Graduate Operating Systems

Fall 2017

Threads
Threads vs. Events

- 1995: “Why Threads are a Bad Idea (for most purposes)” [Ousterhout]
- 2003: “Why Events are a Bad Idea (for high-concurrency servers)” [van Behren et al.]

Concurrency

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

- “Thread per Request”

\[
\text{while (true) \{} \\
\quad \text{read request from socket} \\
\quad \text{read requested file into buffer} \\
\quad \text{write buffer content over socket} \\
\quad \text{close socket} \\
\\text{\}} \\
\]

- Example: Apache web server

Concurrency Strategy #2

- Single thread for all requests
- Non-blocking I/O

\[
\text{while (true) \{} \\
\quad \text{find sockets with active I/O} \\
\quad \text{Socket sock = getActiveSocket();} \\
\quad \text{if (sock.isReadable())} \\
\quad \quad \text{handleReadEvent(sock);} \\
\quad \text{if (sock.isWriteable())} \\
\quad \quad \text{handleWriteEvent(sock);} \\
\\text{\}} \\
\]

- Example: GUI frameworks
UNIX “select” System Call

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

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
Threads vs. Events

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

Threads vs. Events

• Availability of kernel threads
• Support for non-blocking I/O
• Ease/difficulty of programming
• Ease of reading/understanding code
• UNIX select: poor scalability
• Scalability of thread packages
• Resource requirements