

# Concurrent Online Servers

CS 256/456

Dept. of Computer Science, University of Rochester

# Concurrent Online Servers

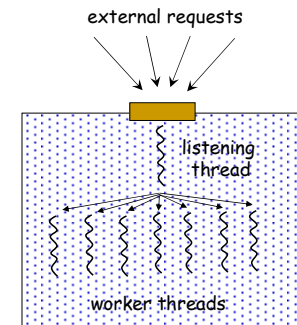
- Servers that
  - accept concurrent requests (potentially high concurrency)
  - serve online users (interactive responses)
- Examples
  - Web server: online server that implements HTTP
  - More complex ones: search engines (Google), online auction sites (eBay), discussion forums (slashdot), ... ..
- We examine supporting concurrent online servers
  - application level issues (with a good understanding of the OS)
  - OS issues

# Web Servers

- We take Web server as the example
  - although discussion should be applicable to other online servers.
- Steps of a HTTP request processing
  - read request from network
  - send HTTP header
  - read a piece of file and send data
  - read another piece of file and send data
  - read yet another piece of file and send data
  - ... ..
- Web servers you know of?

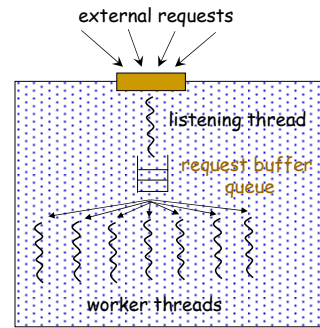
# Multi-processing vs. Multi-threading

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



## Controlling the Concurrency

- Overhead of high concurrency
  - more frequent context switches?
  - or what?
- How to control the execution concurrency?
  - dropping requests
  - employ a request buffer queue



4/4/2005

CSC 256/456 - Spring 2005

5

## User-level Threads

- Kernel threads
  - thread management/scheduling done by the OS kernel
- User threads
  - thread management/scheduling done at user-level
  - Benefits: (lightweight) less context switching overhead
- Problem of user threads
  - oblivious to kernel events, transparent to the kernel
  - e.g., 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?

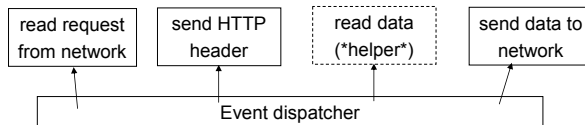
4/4/2005

CSC 256/456 - Spring 2005

6

## How to solve the blocking problem for user threads?

- For each potential blocking operation (e.g., read())
  - the server forwards it to a \*helper\* process (either spawned or pooled); the server then does something else ...
  - when the blocking operation in the helper process completes, the server is informed through IPC ...
- Event-driven concurrent servers



- Flash Web server [Pai et al, 1999 USENIX ATC]

4/4/2005

CSC 256/456 - Spring 2005

7

## How to solve the blocking problem for user threads?

- OS support for asynchronous I/O
  - server doesn't call blocking operations (e.g., read()) directly
  - instead, it calls asynchronous I/O operations (e.g., aio\_read()), which doesn't block
  - the server is notified by a signal (e.g., SIGIO) when the asynchronous I/O completes
  - an I/O handler is invoked at the receipt of the signal
- More general event-driven server design (but require asynchronous I/O support in OS)

4/4/2005

CSC 256/456 - Spring 2005

8

## Resource Management

- Scheduling or request execution
  - according to default CPU scheduling policy
- Staged request scheduling
  - Each request execution is partitioned into stages
    - parsing the input; read the file; send out the file
  - Request scheduling according to
    - which stage each request execution is at; and
    - whether the primary required resource is scarce or not.
- SEDA [Welsh et al., SOSP 2001]
- Capriccio [von Behren et al., SOSP 2003]

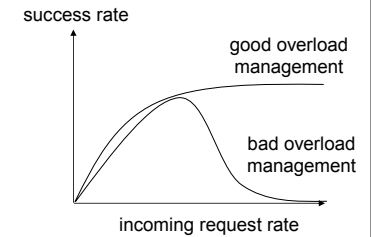
4/4/2005

CSC 256/456 - Spring 2005

9

## Handle Server Overload

- 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
- Managing overload?
  - drop requests if the buffer queue is already very long

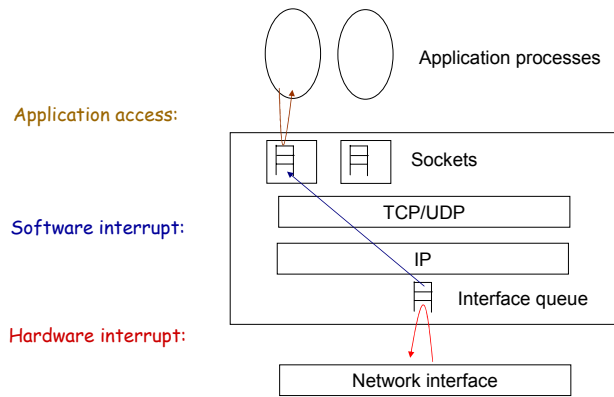


4/4/2005

CSC 256/456 - Spring 2005

10

## OS Overhead for Each Request

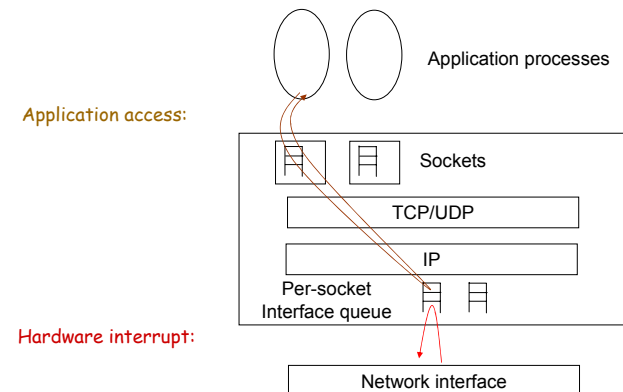


4/4/2005

CSC 256/456 - Spring 2005

11

## Lazy Receiver Processing [Druschel&Banga OSDI 1996]



4/4/2005

CSC 256/456 - Spring 2005

12

## Isolation in Server Systems

- Isolation of request execution in
  - fault
  - resource provisioning
  - system/server configuration
- Difficulty
  - process/thread does not completely encapsulate a request execution
- Request execution encapsulation in OS:
  - Resource containers [Banga et al., OSDI 1999]
  - Magpie [Barham et al., OSDI 2004]
- Request execution encapsulation in virtual machines:
  - Denali [Witaker et al., OSDI 2002]