Concurrency

Edward Z. Yang
What is concurrency?
“It’s when you have multiple threads”

“Things execute at the same time”
Why concurrency?
Concurrency is ubiquitous
Concurrency is ubiquitous
How can programming languages make concurrent programming easier?

What abstractions are most effective?
Monitors (Java) → Locks → Deterministic Parallelism → Transactional Memory

Message Passing CSP → Locks → Deterministic Parallelism

Transaction Memory
Shared Resource

Concurrent Threads
Memory = Shared Resource

balance := 100

---

Thread #1

if (balance \geq 100) \{
    balance -= 100;
} else \{
    insufficient funds
\}

---

Thread #2

if (balance \geq 50) \{
    balance -= 50;
} else \{
    insufficient funds
\}
Memory = Shared Resource

balance := 100

Thread #1

if (balance ≥ 100) {
    balance -= 100;
} else {
    insufficient funds
}

a "critical section"
... at any moment, only one of these... processes is in its critical section

Dijkstra 1965
a “critical section”
... at any moment, only one of these... processes is in its critical section

Dijkstra 1965

I will tell you how to implement a lock.

But Dijkstra, how do I implement this?

Impressively, Dijkstra did this without assuming any sort of compare-and-swap instruction
We have locks, can we go home?

(or atomic test-and-set, or semaphores)
Where do I put locks? And what should happen if I leave out locks?

Thread #1

lock(1)
x := x + 1
y := y + 1
unlock(1)

Thread #2

tmp := x + y

Locks should be associated with the resource
How should I schedule requests?

Thread #1

lock(1)
// high priority request
unlock(1)

Thread #2

lock(1)
// only run me if buffer is empty
unlock(1)

Schedule should be user-programmable
What about composability?

Thread #1
lock(1)
lock(2)
// critical region
unlock(2)
unlock (1)

Thread #2
lock (2)
lock (1)
// critical region
unlock(1)
unlock (2)

Must know which locks are taken out and in what order.

Deadlock!
How can programming languages make concurrent programming easier?

By abstractions, of course!
Monitors (Concurrent Pascal, Java)

```java
public synchronized T
recv() throws InterruptedException {
    while (queue.isEmpty()) {
        wait();
    }
    return queue.pop();
}

public synchronized void
send(T x) {
    queue.push(x);
    notify();
}
```
Monitors (Concurrent Pascal, Java)
Monitors (Concurrent Pascal, Java)

Data in an Object

Methods

- send
- recv

I want to access the data through the send method.

Caller
Monitors (Concurrent Pascal, Java)

Data in an Object
- Methods: send, recv

Monitor

Denied, wait until the lock is free

Caller
Monitors (Concurrent Pascal, Java)

Data in an Object

Methods

Send

recv

OK, you have the lock

Caller
Monitors (Concurrent Pascal, Java)

Data in an Object

Methods

send

recv

I want to access the data through the recv method.

Caller
Monitors (Concurrent Pascal, Java)

Methods

Send
recv

Data in an Object

MONITOR

Denied, please wait until I have data for you.

Caller
Monitors (Concurrent Pascal, Java)

Producer-consumer

Producer

Producer

Producer

Buffer

Consumer

Consumer

Consumer

Consumer

must wait until something in buffer
Monitors (Concurrent Pascal, Java)

Data in an Object

Methods
- Send
- recv

MONITOR

OK, you have the lock

Caller
Monitors (Concurrent Pascal, Java)

Methods

Data in an Object

Wait! I have no data for you.

recv

send

Caller

MONITOR
Monitors (Concurrent Pascal, Java)

Data in an Object

Methods
- send
- recv

MONITOR

OK, return the lock and wait

Caller
Monitors (Concurrent Pascal, Java)

- Wake up!
- Data in an Object
- Methods: send, recv
- MONITOR
- Caller
- Caller 2
Monitors (Concurrent Pascal, Java)

Methods
- send
- recv

Data in an Object

OK, you have the lock

Caller

Here's the data!
Monitors (Concurrent Pascal, Java)

- Object
  - waiting threads
  - accessing
  - Thread 1
Monitors (Concurrent Pascal, Java)

Diagram:
- A rectangle labeled `Object`
  - An arrow labeled `wait()` from `Object` to a square labeled `Thread 1`
  - An arrow labeled `waiting threads` from `Object`

Explanation:
- `Thread 1` calls the `wait()` method on the object, causing it to enter a waiting state.
Monitors (Concurrent Pascal, Java)

- Object
- Thread 1

waiting threads
not executing
Monitors (Concurrent Pascal, Java)

- Object
- Waiting threads
  - Thread 1
  - Thread 2
  - Thread 3

wait
Monitors (Concurrent Pascal, Java)

Object

Waiting threads
not executing

Thread 1
Thread 2
Thread 3

Thread 4

notify()
Monitors (Concurrent Pascal, Java)

Object

waiting threads not executing

notify()

Thread 1

Thread 2

Thread 3

Thread 4

start executing
Monitors (Concurrent Pascal, Java)

Object

waiting threads
not executing

notifyAll()

Thread 4

Thread 2

Thread 3
Monitors (Concurrent Pascal, Java)

![Diagram of monitors and threads]

- **Object**
  - Waiting threads: not executing
  - `notifyAll()`

- **Thread 3**
- **Thread 4**
- **Thread 2**
Monitors (Concurrent Pascal, Java)

```java
public synchronized T
recv() throws InterruptedException {
    while (queue.isEmpty()) {
        wait();
    }
    return queue.pop();
}

public synchronized void
send(T x) {
    queue.push(x);
    notify();
}
```
Monitors in Java

- Shared memory concurrency programmer responsibility!
- Every object has a lock
- Synchronized / wait / notify / notifyAll
  (Not part of type signature)
Monitors: Nested Monitor Lockout

```java
Synchronized (a) {
  synchronized (b) {
    a.wait();
  }
}
```
Monitors: Nested Monitor Lockout

b

MONITOR

OK, return the lock and wait

Caller

Wait! I have no data for you.
Monitors: Nested Monitor Lockout

\[ \text{b: Monitor} \quad \text{a: Monitor} \]

"I still have the lock!"

Actual Caller

Red dot connecting between the two monitors.
Monitors: Nested Monitor Lockout

“Advice”:
No blocking calls in synchronized methods (use un-synchronized?)

Does not work in all situations.

I still have the lock!

Actual Caller
Monitors

- Synchronized / wait / notify

- Centralizes and hides synchronization logic associated with data
- A bit complicated / a bit inflexible
- Nested monitor lockout
- Lost notifications
- Priority must be supported (Java does not)
- Doesn't solve deadlock
Message Passing

I want to access the data through the recv method.

Denied, wait until the lock is free

eliminate shared resources
Message Passing

Actor Model
- Erlang

Communicating Sequential Processes
- Go, Haskell

Asynchronous

Sending me the data.

OK.

Synchronous
Message Passing: MVars in Haskell

one-place buffers

data MVar a

newEmptyMVar :: IO (MVar a)
newMVar :: a → IO (MVar a)
takeMVar :: MVar a → IO a
putMVar :: MVar a → a → IO a
Message Passing: MVars in Haskell

one-place buffers

putMVar

putsMVar

block

takeMVar

block

takeMVar
Message Passing: Channels with MVars

1st Value 2nd Value
Message Passing: Channels with MVars

1st Value

2nd Value
Message Passing: Channels with MVars

1st Value → 2nd Value
Message Passing

- Very simple (basis for many theoretical models; also effective programming style)

For asynchronous messaging:
- How big to maintain buffers?
- Unbounded nondeterminism

For synchronous messaging:
- Still easy to deadlock
Message passing and **immutability**

Once you send a message, **must not r/w it!**

**Erlang**
- separate address space

**Haskell**
- immutable data (but laziness!)

**Rust/C++**
- ownership types (no aliasing)
  - like juggling chainsaws
Concurrency and immutability

Immutable data definitionally not shared
Deterministic Parallelism

Concurrency ≠ Parallelism
Deterministic Parallelism

Concurrency
- Availability
- Interactivity

Parallelism
- Speed
- Distribution

Parallelism can be achieved with concurrency
Concurrency without parallelism
Deterministic Parallelism

Goal: Get speedups, w/o headaches of manual synchronization

- MapReduce (similar pipelines)
- Pure computation (sparks)
- Monotonic data structures (LVars)
Software Transactional Memory

Locks are non-compositional
(nested locking $\implies$ deadlock)

synchronized (this) {
    if (balance $\geq$ 100) {
        balance -= 100;
    } else {
        insufficient funds
    }
}

atomically {
    if (balance $\geq$ 100) {
        balance -= 100;
    } else {
        insufficient funds
    }
}

behaves as if it were an atomic transaction
Monitors (Java)

Message Passing

CSP

Actors

Deterministic Parallelism

Locks

Transactional Memory

Memory Model
Memory Model

Thread #1

\[ x = 1; \]
\[ \text{tmp1} = y; \]

Thread #2

\[ y = 1; \]
\[ \text{tmp2} = x; \]

What values of tmp1/tmp2 are allowed?
Memory Model

Thread #1
\[
x = 1; \\
tmp1 = y;
\]

Thread #2
\[
y = 1; \\
tmp2 = x;
\]

Java has a memory model

Thread #1
\[
\text{MOV} [x] \leftarrow 1 \\
\text{MOV} \ EAX \leftarrow [y]
\]

Thread #2
\[
\text{MOV} [y] \leftarrow 1 \\
\text{MOV} \ EBX \leftarrow [x]
\]

so does your processor!
Memory Model: Sequential Consistency

Without data races, should look like this!

Thread

Shared Memory

Thread

assignments/reads

(implemented by basically no one)
Memory Model: Total Store Order

SPARC, x86
Memory Model: Total Store Order

Buffers are FIFO, serve reads if possible

MFENCE flushes buffer

LOCK takes global lock, flushes buffer at end (e.g. LOCKAOD, XCHG)

Lock blocks buffered writes
Thread #1

MOV [x] ← 1
MOV EAX ← [y]

Thread #2

MOV [y] ← 1
MOV EBX ← [x]

\( \chi = \emptyset, \, \gamma = \emptyset \)
Thread #1

MOV [x] ← 1
MOV EAX ← [y]

Thread #2

MOV [y] ← 1
MOV EBX ← [x]

x = Ø, y = Ø
Thread #1

MOV [x] ← 1
MOV EAX ← [y]

x ← 1

y = Ø

Thread #2

MOV [y] ← 1
MOV EBX ← [x]

y ← 1

x = Ø

x = Ø, y = Ø
Thread #1

MOV [x] ← 1
MOV EAX ← [y]

Thread #2

MOV [y] ← 1
MOV EBX ← [x]

x = 1, y = 1
On entry the address of spinlock is in register EAX and the spinlock is unlocked iff its value is 1

| acquire: | LOCK;DEC [EAX] ; LOCK’d decrement of [EAX] |
|          | JNS enter ; branch if [EAX] was ≥ 1       |
| spin:    | CMP [EAX],0 ; test [EAX]                   |
|          | JLE spin ; branch if [EAX] was ≤ 0         |
|          | JMP acquire ; try again                    |
| enter:   | ; the critical section starts here         |
| release: | MOV [EAX] ← 1                             |

does this need to be locked?
Memory Model: ARM/POWER

- May read/write out of order

- Writes may not become visible to all threads at the same time point
Memory Model: Java

Java Language

Hardware Architecture

compilation

x = 1;
tmp1 = y;

MOV [x] ← 1
MOV EAX ← [y]
Memory Model: Java

Java Language

compilation

Hardware Architecture

\[
x = 1; \\
tmp1 = y;
\]

optimizations!

\[
\text{MOV EAX} \leftarrow [y] \\
\text{MOV [x]} \leftarrow 1
\]
Memory Model: Java

- What optimizations are acceptable?
  Notice, concurrency breaks abstraction barrier relied upon by optimization

- What program outputs can occur?
  Undefined is unacceptable if it leads to a security problem
Memory Model: Java

Happens-before

\[ x = 1; \]
\[ y = 2; \]
Memory Model: Java

Happens-before (Synchronizes-with)

Thread #1

- LOCK M
- $X = 1$
- UNLOCK M

Thread #2

- LOCK M
- $I = X$
- UNLOCK M
Memory Model: Java

Happens-before (Synchronizes-with)

volatile x;

Thread #1

x = 1;

Thread #2

y = x;

(Not like C volatiles!)
Memory Model: Java

Happens-before:
Program order + Synchronizes with

Data-race: two accesses on different threads (one being a write) not ordered by happens-before

If no data-races, program acts as if it is sequentially consistent (if not? don’t ask!)
\[
\begin{align*}
x &= y = \emptyset \\
x &= 1 \\
tmp1 &= y \\
y &= 1 \\
tmp2 &= x \\
i &= \emptyset, j &= \emptyset \quad \text{OK}
\end{align*}
\]
Double-checked locking

private static Something instance = null;

public Something getInstance() {
    if (instance == null) {
        synchronized (this) {
            if (instance == null) {
                instance = new Something();
            }
        }
    }
    return instance;
}
Double-checked locking

Thread 1:
- READ instance
- LOCK
- READ instance
- INITIALIZE Something
- WRITE instance
- UNLOCK

Thread 2:
- READ instance
- data race
- INITIALIZE Something
- WRITE instance
- READ instance
- LOCK
- READ instance
- INITIALIZE Something
- WRITE instance
- UNLOCK

Synchronizes with
Double-checked locking

thread 1

READ instance
LOCK
READ instance
WRITE instance
INITIALIZE Something
UNLOCK reorder!

thread 2
reads the uninitialized object
READ instance
Recall

How can programming languages make concurrent programming easier?

What abstractions are most effective?

Wednesday: Forward-looking research ideas
Extra examples

- Implementing monitors w/ MVars