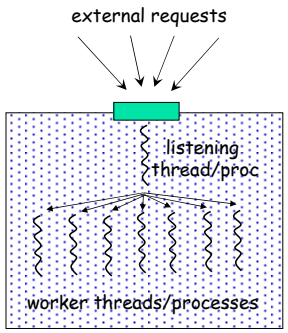
Concurrency Management

Kai Shen

Dept. of Computer Science, University of Rochester

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

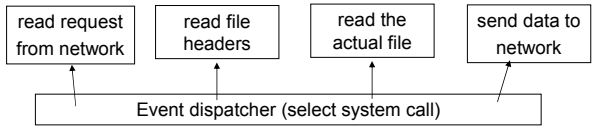


User-level Threads

- 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?
  - helper (kernel) 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]

Request Scheduling

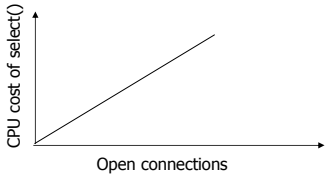
- Scheduling or request execution
  - normal multi-tasking (a task is a process/thread or a request)
- Staged resource-aware request scheduling
  - Each request execution is partitioned into stages (like in event-driven servers)
  - Request scheduling according to
    - which stage each request execution 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]

Problem with High Concurrency

- With a multi-threaded server, what problem do you first encounter when increasing the concurrency?
  - not enough stack space when there are too many threads
- With compiler analysis, can we bound the stack usage of each thread?
  - yes if there is no recursion
- Linked stack management
  - add checkpoint code to dynamically switch to new stack chunks

Overhead with High Concurrency

- Overhead of high concurrency
  - more frequent context switches?
- Or what?
  - scalability of the select() system call [Banga et al., USENIX1998]

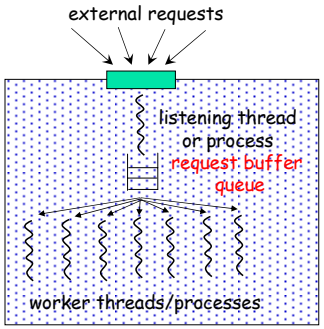


Overhead with High Load Network Server

- 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

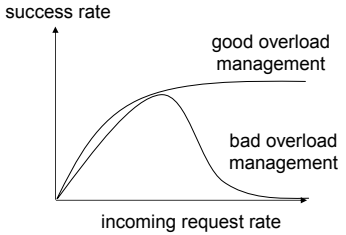
Control the Concurrency

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

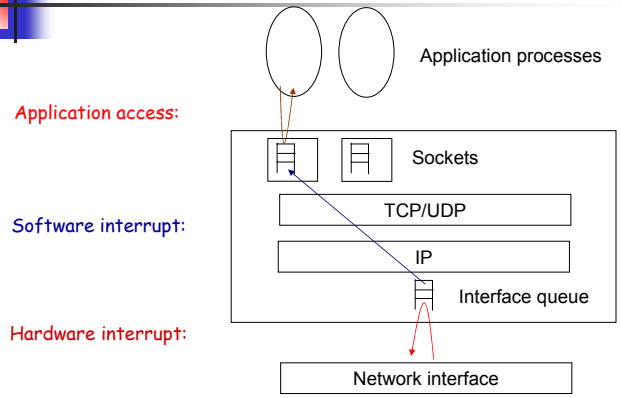


Handle Server Overload

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



OS Overhead for Each Request



Lazy Receiver Processing
[Druschel&Banga OSDI1996]