Random Testing

Question 1. Consider the following concurrent program with two threads and shared variable x. Variables tmp1 and tmp2 are local to the respective threads. This program has a concurrency bug: it can lose an update to x.

```
    Thread 1              Thread 2
 1: tmp1=x             4: tmp2=x
 2: tmp1=tmp1+1        5: tmp2=tmp2+1
 3: x=tmp1             6: x=tmp2
```

a. Write one possible execution of the six statements that does not cause a concurrency bug.

Answer: Any order which performs a read and a write before the second read, e.g., 1 2 3 4 5 6 OR 4 5 6 1 2 3.

b. Write one possible execution of the six statements that does trigger a concurrency bug.

Answer: Any order which performs both reads before any write, e.g., 1 4 2 5 3 6.

c. What is the depth of the concurrency bug?

Answer: 2

d. Specify the ordering constraints needed to trigger the bug.

Answer: (4, 3) (1, 6)

Question 2. Consider the following pseudo-Java function, in which `HashMap<char,int>` is used. A `HashMap<K,V>` is a data structure that associates a value of type `V` to a key of type `K`. The value `v` associated with a key `k` can be set with the API call `put(k, v)`, and the value associated with the key `k` is returned by the API call `get(k)`. For this problem, if no value has been associated with `k`, then assume `get(k)` returns `0`.

```
double charRatio(String s, char a, char b) {
  int N = s.length();
  HashMap<char,int> counts = new HashMap<char,int>();
  for (int i = 0; i < N; i++) {
    char c = s.charAt(i);
    int v = counts.get(c);
    counts.put(c, v+1);
  }
  return counts.get(a) / counts.get(b);
```

```
}
```

Describe how you could use a fuzzer to test this function. What bugs would you expect a fuzzer to identify in this function? What bugs would be more challenging for a fuzzer to identify? Explain your reasoning fully, including any assumptions you are making.

Answer: There are three main bugs in this program:

- the possibility of a null dereference if s == null,
- the possibility of a division by zero when b doesn't appear in the input string, and
- the possibility of counts.get(a) / counts.get(b) not equaling the correct ratio of a's to b's in the input string due to integer division.

Fuzzing would likely quickly detect the division-by-zero error by generating a string s with no instances of the char b. This would depend on the implementation of the fuzzer. If the fuzzer only generated '0' and '1's, then it would likely be difficult for b not to appear in s. On the other hand, if the fuzzer uniformly generated legal strings of length 100 from the ASCII character set (and b were uniformly chosen from the ASCII character set), then there would be a 45% chance of the bug being triggered. (As the length of the input string grows, the probability of selecting b so that b were not in s would vanish quickly, though.)

The null-dereference error would also likely be caught by the fuzzer using a similar argument.

The integer division bug would be harder to detect, as fuzzing does not generally entail matching the output of a function against an expected output: we usually just give the function random strings until we detect a crash.