

Analysis of Concurrent and Distributed Programs

Burcu Kulahcioglu Ozkan and Soham Chakraborty

09.02.2022

(1) Introduction

- Course logistics & assessment (Link)
- Lecture content (Link)
- Course projects (Link)

Overview of concurrency

- Paradigms
- Concepts

Interactions

Multiple computations *that may/may not influences one another*

- Parallelism (SIMD, MIMD, ...)
 - Independent computation on multiple threads/processes on independent data
- Distributed computing
 - Independent computation on multiple machines with message passing
- Asynchronous programming
 - Multiple tasks on single/multiple threads that may share memory
 - Event driven systems
- Shared memory concurrency
 - Multiple threads that communicate by shared memory

Concurrent Programming: Motivation

Pros

- + Concurrency improves performance

Cons

- Concurrent programming is hard
- May result in tricky bugs in programs

Requires careful analysis

Concurrent Programming: Motivation

Pros

- + Concurrency improves performance

Cons

- Concurrent programming is hard
- May result in tricky bugs in programs

Requires careful analysis

This course !

Example

$$X = 0;$$

$$X = X + 1; \quad || \quad X = X + 1;$$

What is the final value(s) of X ?

Example

$$X = 0;$$

$$X = X + 1; \quad || \quad X = X + 1;$$

What is the final value(s) of X ?

- Expected: $X = 2$.

Example

$X = 0;$

$a = X;$		$b = X;$
$a = a + 1;$		$b = b + 1;$
$X = a;$		$X = b;$

Example: Execution (1)

$X = 0;$

$a = X;$	$// 0$	$b = X;$
$a = a + 1;$		$b = b + 1;$
$X = a;$		$X = b;$

Example: Execution (1)

$X = 0;$

$a = X;$	// 0		$b = X;$
$a = a + 1;$			$b = b + 1;$
$X = a;$	// 1		$X = b;$

Example: Execution (1)

$X = 0;$

$a = X;$	// 0		$b = X;$	// 1
$a = a + 1;$			$b = b + 1;$	
$X = a;$	// 1		$X = b;$	

Example: Execution (1)

$X = 0;$

$a = X;$	// 0		$b = X;$	// 1
$a = a + 1;$			$b = b + 1;$	
$X = a;$	// 1		$X = b;$	// 2

Example: Execution (2)

$X = 0;$

$a = X;$	$// 0$	$ $	$b = X;$	$// 0$
$a = a + 1;$		$ $	$b = b + 1;$	
$X = a;$		$ $	$X = b;$	

Example: Execution (2)

$X = 0;$

$a = X;$	// 0		$b = X;$	// 0
$a = a + 1;$			$b = b + 1;$	
$X = a;$	// 1		$X = b;$	// 1

Example

$$X = 0;$$

$$X = X + 1; \quad || \quad X = X + 1;$$

What is the final value(s) of X ?

Example

$$X = 0;$$

$$X = X + 1; \quad || \quad X = X + 1;$$

What is the final value(s) of X ?

- Expected: $X = 2$.
- Reality: $X \in \{1, 2\}$

What are the semantics of these primitives?

What are the semantics of these primitives?

Given a concurrent program and an outcome,

What are the semantics of these primitives?

Given a concurrent program and an outcome,

- Is it correct?
- Is it a bug?
- What are common concurrency bugs?
- Will it happen in *all* execution?
- Will it happen in *at least one* execution?

What are the semantics of these primitives?

Given a concurrent program and an outcome,

- Is it correct?
- Is it a bug?
- What are common concurrency bugs?
- Will it happen in *all* execution?
- Will it happen in *at least one* execution?

After finding a bug:

- How to fix it?
- Is it the best way to fix it?
- Is it affecting performance?

Going Back to the Example

$X = 0;$

$a = X; // 0$		$b = X; // 1$
$a = a + 1;$		$b = b + 1;$
$X = a; // 1$		$X = b; // 2$



$X = 0;$

$a = X; // 0$		$b = X; // 0$
$a = a + 1;$		$b = b + 1;$
$X = a; // 1$		$X = b; // 1$



Going Back to the Example

```
      X = 0;  
a = X; // 0 || b = X; // 1  
a = a + 1;   || b = b + 1;  
X = a; // 1 || X = b; // 2
```

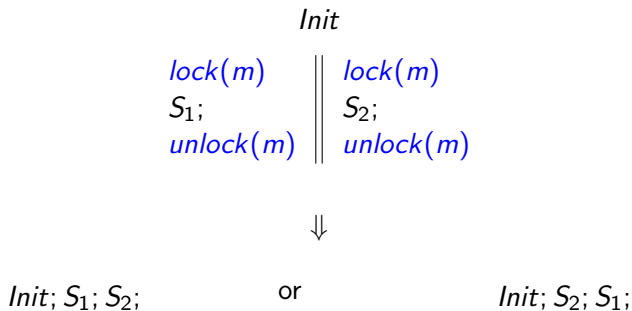


```
      X = 0;  
a = X; // 0 || b = X; // 0  
a = a + 1;   || b = b + 1;  
X = a; // 1 || X = b; // 1
```



Solution: use lock/unlock primitives.

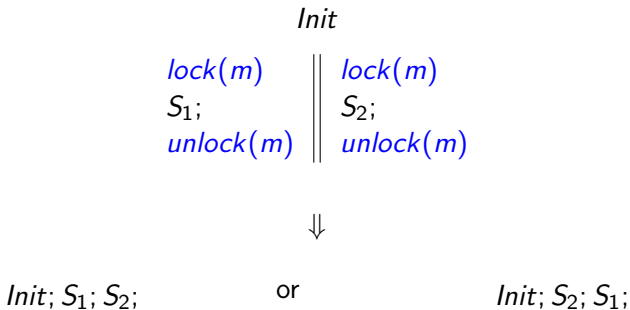
```
      X = 0;  
  
lock(m) || lock(m)  
a = X;   || b = X;  
a = a + 1; || b = b + 1;  
X = a;   || X = b;  
unlock(m) || unlock(m)
```



lock/unlock

- Reduces concurrency

Solution: Tradeoff



lock/unlock

- Reduces concurrency

Goal: minimal lock/unlock for correct programming

Locking mechanism.

- Primitive: mutex
- Properties: mutual exclusion

Errors

- Datarace
- Atomicity violation

Pitfalls

- Deadlock
- Livelock
- Non-termination

Lock free programming

- Non-blocking concurrency

Concurrency analysis for multithreaded programs

- Race detection
- Atomicity violation detection

Analysis techniques

- Static and dynamic analysis
- Model checking
- testing

Sequential consistency (SC)

Weak memory (Non-SC) models

- TSO
- PSO
- RMO
- RA
- ...

Analysis of weak memory programs

- Many flavors of concurrency
 - Single-threaded/multi-threaded asynchronous
 - Event driven
 - Distributed

Distributed Programming

- Many flavors of concurrency
 - Single-threaded/multi-threaded asynchronous
 - Event driven
 - Distributed
- Single-threaded/multi-threaded asynchronous, event driven, distributed concurrency

Distributed Programming

- Many flavors of concurrency
 - Single-threaded/multi-threaded asynchronous
 - Event driven
 - Distributed
- Single-threaded/multi-threaded asynchronous, event driven, distributed concurrency
- More on distributed concurrency

Distributed Programming

- Many flavors of concurrency
 - Single-threaded/multi-threaded asynchronous
 - Event driven
 - Distributed
- Single-threaded/multi-threaded asynchronous, event driven, distributed concurrency
- More on distributed concurrency
- Concurrency analysis for distributed programs

Distributed Programming

- Many flavors of concurrency
 - Single-threaded/multi-threaded asynchronous
 - Event driven
 - Distributed
- Single-threaded/multi-threaded asynchronous, event driven, distributed concurrency
- More on distributed concurrency
- Concurrency analysis for distributed programs
- Replicated systems: Linearizability & CAP Theorem

Distributed Programming

- Many flavors of concurrency
 - Single-threaded/multi-threaded asynchronous
 - Event driven
 - Distributed
- Single-threaded/multi-threaded asynchronous, event driven, distributed concurrency
- More on distributed concurrency
- Concurrency analysis for distributed programs
- Replicated systems: Linearizability & CAP Theorem
- Replicated systems: Strong vs weak consistency in distributed systems

Distributed Programming

- Many flavors of concurrency
 - Single-threaded/multi-threaded asynchronous
 - Event driven
 - Distributed
- Single-threaded/multi-threaded asynchronous, event driven, distributed concurrency
- More on distributed concurrency
- Concurrency analysis for distributed programs
- Replicated systems: Linearizability & CAP Theorem
- Replicated systems: Strong vs weak consistency in distributed systems
- Replicated systems: Strong vs weak isolation in distributed systems