Dor Gross, 039344999

Operating Systems
Exercise 2

Part 2

1. Each thread performs exactly 7 iterations, as the loop is controlled by a local unshared variable `i`.
2. `x++` is not an atomic operation (as seen in class), therefore, we cannot know exactly the value of `x` after both threads (`t1` and `t2`) are done.
3. We can bound `x`'s value:

   Upper bound:

   a. No contention - assume both threads never access `x` together, as if `x++` is an atomic operation.

   b. `x` is accessed 14 times (7 by each thread), therefore its final value is **18**.

   Lower bound:

   a. Full contention - each time, both threads access `x` together in the following manner: One thread reads it, while the other increases it 6 times, then the first one overrides `x`'s value. Then the second reads `x`'s value while the first increases it 6 times, and then the second one overrides it value.

   b. A total of only 2 increases leads to a final value of **6**:

Time	Thread 1	Thread 2
1*	read `x=a`, add 1	
2*		read `x=a`, add 1
3		write `x=a+1`
4*		read `x=a+1`, add 1
5		write `x=a+2`
...		...
13		write `x=a+6`
14	write `x=a+1`	
15*		read `x=a+1`, add 1
16*	read `x=a+1`, add 1	
17	write `x=a+2`	
18*	read `x=a+2`, add 1	
	...	
27	write `x=a+7`	
28		write `x=a+2`

* Since "`add 1`" is not a critical section, it may occur in any stage between its corresponding read and the write operations.

And since the print command is only issued after both threads are done, the output of the print command will be an integer number in the range 6-18 (inclusive).

Part 3

A.

`fork()` creates a new process with its own memory, therefore `x`, which is a global variable in each process, is not shared between different processes and its initial value is determined, like the variable `i`, when the process image is duplicated in the `fork()` command. And since `x` is not shared between the processes, the output will be $2^{10} = 1024$ serieses of multiples of 2 with lengths varying from 1 to 10, shuffled due to context switch between the processes, where 2 have the length of 10 (starting from 1), 2 have the length of 9 (starting from 2), 4 have the length of 8 (starting from 4), and so on.

B.

Since `printf()` is not an atomic operation, the printed format would be calculated in one operation, and the print itself in a different one, and therefore it might happen that a context-switch would occur between these 2 operation. In addition, a context-switch might occur right after the thread creation, changing the value of `k` before the previous thread had managed to print its value.

Due to the issues mentioned above, the first thread could print any of the values 1,2 and 3, the second thread could print 2 or 3, and the third thread could only print 3 (as 3 atomic additions were made before it printed the value of `k`). And since the order of the values depends on the context-switches between the threads, any series of those value is a possible output.