On the Duality of Operating System Structure

- Hugh C. Lauer
  - Adjunct Prof., Worcester Polytechnic Institute
  - Xerox, Apollo Computer, Mitsubishi Electronic Research Lab, etc.
  - Founded a number of businesses:
    - Real-Time Visualization unit of Mitsubishi Electric Research Labs (MERL)

- Roger M. Needham
  - Prof., Cambridge University
  - Microsoft Research, Cambridge Lab
  - Kerberose, Needham-Schroeder security protocol, and key exchange systems
Message vs Procedure oriented system (i.e. Events vs Threads)

- Are they really the same thing?
- Lauer and Needham show
  - 1) two models are duals
    - Mapping exists from one model to other
  - 2) dual programs are logically identical
    - Textually similar
  - 3) dual programs have identical performance
    - Measured in exec time, compute overhead, and queue/wait times
Message-oriented system (Event)

- Small, static # of process
- Explicit messaging
- Limited data sharing in memory
- Identification of address space or context with processes
Message-oriented system

- **Characteristics**
  - Queuing for congested resource
  - Data structure passed by reference
    (no concurrent access)
  - Peripheral devices treated as processes
  - Priority of process statically determined
  - No global naming scheme is useful
Message-oriented system

- Calls:
  - SendMessage, AwaitReply
  - SendReply
  - WaitForMessage

- Characteristics
  - Synchronization via message queues
  - No sharing of data structures/address space
  - Number of processes static
Message-oriented system

- Canonical model
  ```
  begin
    Do forever
      WaitForMessages
      case port
        port 1: ...;
        port 2: ...; SendReply; ...;
      end case
    end loop
  end
  ```
Procedure-Oriented System (Thread)

- Large # of small processes
- Rapidly changing # of processes
- Communication using direct sharing and interlocking of data
- Identification of context of execution with function being executed
Process-oriented system

- Characteristics
  - Synchronization and congestion control associates with waiting for locks
  - Data is shared directly and lock lasts for short period of time
  - Control of peripheral devices are in form of manipulating locks
  - Priority is dynamically determined by the execution context
  - Global naming and context is important
Process-oriented system

- Calls:
  - Fork, Join (process)
  - Wait, Signal (condition variables)

- Characteristics
  - Synchronization via locks/monitors
  - Share global address space/data structures
  - Process (thread) creation very dynamic and low-overhead
Process-oriented system

- Canonical model
  - Monitor
    - global data and state info for the process
    - proc1: ENTRY procedure
    - proc2: ENTRY procedure returns
      - begin
        - If resourceExhausted then WAIT; …;
        - RETURN result; …;
      - end
    - proc L: ENTRY procedure
      - begin
        - …; SIGNAL; …
      - end;
      - endloop;
    - initialize;
  - end
# Dual Mapping

<table>
<thead>
<tr>
<th>Event</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processes: CreateProcess</td>
<td>Monitors: NEW/START</td>
</tr>
<tr>
<td>Message channel</td>
<td>External procedure id</td>
</tr>
<tr>
<td>Message port</td>
<td>Entry procedure id</td>
</tr>
<tr>
<td>Send msg (immediate); AwaitReply</td>
<td>Simple procedure call</td>
</tr>
<tr>
<td>Send msg (delayed); AwaitReply</td>
<td>FORK; … JOIN</td>
</tr>
<tr>
<td>Send reply</td>
<td>Return from procedure</td>
</tr>
<tr>
<td>Main loop of std resource manager, wait for message stmt, case stmt</td>
<td>Monitor lock, ENTRY attribute</td>
</tr>
<tr>
<td>Arms of case statement</td>
<td>ENTRY proc declaration</td>
</tr>
<tr>
<td>Selective waiting</td>
<td>Condition vars, WAIT, SIGNAL</td>
</tr>
</tbody>
</table>
Preservation of Performance

- Performance characteristics
  - Same execution time
  - Same computational overhead
  - Same queuing and waiting times
- Do you believe they are the same?
- What is the controversy?
20 to 30 years later, still controversy!

Analyzes threads vs event-based systems, finds problems with both

Suggests trade-off: stage-driven architecture

Evaluated for two applications

- Easy to program and performs well
SEDA: An Architecture for Well-Conditioned, Scalable Internet Services (Welsh, 2001)

- **Matt Welsh**
  - Cornell undergraduate Alum (Worked on U-Net)
  - PhD from Berkeley (Worked on Ninja clustering)
  - Prof. at Harvard (Worked on sensor networks)
  - Currently at Google

- **David Culler**
  - Faculty at UC Berkeley

- **Eric Brewer**
  - Faculty at UC Berkeley (currently on leave at Google)
A traditional “process” is an address space and a thread of control.

Now add multiple thread of controls

- Share address space
- Individual program counters and stacks

Same as multiple processes sharing an address space.
Thread Switching

- To switch from thread T1 to T2:
  - Thread T1 saves its registers (including pc) on its stack
  - Scheduler remembers T1’s stack pointer
  - Scheduler restores T2’ stack pointer
  - T2 restores its registers
  - T2 resumes
Thread Scheduler

- Maintains the stack pointer of each thread
- Decides what thread to run next
  - E.g., based on priority or resource usage
- Decides when to pre-empt a running thread
  - E.g., based on a timer
- Needs to deal with multiple cores
  - Didn’t use to be the case
- “fork” creates a new thread
Synchronization Primitives

- **Semaphores**
  - $P(S)$: block if semaphore is “taken”
  - $V(S)$: release semaphore

- **Monitors:**
  - Only one thread active in a module at a time
  - Threads can block waiting for some condition using the WAIT primitive
  - Threads need to signal using NOTIFY or BROADCAST
Uses of threads

- To exploit CPU parallelism
  - Run two threads at once in the same program

- To exploit I/O parallelism
  - Run I/O while computing, or do multiple I/O
  - I/O may be “remote procedure call”

- For program structuring
  - E.g., timers
Common Problems

- Priority Inversion
  - High priority thread waits for low priority thread
  - Solution: temporarily push priority up (rejected??)

- Deadlock
  - X waits for Y, Y waits for X

- Incorrect Synchronization
  - Forgetting to release a lock

- Failed “fork”

- Tuning
  - E.g. timer values in different environment
What is an Event?

- An object queued for some module

- Operations:
  - `create_event_queue(handler) → EQ`
  - `enqueue_event(EQ, event-object)`
    - Invokes, eventually, `handler(event-object)`

- Handler is *not* allowed to block
  - Blocking could cause entire system to block
  - But page faults, garbage collection, …
(Also common in telecommunications industry, where it’s called ‘workflow programming’).
Decides which event queue to handle next.
- Based on priority, CPU usage, etc.

Never pre-empts event handlers!
- No need for stack / event handler

May need to deal with multiple CPUs
Synchronization?

- Handlers cannot block → no synchronization
- Handlers should not share memory
  - At least not in parallel
- All communication through events
Uses of Events

- CPU parallelism
  - Different handlers on different CPUs

- I/O concurrency
  - Completion of I/O signaled by event
  - Other activities can happen in parallel

- Program structuring
  - Not so great...
  - But can use multiple programming languages!
Common Problems

- Priority inversion, deadlock, etc. much the same with events
- Stack ripping
Threaded Server Throughput
Event-driven Server Throughput
Threads vs. Events

- Events-based systems use fewer resources
  - Better performance (particularly scalability)

- Event-based systems harder to program
  - Have to avoid blocking at all cost
  - Block-structured programming doesn’t work
  - How to do exception handling?

- In both cases, tuning is difficult
**SEDA**

- Mixture of models of threads and events
- Events, queues, and “pools of event handling threads”.
- Pools can be dynamically adjusted as need arises.
SEDA Stage
Best of both worlds

- Ease of programming of threads
  - Or even better

- Performance of events
  - Or even better

- Did we achieve Lauer and Needham’s vision with SEDA?
Next Time

- Read and write review:

- MP1 – due next Friday
  - Let us know how you are doing; if need help

- Presentations
  - Sign up to present

- Project Proposal due tomorrow
  - Also, talk to faculty and email and talk to me

- Check website for updated schedule
Read and write review:
