Why Events Are A Bad Idea (for high-concurrency servers)

**Paper:** Rob von Behren, Jeremy Condit and Eric Brewer

**Presentation:** David Baldwin
Overview

- Brief Recap: Threads vs. Events
  - Review relevant details from prior classes
- Limitations and Criticisms of Threads
- Possible Ways to Improve Limitations
- Evaluation of Improvements
  - Via simple web server implementation
- Discussion
Threads

- Useful tool for concurrency
  - Allow the programmer to straightforwardly adapt serial program
  - Provides primitives for shared memory access
  - Can greatly improve performance when used correctly

- But...
  - Deadlock and race conditions
  - Tricky to get right in complex cases
  - More on this later....
Threads vs. Events

- Events are advantageous for high concurrency (Ousterhout)
  - No context switching between blocking calls
  - No overhead from locking
  - Deadlock easily avoidable
    - Though circular events might create the same effect
  - More memory efficient
Threads vs. Events

- Events are advantageous for high concurrency (Ousterhout)
  - Fine tuning of scheduling based on the application
  - Ease of debugging, subjective ease of programming vs. threads
Threads vs. Events

- A bit of a false dichotomy
  - Duality of threads and events (Aday et al., 2002; Lauer et al, 1978)
    - Threads and events have the same 'blocking graph'
  - Events are programmatically more explicit about state, blocking calls and scheduling
    - Thread libraries handle this for the programmer
Threads vs. Events

- Threads evolved from processes
  - Obvious benefits: shared memory, comparably lightweight, etc.
  - Original intent likely not thread-per-request web servers!

- The big questions of this paper:
  - Is the thread abstraction intrinsically flawed?
  - Can theoretical duality hold up in practice?
  - What accounts for the differences between thread and event performance and consequences?
More Critiques of Threads

1. Threads have restricted control flow
   - Unnatural for fan-in/out and pub/sub patterns

2. Synchronization is heavyweight
   - Events get 'free' locking
More Critiques of Threads

- **3. Stack management**
  - Events unwind stacks upon completion

- **4. Scheduling**
  - Events can be explicit, threads are generic and libraries don’t usually provide facilities to alter scheduling
  - Events can exploit things like code locality by batch processing events

- **5. Events minimize live state**
  - Obvious memory advantages
More Critiques of Threads

6. Many thread based implementation of concurrent programs have not yielded large performance gains

- Context switching can be $\mathcal{O}(n)$ w.r.t. number of threads
- High cost of context switch
  - Save registers
  - Kernel crossings
Solutions and Responses

1. Threads have restricted control flow
   - Most programs follow patterns that are more natural in threaded code
   - Pipelines, call/return, parallel calls
     - Most high concurrency applications use these
Threads vs. Events

- Events introduce their own complexity
  - Stack ripping

```java
run()
{
    foo = readData();
    bar = callWebService(foo);
    writeData(bar);
}

run()
{
    foo = readData();
    bar = callWebService(foo);
    writeData(bar);
}

readDataEvent()
{
    readDataAsync(callWebServiceEvent);
}

callWebServiceEvent(foo)
{
    callWebServiceAsync(foo,writeDataEvent);
}

callWebServiceEvent(foo)
{
    callWebServiceAsync(foo,writeDataEvent);
}

writeDataEvent(bar)
{
    writeDataAsync(bar);
}

/* ... and glue code (queue, etc); need continuations for local state*/
Solutions and Responses

● 2. Synchronization is heavyweight
  ♦ To some extent this is because events don't preempt each other
  ♦ Possibly true of any system
  ♦ Cooperative multithreading introduces some control over context switching
Solutions and Responses

● 3. Stack management
  ◆ Dynamic stack growth – upper bound space per function call
  ◆ Live state management – analyze which variables can be discarded from the stack

● 4. Scheduling
  ◆ Cooperatively scheduled threads can use same techniques as events
  ◆ Underlying libraries might be able to exploit same tricks as event systems
Solutions and Responses

5. Events minimize live state
   - This can be a disadvantage!
   - Threads naturally handle exceptions and other control flow issues
   - Heap allocated events have trouble here
     - Garbage collection difficult
     - Case: Inktomi Traffic Server – an event based system with memory leaks
6. Many thread based implementation of concurrent programs with threads have not yielded large performance gains

- Context switching can be $O(n)$ w.r.t. number of threads
- This is an implementation issue that can be optimized away
  - $10^5$ threads possible on SEDA benchmark
- Using user threads instead of kernel
Performance Analysis

- Authors developed cooperative threading based web server - Knot
  - Optimized context switching
  - Blocking I/O translated to asynchronous
  - Threading library and server ~6000 lines
- Compared to SEDA based web server (Haboob)
Haboob and Knot-A accept new connections

Haboob crashes at ~16000: out at memory
Conclusion

• Back to the big question:
  ◆ Is the thread abstraction intrinsically flawed?
    ◆ No, we can achieve similar performance
  ◆ What accounts for prior differences?
    ◆ Library implementation
    ◆ Program implementation
    ◆ Hardware

• But to show this, the authors make threads look more like events
And the winner is...

- Code complexity cuts both ways
  - Events require stack ripping, explicit wiring between events; more up front coding work
  - Both patterns can introduce subtle bugs
- Paper complementary to existing research
  - Title a bit misleading
  - “Why Threads and Events are Possibly Applicable (depending on your situation)” doesn't have the same ring to it....
And the winner is...

- No magic bullets for concurrency
  - Threads and events are tools and abstractions that can be used and abused
  - Performance is going to be based largely on your implementation (and your library)
  - Large scale systems are likely to be some hybrid
  - Efficient concurrent systems can be hard to program regardless of the paradigm
- The sad corollary:
  - We still have to think when writing code
Additional Thoughts

- Concurrent and distributed systems
  - A concurrent event code can be adapted to a distributed system more easily than a threaded one