CSE 60641: Operating Systems

- Next topic: CPU (Process/threads/scheduling, synchronization and deadlocks)
  - Why threads are a bad idea (for most purposes). J Ousterhout - Keynote at the 1996 Usenix Annual Technical
  - Why Events Are A Bad Idea (for high-concurrency servers) Rob von Behren, Jeremy Condit and Eric Brewer, HotOS IX
  - Event-driven Programming is Not the Opposite of Threaded Programming, Atul Adya, Jon Howell, Marvin Theimer, William J. Bolosky, John R. Douceur. USENIX '02

What is the problem being addressed?

- Programming for concurrency
  - Why do we worry about this?
- Threads are a popular abstraction
  - Light weight (as compared to tasks)
  - They retain stack
- Event based program
  - When a event happens, the system calls a event dispatcher which calls the appropriate service routine.
  - Easy to program
  - The service routine does not know where it came from (no stack). Hence the need to manually manage stacks

Ousterhout

- Author of TCL programming language
- Threads are too hard for most programmers. Even for experts, development is painful.
  - Synchronization, deadlocks
  - Hard to debug: data dependencies, timing dependencies
  - Break abstraction: cannot design modules independently
  - Callbacks don't work with locks
  - Achieving good performance is hard
    - Simple locking yields low concurrency
    - Fine-grain locking increases complexity, reduces performance
  - Threads not well-supported (circa 1995)

Event-driven programming

- One execution stream: no CPU concurrency
- Long-running handlers make applications nonresponsive
  - Fork off sub-processes for long running things, use events to find out when done.
  - Break up handlers (event-driven I/O)
- Can't maintain local state across events (handler must return)
  - Manual stack maintenance
    - Stack is used to maintain local data, state and return
    - Events – all local state is lost after scheduling an event

Conclusions

- Concurrency is fundamentally hard; avoid whenever possible (most of recent machines are multi-core)
- 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

Adya et al.

- Argue that things are little more complicated
- Task management: preemptive, serial and cooperative (yields control at well defined points)
- Stack management: manual, automatic

- I/O management: synchronous, asynchronous
- Conflict management:
  - Pessimistic – locks
  - Optimistic – use speculation; if conflict, roll back and retry
- Data partitioning

- Event-driven: cooperative task management and manual stack management
- Threaded: preemptive and automatic stack mgmt
- Sweet spot: cooperative task management, automatic stack management
- Stack ripping: event driven code
  - Function scoping: two or more functions represent a single conceptual function
    1. Read network event, schedule disk read
    2. Process read event, schedule write event
  - Automatic variables: local (stack variables) need to be moved into heap to survive across yield points

- Control structures: entry point must be a language function
- Debugging stack: call stack must be manually recovered, manual optimization of tail calls

- The authors show a hybrid approach: manual calling automatic, automatic calling manual

Brewer et al. – Events are a bad idea

- Weaknesses of threads are artifacts of poor threads
- Compiler support for thread systems
- Applicable for high concurrency servers

- Problems with threads:
  - Performance: Many attempts to use threads for high concurrency have not performed well
    - Poor implementation. O(n) components
  - Control flows: encourages programmers to think linearly
    - Robust systems need acknowledgments, even in events

- Synchronization: mechanisms are too heavy-weight
  - Cooperative threads
- State management: Thread stacks are an ineffective way to manage live state
  - Dynamic thread stack size management
- Scheduling: Threads treat processors are virtual – runtime is too generic and prevents it from making optimal scheduling decisions. Events can perform shortest remaining completion time scheduling, favor certain request streams to maintain locality etc.
  - Duality argument