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

• Background
• Criticism of Threads
• Defense of Threads
• Compiler Support for Threads
• Evaluation
• Conclusion
Background: Events

- Small, static number of control flows
- One control flow per CPU
- No preemption of event handlers
- Events/messages for communication and synchronization
- Essentially a “message-oriented system”
- I/O requires stack ripping
Background: Threads

• Many and variable number of control flows
• Communication via shared data
• Synchronized access to shared data
• Essentially, a “procedure-oriented system”
• I/O simplified via thread blocking
Background: Opinion 1

Events are better!

- Synchronization is free, via cooperative multitasking
- No space overhead for stacks during blocking
- Application level scheduling enables performance optimizations
- More flexible control flow
Background: Opinion 2

Events and threads are the same!

• Based on Lauer & Needham study
• Events are message-oriented systems
• Threads are procedure-oriented systems
• These are duals, quit arguing!
Background: Opinion 3

Threads are better!

• If implemented well, threads can have the same good properties as events
• Compiler improvements can eliminate historical drawbacks with threads
• Threads are more natural than events, and therefore are better!
Overview

• Background
• Criticism of Threads
• Defense of Threads
• Compiler Support for Threads
• Evaluation
• Conclusion
Criticism of Threads

1- Poor performance for high concurrency systems

Response:

• Due to poor thread implementations
  • Operations $O(n)$ in the number of threads
  • Context switch overhead and kernel crossing

✔ Modified GNU Pth user-level threads
✔ Removed $O(n)$ operations from the scheduler
✔ Repeated the SEDA benchmarks and did better!
Criticism of Threads

2- Restrictive control flow
   - Encourages programmers to think too linearly

Response:
✓ Complex patterns not used in practice
   - they are difficult and error prone
✓ Threads express common patterns more naturally:
   - call/return
   - parallel calls
   - pipelines
Criticism of Threads

3- Heavyweight synchronization
   - synchronization is free with events

Response:
 ✓ Adya et al show that this advantage is due to cooperative multitasking, not events
 ✓ It is free only with a uniprocessor
   • High-concurrency servers have multiple CPUs
Criticism of Threads

4- Inefficient live state management:
• Unavoidable tradeoff between risking stack overflow or wasting space on large stacks
• Event systems unwind the thread stack after each event handler, saving only what is necessary

Response:
✓ Future compiler optimizations can reduce stack requirements
Criticism of Threads

5- Scheduling: can’t optimize scheduling decisions
  • Runtime system is too generic in thread-based
  • Event systems can schedule at application level (better scheduling, and code locality)

Response:
✓ Lauer and Needham argue that Threads and events are equivalent
Overview

- Background
- Criticism of Threads
- Defense of Threads
- Compiler Support for Threads
- Evaluation
- Conclusion
Defense of Threads

Observations about modern servers:

1- The concurrency is because of the largely independent concurrent requests

2- The code that handles each code is usually sequential

➤ Threads provide a better programming abstraction for servers.
Defense of Threads

Control Flow:
Event-based systems: obfuscate the control flow
1- Programmers manually match call/return pairs
2- Programmers manually save and restore live state
   “Stack Ripping”

Threads: express control flow in a natural manner
1- Group calls with returns, one-to-one
2- Encapsulate live state in a stack: easier debugging!
Defense of Threads

Exception Handling and state Lifetime

- Cleaning up state is simple, its all on the stack
- With events, state is typically heap allocated and difficult to trace
- Garbage collection is inappropriate for high performance systems
Defense of Threads

Existing Systems:
- Often, event-based systems use threads in the complex parts of the code: Ninja system
- Often applications with no concurrency also use threads for simplicity: FTP server in Harvest

Just Fix Events:
- Fixing the problems with events is equivalent to switching to threads
  - Just use threads
Overview

• Background
• Criticism of Threads
• Defense of Threads
• **Compiler Support for Threads**
• Evaluation
• Conclusion
Compiler Support for Threads

Hand waiving!
Interesting example based on tail call optimization
Overview

- Background
- Criticism of Threads
- Defense of Threads
- Compiler Support for Threads
- Evaluation
- Conclusion
Evaluation

- Designed and implemented a cooperative user-level threads package for Linux (5000 lines)
- Used it to write a web server, Knot. (700 lines)
- Compared the performance of Knot with Haboob (SEDA’s event-driven web server)
- Two different scheduling policies for Knot:
  1-favors processing of current connections
  2-favors processing of accepting connections
Web server bandwidth versus number of simultaneous clients.
Overview

• Background
• Criticism of Threads
• Defense of Threads
• Compiler Support for Threads
• Evaluation
• Conclusion
Conclusion

Threads have similar or better performance to events in high concurrency systems.

Compiler improvements could eliminate the problems associated with threads.