# Parallelism and Concurrency in Servers

Kai Shen



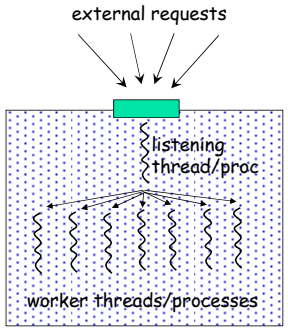
## Server

- Computer application serving (potentially many) interactive clients
  - An HTTP server that sends files upon requests
  - A database server that executes SQL queries upon requests
  - An email/messaging server that manages messages upon requests
  - … …
- 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



## 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
  - reuse a thread/process for multiple requests
  - may reduce process/thread creation and termination costs



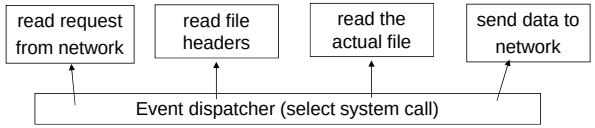



## User-level Threads

- Normal 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 (normal) threads
  - asynchronous I/O

## 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?



## 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]



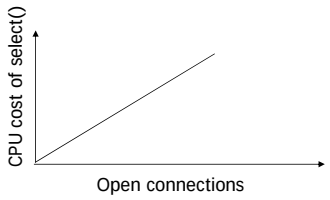
## High Concurrency Threaded Server

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



## Overhead with High Concurrency

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



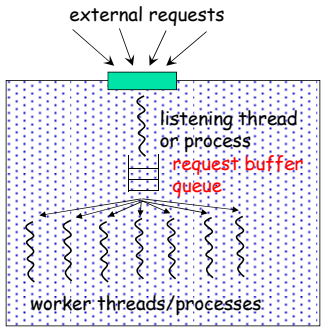
## Overhead with High Load Network Servers

- 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



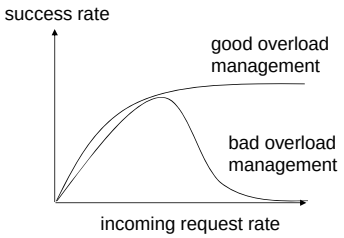
## Control the Concurrency

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





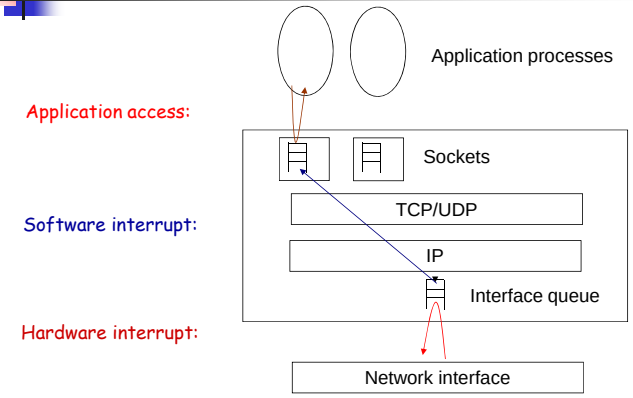

## Handle Server Overload



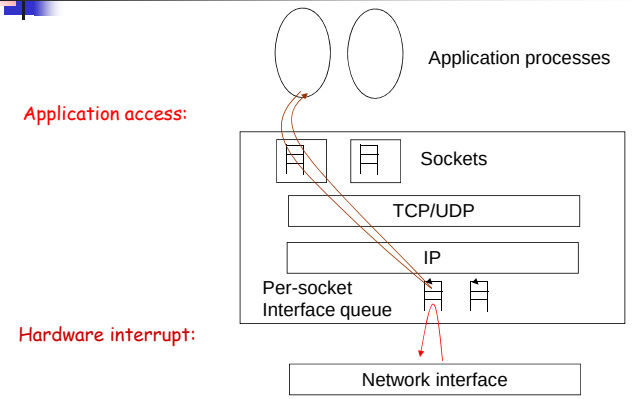

- Overhead of server overload:
  - some requests have to be abandoned
  - when a request has to be abandoned, resources already consumed by this request is wasted
- Principle: when abandoning a request, do so as early as possible
  - drop new requests if the buffer queue is already long



## OS Overhead for Each Request



## Lazy Receiver Processing
### [Druschel&Banga OSDI1996]






## Request Context Tracking

- Oftentimes a request maps to a process/thread, but not always:
  - Multi-stage server
  - Process pooling
  - Background tasks
- Tracking a request context helps:
  - Fine-grained resource accounting
  - Performance debugging
- How to track the request context?
  - Capture data dependencies – the web server request thread is sending a socket message to a database thread
  - Capture control dependencies – the main request thread is cloning another thread to do some auxiliary work



## An Example of Request Context