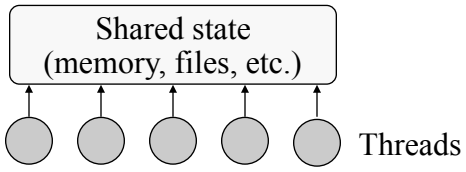
Why Threads Are A Bad Idea (for most purposes)

Taken from work of
John Ousterhout

Introduction

- Threads
  - Grew up in OS world (processes)
  - Every programmer should be a threads programmer?
- Problem: threads are very hard to program.
- Alternative: events
- Claims:
  - For most purposes, events are better.
  - Threads should be used only when true CPU concurrency is needed

What Are Threads?

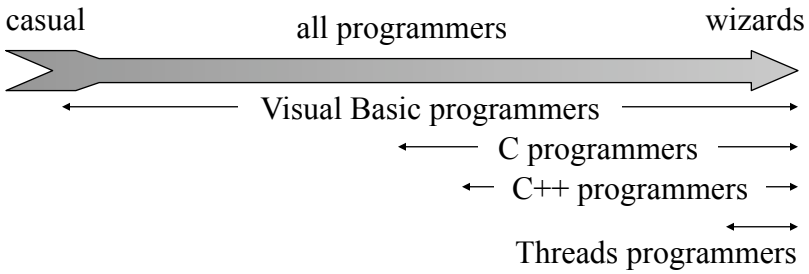


- General-purpose solution for managing concurrency
- Multiple independent execution streams
- Shared state
- Pre-emptive scheduling
- Synchronization (e.g. locks, conditions)

What Are Threads Used For?

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

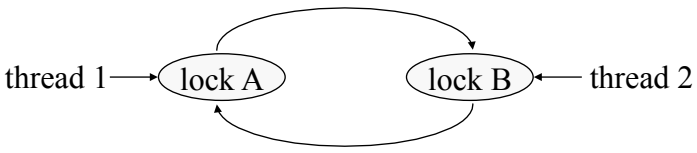
What's Wrong With Threads?



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

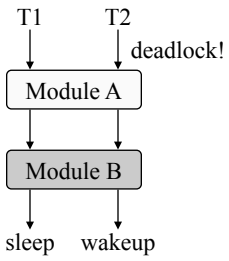
Why Threads Are Hard

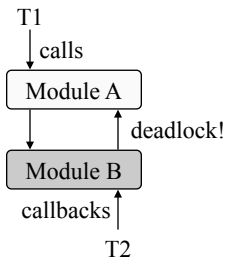
- Synchronization:
  - Must coordinate access to shared data with locks
  - Forget a lock? Corrupted data
- Deadlock:
  - Circular dependencies among locks
  - Each process waits for some other process: system hangs



Why Threads Are Hard, cont'd

- Hard to debug: data and timing dependencies
- Threads break abstraction: can't design modules independently
- Callbacks don't work with locks



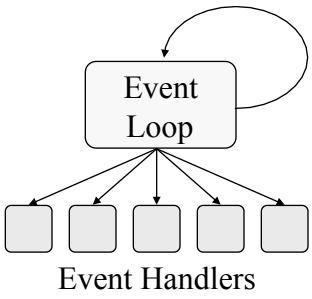


Why Threads Are Hard, cont'd

- Achieving good performance is hard:
  - Simple locking yields low concurrency
  - Fine-grain locking reduces performance
  - OSes limit performance (context switches)
- Threads not well supported:
  - Hard to port threaded code (PCs? Macs?)
  - Standard libraries not thread-safe
  - Kernel calls, window systems not multi-threaded
  - Few debugging tools (LockLint, debuggers?)

Event-Driven Programming

- One execution stream: no CPU concurrency
- Register interest in events (callbacks)
- Event loop waits for events, invokes handlers
- No preemption of event handlers
- Handlers generally short-lived



What Are Events Used For?

- Mostly GUIs:
  - One handler for each event (press button)
  - Handler implements behavior (undo, delete file, etc.)
  - See: Beeper.java
- Distributed systems:
  - One handler for each source of input (i.e., socket)
  - Handler processes incoming request, sends response
  - Event-driven I/O for I/O overlap
  - See: MultiPortEcho.java & MultiEchoClient.java

Problems With Events

- Long-running handlers make application non-responsive
  - Fork off subprocesses for long-running things (e.g., multimedia), use events to find out when done
  - Break up handlers (e.g. event-driven I/O)
- Can't maintain local state across events (handler must return)
- No CPU concurrency (not suitable for sci. apps)

Events vs. Threads

- Events avoid concurrency as much as possible:
  - Easy to get started with events: no concurrency, no preemption, no synchronization, no deadlock
  - Use complicated techniques only for unusual cases
  - With threads, even the simplest application faces the full complexity

Events vs. Threads

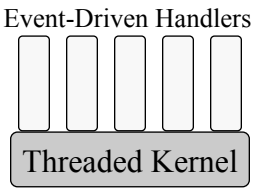
- Debugging easier with events:
  - Timing dependencies only related to events, not to internal scheduling
  - Problems easier to track down: slow response to button vs. corrupted memory

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
- 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



Summary

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