Why Threads Are A Bad Idea
(for most purposes)

John Ousterhout
Sun Microsystems Laboratories

john.ousterhout@eng.sun.com
http://www.sunlabs.com/~ouster

Introduction

* Threads:
  - Grew up in OS world (processes).
  - Evolved into user-level tool.
  - Proposed as solution for a variety of problems.
  - Every programmer should be a threads programmer?
* Problem: threads are very hard to program.
* Alternative: events.
* Claims:
  - For most purposes proposed for threads, events are better.
  - Threads should be used only when true CPU concurrency is needed.

What Are Threads?

* General-purpose solution for managing concurrency.
* Multiple independent execution streams.
* Shared state.
* Pre-emptive scheduling.
* Synchronization (e.g. locks, conditions).

What Are Threads Used For?

* Operating systems: one kernel thread for each user process.
* Scientific applications: one thread per CPU (solve problems more quickly).
* Distributed systems: process requests concurrently (overlap I/Os).
* GUIs:
  - Threads correspond to user actions; can service display during long-running computations.
  - Multimedia, animations.

What's Wrong With Threads?

* Too hard for most programmers to use.
* Even for experts, development is painful.

Why Threads Are Hard

* Synchronization:
  - Must coordinate access to shared data with locks.
  - Forget a lock? Corrupted data.
* Deadlock:
  - Circular dependencies among locks.
  - Each process waits for some other process: system hangs.

Why Threads Are A Bad Idea September 28, 1995, slide 2
Why Threads Are A Bad Idea September 28, 1995, slide 3
Why Threads Are A Bad Idea September 28, 1995, slide 4
Why Threads Are A Bad Idea September 28, 1995, slide 5
Why Threads Are A Bad Idea September 28, 1995, slide 6
Why Threads Are Hard, cont'd

- Hard to debug: data dependencies, timing dependencies.
- Threads break abstraction: can't design modules independently.
- Callbacks don't work with locks.
Events vs. Threads, cont'd

- Events faster than threads on single CPU:
  - No locking overheads.
  - No context switching.
- Events more portable than threads.
- Threads provide true concurrency:
  - Can have long-running stateful handlers without freezes.
  - Scalable performance on multiple CPUs.

Should You Abandon Threads?

- No: important for high-end servers (e.g. databases).
- But, avoid threads wherever possible:
  - Use events, not threads, for GUIs, distributed systems, low-end servers.
  - Only use threads where true CPU concurrency is needed.
  - Where threads needed, isolate usage in threaded application kernel; keep most of code single-threaded.

Conclusions

- Concurrency is fundamentally hard; avoid whenever possible.
- Threads more powerful than events, but power is rarely needed.
- Threads much harder to program than events; for experts only.
- Use events as primary development tool (both GUIs and distributed systems).
- Use threads only for performance-critical kernels.