

Managing Parallelism/Concurrency in Servers

Kai Shen

2/28/2011

CSC 258/458 - Spring 2011

1

Server

- Computer application serving (potentially many) interactive clients
 - **Parallelism**: many requests run concurrently in a server
- But unlike the parallel applications we have seen so far
 - Easily partitioned to fine-grained requests without inter-request dependencies, no need for synchronization ⇒ embarrassingly parallel
 - Highly multiprogrammed, many context switches
 - Performance (quality-of-service) and resource accounting at each request

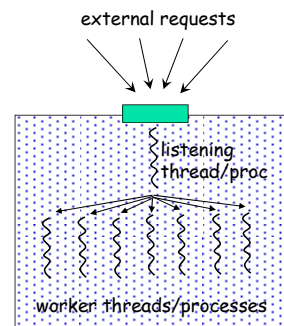
2/28/2011

CSC 258/458 - Spring 2011

2

Multi-processing vs. Multi-threading

- Multi-processing server
 - each request is served by a process (Apache).
- Multi-threading server
 - each request is served by a thread.
- Compare multi-processing server with multi-threading server
 - efficiency
 - robustness/isolation
- Pooling can be used to reduce the overhead on process/thread creation and termination



2/28/2011

CSC 258/458 - Spring 2011

3

User-level Threads

- Kernel threads
 - thread management/scheduling done by the OS kernel
- User threads
 - thread management/scheduling done at user-level
 - Benefit: efficiency, e.g., less context switching overhead
- Problem of user threads
 - oblivious to kernel events, so all threads in a process are put to wait when only one of them blocks on I/O (e.g., read())
- How to solve this problem?
 - helper (kernel) threads
 - asynchronous I/O

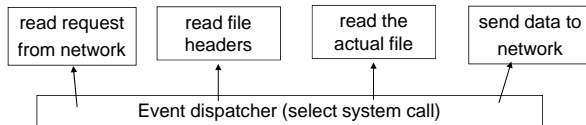
2/28/2011

CSC 258/458 - Spring 2011

4

Event-driven Servers

- Event-driven servers
 - divide request processing into stages, each of which is non-blocking
 - each stage is triggered by an event
 - the whole event controller runs in a single user thread



- Flash Web server [Pai et al., USENIX1999]
- Problems: difficult programming, hidden I/O?

2/28/2011

CSC 258/458 - Spring 2011

5

Request Scheduling

- Scheduling or request execution
 - normal multi-tasking (a task is a process/thread or a request)
- Staged resource-aware request scheduling
 - Each request execution is partitioned into stages (like in event-driven servers)
 - Each stage has particular resource needs.
 - Throttling early stage for admission control.
- SEDA [Welsh et al., SOSP2001]

2/28/2011

CSC 258/458 - Spring 2011

6

High Concurrency Threaded Server

- What is your first problem when you try to run 10,000 requests/threads concurrently in a server?
- Capriccio [von Behren et al., SOSP2003]

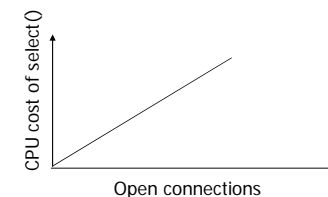
2/28/2011

CSC 258/458 - Spring 2011

7

Overhead with High Concurrency

- More frequent context switches? Cache pollution?
- Scalability of the `select()` system call [Banga et al., USENIX1998]



2/28/2011

CSC 258/458 - Spring 2011

8

Overhead with High Load Network Server

- With gigabit Ethernet:
 - 125,000,000 bytes per second for 1,500bytes/frame
 ⇒ 12us per frame
 - if an interrupt handler consumes 3us CPU, then 25% CPU processing on interrupt handling
- Soft timers [Aron and Druschel, SOSP1999]
 - NIC buffers frames; only interrupt after multiple frames arrive
 - CPU does a coarse-granularity polling

2/28/2011 CSC 258/458 - Spring 2011 9

Control the Concurrency

- How to control the execution concurrency?
 - use a request buffer queue

The diagram illustrates the flow of external requests into a server. At the top, three arrows labeled 'external requests' point to a green box representing the server's entry point. Below this, a vertical stack of boxes represents the 'request: buffer queue'. A 'listening thread or process' is shown at the top of the queue, with arrows pointing to the queue. From the bottom of the queue, multiple arrows point to several 'worker threads/processes' at the bottom of the diagram.

2/28/2011 CSC 258/458 - Spring 2011 10

Handle Server Overload

The graph plots 'success rate' on the y-axis against 'incoming request rate' on the x-axis. Two curves are shown: 'good overload management' and 'bad overload management'. The 'good' curve rises to a peak and then gradually declines. The 'bad' curve rises to a peak and then drops sharply to zero. The x-axis is labeled 'incoming request rate' and the y-axis is labeled 'success rate'.

- Overhead of server overload:
 - some requests have to be abandoned
 - when a request has to be abandoned, resources already consumed by this request is wasted
- Principle:** when abandoning a request, do so as early as possible
 - drop new requests if the buffer queue is already long

2/28/2011 CSC 258/458 - Spring 2011 11

OS Overhead for Each Request

The diagram shows the path of a request from application processes to the network interface. At the top, two ovals represent 'Application processes'. Below them are two boxes labeled 'Sockets'. A red arrow labeled 'Application access:' points from the application processes to the sockets. Below the sockets are boxes for 'TCP/UDP' and 'IP'. A blue arrow labeled 'Software interrupt:' points from the sockets to the TCP/UDP layer. Below the IP layer is a box labeled 'Interface queue'. A red arrow labeled 'Hardware interrupt:' points from the interface queue to the 'Network interface' box at the bottom.

2/28/2011 CSC 258/458 - Spring 2011 12

