnt 2

Child processes: **3**

```
(2,2)
(2,1)
(2,0)
```

Same as before, similar to having 1, having 2 would make it loop through more times and would only create one child every time since it cannot satisfy fork () == 0 and hence no room for recursion.

vii. ./cnt 1 2

Child processes: **8**

```
(1,1)
(2,2)
(1,0)
(2,2)
(2,1)
(2,1)
(2,0)
(2,0)
```

Since it cannot exit the loop when fork () == 0 is satisfied, it creates more child processes and more parent processes as well.

viii. ./cnt 1 2 3

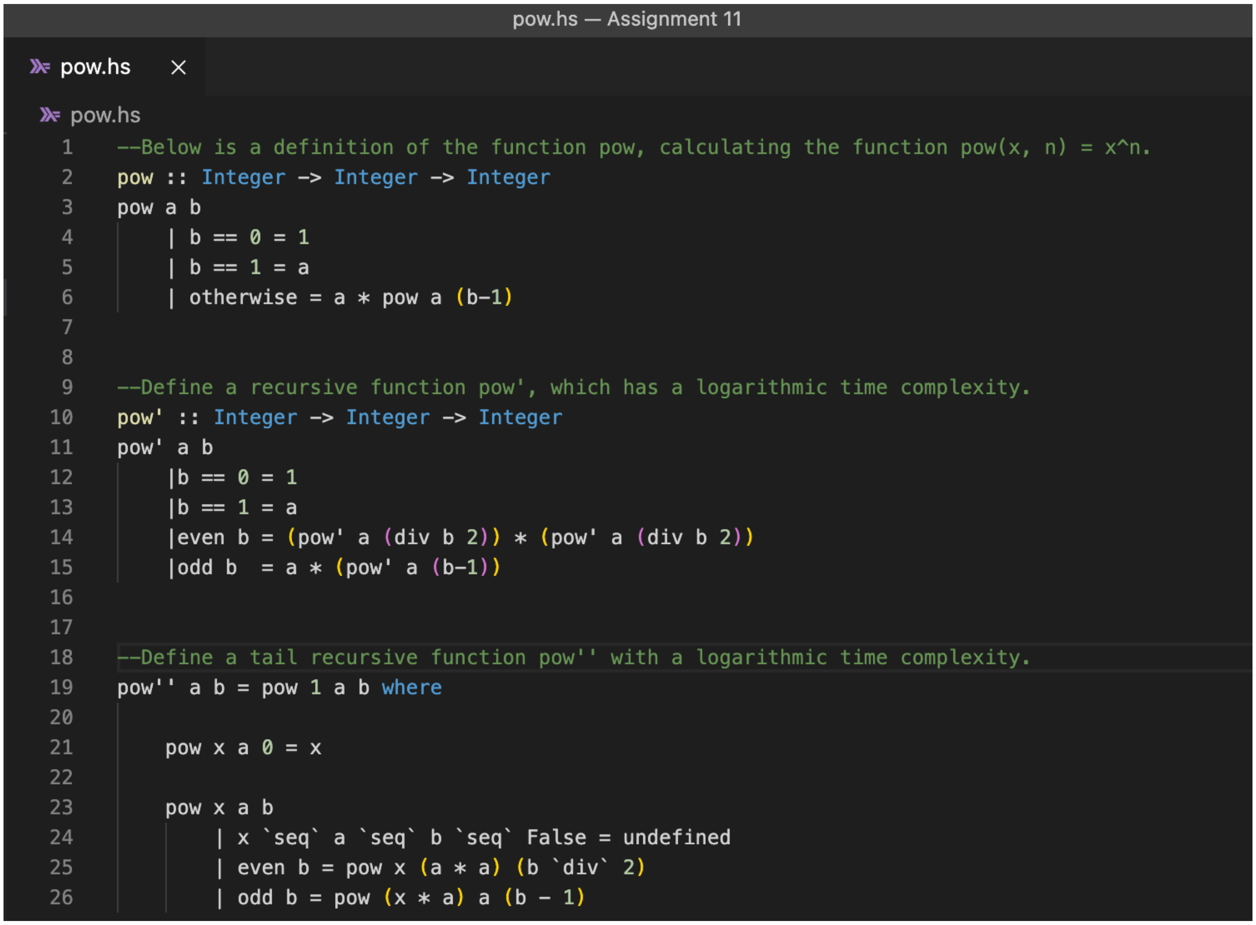
Child processes: **32**

```
(1,1)
(2,2)
(1,0)
(2,2)
(3,3)
(2,1)
(2,1)
(3,3)
(3,2)
(3,3)
(3,3)
(2,0)
(3,3)
(2,0)
(3,3)
(3,2)
(3,1)
(3,2)
(3,2)
(3,2)
(3,1)
(3,2)
(3,0)
(3,1)
(3,1)
(3,1)
(3,0)
(3,1)
(3,0)
(3,0)
(3,0)
(3,0)
```

With the absence of exit(0), the program keeps on continuing and also repeats the same child processes because it cannot exit the loop once fork is satisfied.

Problem 11.2: stack frames and tail recursion

(pow.txt in the same .zip file)



```haskell
--Below is a definition of the function pow, calculating the function pow(x, n) = x^n.
pow :: Integer -> Integer -> Integer
pow a b
    | b == 0 = 1
    | b == 1 = a
    | otherwise = a * pow a (b-1)


--Define a recursive function pow', which has a logarithmic time complexity.
pow' :: Integer -> Integer -> Integer
pow' a b
    |b == 0 = 1
    |b == 1 = a
    |even b = (pow' a (div b 2)) * (pow' a (div b 2))
    |odd b  = a * (pow' a (b-1))


--Define a tail recursive function pow'' with a logarithmic time complexity.
pow'' a b = pow 1 a b where

    pow x a 0 = x

    pow x a b
        | x `seq` a `seq` b `seq` False = undefined
        | even b = pow x (a * a) (b `div` 2)
        | odd b = pow (x * a) a (b - 1)
```

Problem 11.3: evaluation of arithmetic expressions

(Not attempted)