

Graduate Operating Systems

Spring 2022

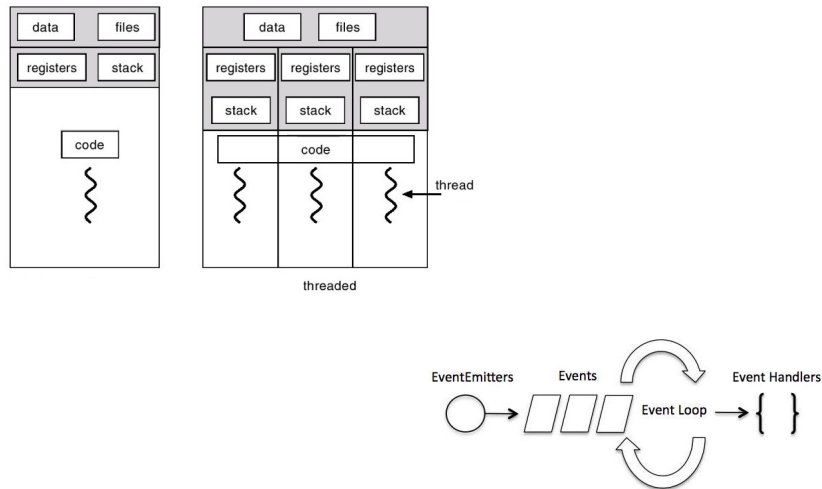
1

Today's Papers

- **[15]** Rob von Behren, Jeremy Condit, and Eric Brewer, "Why Events are a Bad Idea (for high-concurrency servers)", Workshop on Hot Topics in Operating Systems, 2003.
- **[16]** Matt Welsh, David Culler, and Eric Brewer, "SEDA: An Architecture for Well-Conditioned, Scalable Internet Services", ACM Symposium on Operating Systems Principles, 2001.

2

Threads vs. Events



3

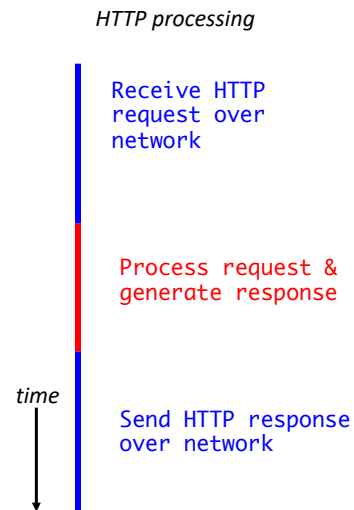
Threads vs. Events

- 1995: *Why Threads are a Bad Idea (for most purposes)*
 - John Ousterhout (UC Berkeley, Sun Labs)
- 2001: *SEDA: An Architecture for Well-Conditioned, Scalable Internet Services*
 - Staged, Event-driven Architecture
 - M. Welsh, D. Culler, and Eric Brewer (UC Berkeley)
- 2003: *Why Events are a Bad Idea (for high-concurrency servers)*
 - R. van Behren, J. Condit, Eric Brewer (UC Berkeley)

4

Background

- *How can we scale up servers to handle many simultaneous requests?*



5

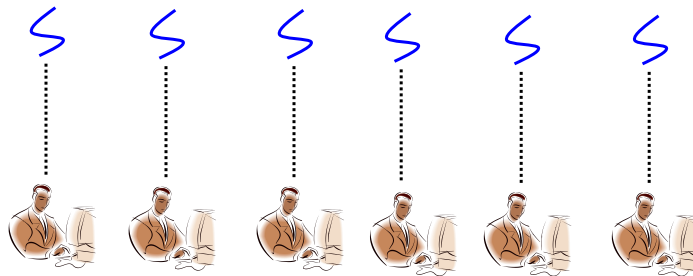
Quick Example

- *Suppose it takes 1 second for a web server to transfer a file to the client. Of this time, 10 milliseconds is dedicated to CPU processing. How many simultaneous requests do we need to keep the CPU fully utilized?*

6

Strategy #1: Thread-per-Request

- Run each web request in its own thread
 - Assuming kernel threads, blocking I/O operations only stall one thread



7

Strategy #1: Thread-per-Request

```
while (true) {  
    read request from socket  
    read requested file into buffer  
    write buffer content over socket  
    close socket  
}
```

8

Strategy #2: Event-Driven Execution

- Use a single thread for all requests
- Use non-blocking I/O
 - Replace blocking I/O with calls that return immediately
 - Program is notified about interesting I/O events
- This is philosophically similar to hardware interrupts
 - “Tell me when something interesting happens”

9

Strategy #2: Event-Driven Execution

```

while (true) {
    find sockets with active I/O
    Socket sock = getActiveSocket();
    if (sock.isReadable())
        handleReadEvent(sock);
    if (sock.isWritable())
        handleWriteEvent(sock);
}

```

- Example: GUI frameworks (*What are examples of events?*)

10

UNIX “select” System Call

```
int select(int nfd, fd_set *readfds, fd_set
*writefds, fd_set *exceptfds, struct timeval
*timeout);
```

11

Threads vs. Events

- *What is biggest problem with threads (in reading assignment)?*
- Threads:
 - Independent execution streams
 - Preemptive scheduling
 - Synchronization
 - Deadlocks
 - Debugging
 - “Threads break abstraction”
 - Getting good performance
 - OS support of threads

12

Threads vs. Events

- Events:
 - No CPU concurrency
 - Callbacks; event handlers
 - No preemption
 - Long-running handlers
 - State across handler invocations
 - Debugging
 - Overheads
 - Portability

13

Problems with Threads (Paper)

- Performance
 - Poor design; not intrinsic properties
- Control flow
 - Complicated control flow patterns are rare (call/return most common)
- Synchronization
 - Cooperative multitasking (no preemption)
- State management
 - Minimize live stack (dynamic stack growth and live state management)
- Scheduling
 - Event scheduling tricks can be applied to threads too

14

Conclusions

- Threads?
- Events?
- Future directions?
 - Many-core systems
 - Locking
 - New languages, compilers, thread packages
 - Hybrid models?

15

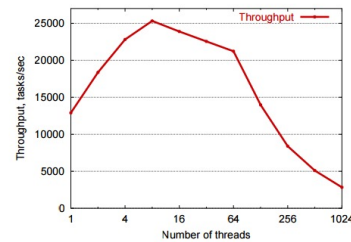
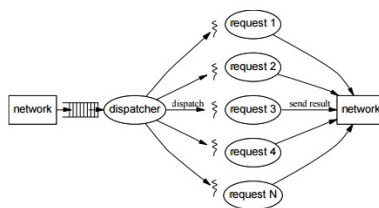
Paper “SEDA”

- “Slashdot effect”; peak load
- “Well-conditioned service”
 - Throughput: saturate with load
 - Response time: increase linearly with load
 - Graceful degradation

16

Thread-Based Concurrency

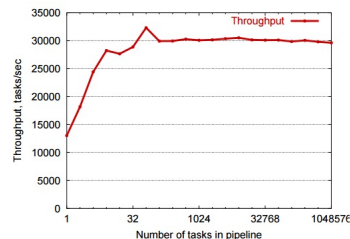
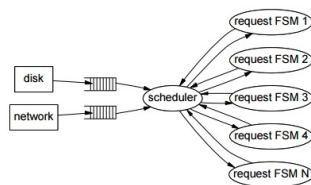
- Easy to program; high concurrency
- Overheads
- Throughput degradation (bounded thread pools)
- Latency



17

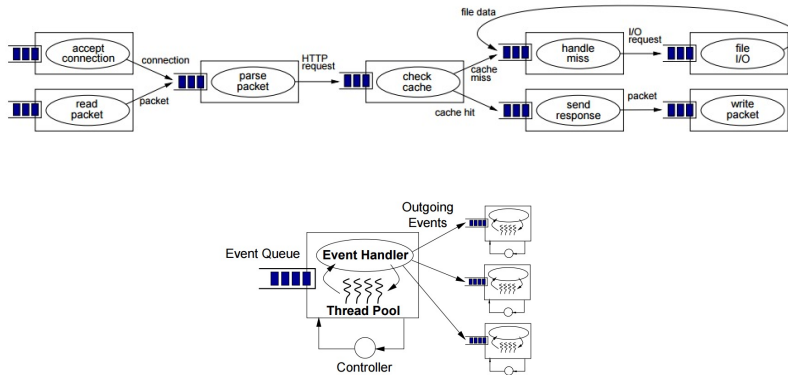
Event-Driven Concurrency

- Small number of threads (typically one per CPU); non-blocking I/O
- Robust to load
- Latency
- Scheduling decisions; load dropping



18

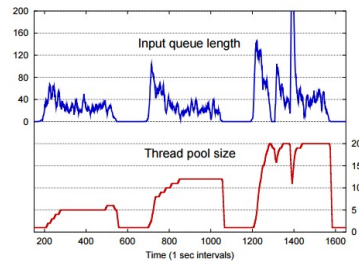
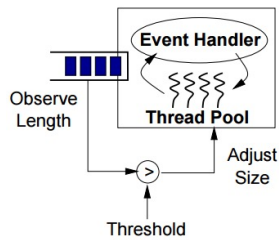
SEDA: Staged Event Driven Architecture



19

Resource Controllers

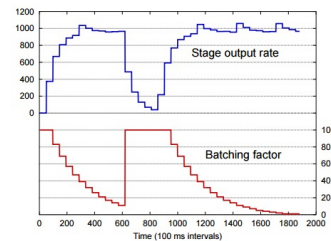
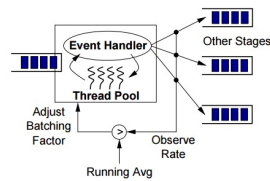
- Thread pool controller
 - Adjust number of threads executing



20

Resource Controllers

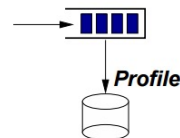
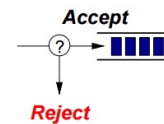
- Batching controller
 - Adjust number of events processed by each iteration of the event handler



21

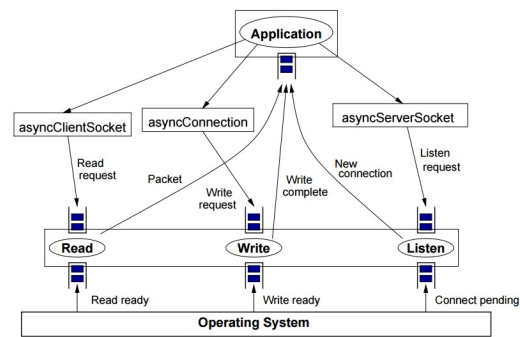
Queues

- Queues are finite
 - Enqueuing may fail
 - Block on queue -> backpressure
 - Drop rejected events -> load shedding
- Queues introduce explicit execution boundaries
 - Threads may only execute within a single stage
 - Performance isolation, modularity, independent load management
- Explicit event delivery support inspection
 - Trace flow of events through application
 - Monitor queue lengths to detect bottleneck



22

Asynchronous I/O



23

Summary & Discussion

- SEDA: Staged, Event-Driven Architecture
 - Applications consist of **connected stages each serviced by one or more threads**
 - **Dynamic resource controllers** examine and react to high load conditions and control thread usage
- Measurement and control vs. reservation
 - Mechanisms for detecting overload
 - Policies to deal with overload
- SEDA ease of programming
 - Reduced need for synchronization & race conditions
 - Separate stages for different components of application/server

24