CMPSCI 630 Systems **Fall 2019**

Lecture 17

Lecturer: Emery Berger *Scribes: Yash Bidasaria*

17.1 Replicated State Machines

Distributed systems are conventionally created as Replicated State Machines. This paradigm has every machine operate on their own state machines assuming inputs come in deterministically. This assumption even allows for asynchronous inputs. This is the standard way of implementing distributed systems. One drawback is that algorithms need to be created by hand to account for these assumptions.

As RSMs need determinism, one cannot implement RSMs with threads as they are non-deterministic. There are various cases like User Input, Scheduling and even the local Time of machines that make them non-deterministic. DieHard in a way get RSMs for free to an extent as they get probabilistic errors. RSM with determinism leads to Bohr Bugs while RSM with DieHard leads to Heisen Bugs.

17.2 DieHard, DieHarder

DieHard was reimplemented at Microsoft due to licensing issues and first named it Robust Heap and later Fault Tolerant Heap.

DieHarder was created as a successor to DieHard while keeping the Threat Model and several other security cases in mind. This project was aimed to better understand the security of memory allocations. The modifications to DieHard for security led to a security - reliability tradeoff. For example, in a brute force attack, if the someone keeps attacking you, you can just stop. But as the system is secure, it is not available.

In DieHard when some memory is freed, the system becomes susceptible to a after free bug as DieHard does not actually change the contents. To combat this bug, DieHarder fills it with random values. DieHarder as contributed to a lot of changes in Windows 8. It is one of the most secure memory allocators. There is another tradeoff between Bits of Entropy used for random values and scalability.

17.3 Profilers

Profilers are used to analyse the programs time/memory usage in order to create better software systems.

prof is one of the earliest profilers. At every line of execution, it increment the counter for that line. This created a lot of overhead resulting in a slow program. This profiler had another assumption of every line having the same cost. This technique although does allow to identify the lines that are executed the most and are the performance bottleneck.

A solution could be to measure the time consumed for each line. The time although needs to be corrected as the cpu might context switch causing the time consumed to be a lot more. There are three times: Wall Clock Times, User Time and the System Time. For profilers, User Time should be used. This solution also adds a lot of overhead to the program. It disrupts the cache having the Probe effect. The Probe Effect states that the program might change when adding an instrument to measure its performance.

Another profiler called gprof takes into account the above problems. It has a low overhead. is accurate and is unbiased. It uses statistical profiling which uses the Law of Large numbers. This kind of profiling samples the memory and intervals, and then using the Law of Large Numbers eventually converges to the true value. But with sampling at a constant interval can create a bias. To combat this problem, sampling is conducted at random intervals.

17.4 Key Terms

1. Replicated State Machines
2. Never Role your own Cryto (or anything else)
3. Fear Uncertainty Doubt
4. Threat Model
5. Security - Reliability Tradeoff
6. Bits of Entropy - Scalability Tradeoff
7. Probe Effect
8. Statistical Profiling