SEDASDA
How We Got Here

On Tuesday we were talking about Multics and Unix.

• Fast forward 30-35 years.

How has the OS (e.g., Linux) changed?

• Some of Multics crept back into Unix.
  Mapped files, shared libraries, shared memory, DLLs

• Added networking cc 1980.
  Berkeley sockets and select (2)

• Added threads cc 1985-1995.
  Preemptive kernel threads

• Computers got cheap.

• Unix became the engine of the Internet.
To Begin…

• What is “resource virtualization”?
  “Existing OS strive to virtualize hardware resources in a way that is transparent to applications.”

• Examples?

• Who is it bad for?

• Why is it bad for them?

• Why do we have it if it is so bad?
  “Multiprogramming mindset”

• What has changed?
Internet Services and Overload

“Internet services are not designed to share the machine with other applications.”

Servers face overload conditions due to “flash crowds”, e.g., the “slashdot effect”.

A new *ility to optimize: stability.

Goal: “graceful degradation”.

Servers want “application-specific adaptation in the face of overload.”

How can a server know that it is in overload?

How to adapt? Why is it application-specific?
**Threads and Events**

Hypothesis: threads hide too much useful info from the app.
- They are “virtualization”, and virtualization is bad.

Servers need threads for asynchrony in I/O (and for SMPs).
- Solution 1: async I/O interfaces.
- Solution 2: bound the number of threads.

If we had full async I/O, we could build servers with one thread per processor.
- Flash SPED structure: Single Process Event Driven
- Can hide sync I/O interfaces behind objects that look asynchronous but use bounded concurrency internally.
SEDAM

Separate server apps into modules, objects, or “stages”.

• One or more internal threads
• processes events
• single input queue

How is this different from processes with pipes? Monitors?

Resource controllers “introspect” by monitoring input/output behavior of stages, and adapting.

• Load shedding
• Vary batching or concurrency
• Arbitrary application-specific responses
SEDA and the OS Model

What does the app know, and when does it know it?

“The basic resource management mechanisms provided by commodity OS, subject to application-level control, are adequate for the needs of Internet services.”

“Threads are good, but restrict your use of them”?

Is the problem with threads fundamental, or is it an implementation problem?

How could we “fix” the OS?
SPED and SEDA

SEDA is “a middle ground between threaded and event-driven designs”?

Claims:

• Handles varying service demand times at stages
• Easy to debug
• Easy to synchronize
Also see...

Matt Welsh’s slides are posted on the SEDA/Sandstorm page at Berkeley (linked through the readings page on the course web).