Why Events Are A Bad Idea
(for high-concurrency servers)
Rob von Behren, Jeremy Condit and Eric Brewer

Presenter: Harry Xingzhi Pan
Jan 24, 2011
Goal

- Threads have all the power of events
- Actually even better for high-concurrency servers
- Simpler and more natural programming style
Ways to the Goal

• **Threads vs. Events**

• Threads are better for high-concurrency servers

• Compiler support

  • Current implementations not good enough

• Validation

• Related work
Duality
(Lauer and Needham)

• Events
  • handlers
  • accepted events
  • Send/WaitReply
  • SendReply
  • wait for msgs

• Threads
  • monitors
  • exported funcs
  • call or fork/join
  • return
  • wait for c.var
Duality cont.  
(Lauer and Needham)

- Mapping
- Equivalent performance
  - Threads shouldn’t be worse
- Depends on target application
  - And threads are better for high-concurrency servers
“Problems” with Threads

Performance

• **Bad performance**

• Poor implementation
  • $O(n)$ in number of threads
  • high context switch overhead
  • kernel crossing

• Modified GNU Pth performs well
  • removed most $O(n)$ operations
“Problems” with Threads
Performance cont.
“Problems” with Threads

Control Flow

- Restrictive Control Flow
  - linear

- Complicated control flow pattern
  - rare
  - difficult to use well
  - call/return, parallel calls, pipelines
“Problems” with Threads

Synchronization

• **Heavy weight synchronization**

• **Free synchronization for event systems**

• Really due to cooperative multitasking, not events
  • free for uniprocessors only
  • like disabling interrupts
  • high-concurrency servers have multi processors
“Problems” with Threads

State Management

• ineffective to manage live state
• trade-off between risking stack overflow and wasting space
• event systems unwind stack
• Dynamic stack growth (later)
• Event systems require programmers to manage live state by hand
• Thread systems have automatic state management
“Problems” with Threads

Scheduling

• can’t optimize scheduling decisions
• runtime system too generic
• event systems can schedule at application level

• The same tricks can be applied to thread systems according to duality
Ways to the Goal

• Threads vs. Events

• **Threads are better for high-concurrency servers**

• Compiler support
  • Current implementations not good enough

• Validation

• Related work
Threads are better

- Two observations about servers
  - concurrent requests, largely independent
  - sequential handling of each request
Threads are better

Control Flow

• Events obfuscate app control flow
  • Send/AwaitReply vs call/return
  • manually save/restore live state
  • unexpected msg arrivals cause subtle race conditions & logic errors
Threads are better
Control Flow cont.

• Thread systems group calls with returns
  • easier to understand cause/effect
• Run-time call stack encapsulates live state
Threads are better
Exception Handling & State Lifetime

- simpler cleaning up task state
- on the contrary task state in event systems is typically in heap
- GC is inappropriate for high performance systems
Threads are better

Existing systems

• threads in existing event-driven systems

• most complex parts: such as recovery

• also applications that don’t need high concurrency: threads are simpler
Threads are better
How about fixing events?

- duplicate threads
- E.g., continuations around blocking calls in Adya et al. [1]
Ways to the Goal

• Threads vs. Events

• Threads are better for high-concurrency servers

• Compiler support
  • Current implementations not good enough

• Validation

• Related work
Compiler Support

Dynamic Stack Growth

- adjust stack size at runtime
- analyze upper bound of stack size when calling
- determine which call sites may require stack growth
Compiler Support
Live State Management

- purge unnecessary state from stack
  - temp vars before calling subroutines
  - tail calls
- warn programmer of large amounts of state across a blocking call
Compiler Support

Synchronization

• compile-time analysis
  • warn programmers
• tractable whole-program analyses
• nesC/TinyOS
  • call graph
• calls within an atomic section cannot yield or block
Ways to the Goal

• Threads vs. Events

• Threads are better for high-concurrency servers

• Compiler support

  • Current implementations not good enough

• Validation

• Related work
Validation

• simple user-level cooperative threading package

• minimalist context switching

• internal translates blocking I/O to asynchronous requests
Validation cont.

- **Knot vs Haboob (SEDA)**

![Graph comparing Knot-C, Knot-A, and Haboob](image-url)
Ways to the Goal

• Threads vs. Events

• Threads are better for high-concurrency servers

• Compiler support
  • Current implementations not good enough

• Validation

• Related work
Related work

- Adya et al. [1]
- Ousterhout [11]
- Welsh, Culler and Brewer [17]
- Lauer and Needham [10]
Questions
&
Thank you!