COMPSCI 630 Systems **Fall 2019**

Lecture 10

Lecturer: Emery Berger *Scribe: Priyanka Mary Mammen*

10.1 Concurrency Error Detectors

There are tools that can help you debug concurrency errors

- Race detectors
- Replay based debugging

There are two types of race detection analysis tools: static and dynamic.

Static analysis tool knows the future, and sees only the abstract values, resulting a loss in precision. Static analysis can give rise to false positives. Static analysis cannot prove whether the program halts. For instance, in the following code

```
int x= 12;

x= x+1;
x=100;

f=..... read value from socket

if(f>1)
{
  x=x+1;
}
else
{
  x=100;
}
// x could be 13/100(loss of precision)
// more the length of the code,  more approximate
will be the value
```

Figure 10.1:

Whereas in dynamic race detection approach, they use only the present and past values. They are always precise and may be not generalizable. There are no false positives and lower RECALL in Dynamic Analysis. They only represent races in a particular execution.

Static analysis is very useful for bug detection in a variety of contexts.

Bug detection v/s Formal verification

Bug detection needs only partial specification, whereas formal verification needs complete specification.

In formal verification, we need to prove whether program P implements its specifications S. The specification is normally expressed in some sort of logic. Basically, it means if we could have executed S, we should get the same result as when we executed P. It maps all the inputs to all the outputs(that is why it is called complete specification). Formal verification is a work in progress. People are working on things that:

- have narrow APIs (very easy to specify)
- are well specified
- are really important

These are the right targets for formal specification. Microsoft had a project called Project Everest about https. They used OpenSSL cryptographic library. Sometime later, "Heart bleed bug"(leaks information about SSL private keys) was identified as a vulnerability in OpenSSL after formal verification.

There is a notion called Trusted Computing Base which may be on the order of 1000 lines of code.

Bug detection is often much weaker as it needs only a partial specification. For instance, program P never derefs NULL. It is totally fine by principle for a program to deref NULL, but if it is a Java program, it will throw a NULL POINT EXCEPTION.

Google has a particular kind of race detector called thread sanitizer.

Right now, we mostly use Dynamic Analysis. Dynamic analysis can be further classified into: 1) Lockset analysis 2) Happens-before-analysis

In Lockset analysis, every shared location is protected by a lock.

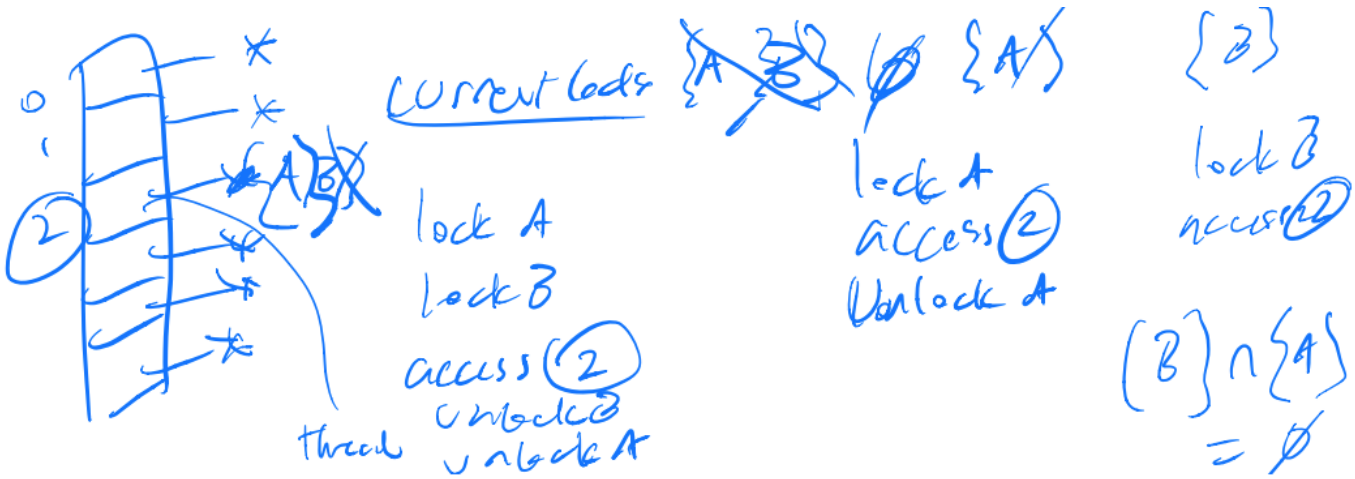

Figure 10.2:

Whereas happens before analysis is quite related to distributed systems. You do keep a timer, at the end of the day, you will have a number and get an order in relationships.

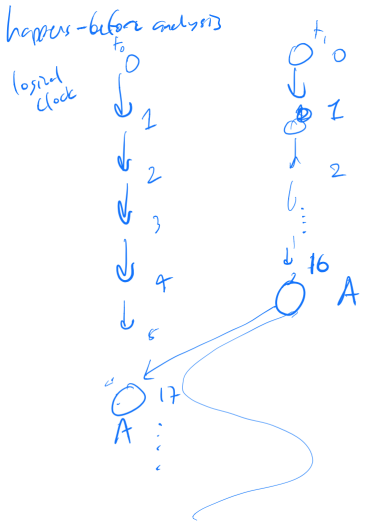

Figure 10.3:

Why threads are needed?

- Parallelism
- Coordination
- Hiding Latency (Major reason for using threads, It give a massive improvement in performance)

"throughput" is the rate of jobs getting done. There is always a throughput-latency trade off. Best example is batch v/s interactive processes.

10.2 UNIX - time sharing system

Time sharing system lets mutliple users have the effect of using the computer at the same time. Predecessor of Unix is MULTICS, which is developed by a team from MIT, GE and Bell Labs. Some of notions introduced by MULTICS are:

- 'hierarchical' file system, whereas the previous Operating Systems had 'flat' file systems.
- Symbolic Links - It is a pointer to the target file. For instance, "foo" can be used to represent "/bnq/biq".
- Commandline Processor, i.e, Shell
- Security rings, where Kernel is the most privileged and applications are the least privileged