

Outline for Today

- Administrative
 - Talk to me about being a UTA for 110 next semester!
 - Course evaluations
 - Who wants to deliver to Owen?
 - Stop me at 3:15
 - Exams and problem sets to distribute (still)
- Objective: Discussion about Performance!

Evaluations

- 4075
- Circulating major codes
- Instructor 1
- Pencil or blue/black ink pens
- Both sides
- Comments, especially on discussion sections (new this semester to try to address class size)

WHY?

Industrial Partners voice concerns about lack of *intuition* about performance among new hires (in general -- not specific to Duke Ugrads, necessarily)

Goal: to spend one lecture making SURE it doesn't apply to Duke Ugrads once YOU are on-the-job.

How to Write a Bad Program (with the "help" of a bad OS)

- *Discussion* of different ways the design of a program and the OS can conspire to produce really *bad* performance (and conversely, what to avoid in order to produce really good performance).
 - Bad things the user can do in writing his/her program*
 - Bad things the OS developer can do
- This is one way of tying together many topics you've learned this semester.

List To Be Developed in Class

You will build this list of ideas in class...contribute!

List Developed in Class

- Make your program bloated – not freeing data structures, inefficient data structures
- Inefficient algorithms – ignore the impact of n , or ignore constant factor
- Create lousy spatial locality – put things related all over the address space
- Use fine grain I/O operations (read or write a byte at a time instead of a buffers-worth) – lots of I/O syscalls – blocking ones
- Turn off interrupts any time you get a chance*

- Choice of scheduling algorithm*
 - Enforce priorities strictly without any accommodation for starvation
 - Quantum too small – context switch overhead is fun.
 - Priority inversions – no inheritance
 - Discriminate against interactive tasks (foreground last)
- Very coarse grain synchronization – put everything in one big critical section
- Bad replacement algorithm in virtual memory – e.g. most recently used (in conflict with common usage patterns – ignore what your workload is)*
- Never yielding – using spinlocks (busywaiting) – (no preemption*)
- Not using threads properly
 - Within one thread – switch between naturally concurrent activities in a coarse-grained fashion and oblivious to response time of the foreground activity
 - Too many threads

My List

Algorithms*

- Chose an exponential algorithm when a logarithmic algorithm would do
- Ignore constant factor. Choosing the $O(c \log n)$ algorithm over the $O(n^2)$ algorithm when $c > n^2$ for the values of n that matter.
- Be clueless as to the significance of “ n ”

Concurrency*

- Wrong lock granularity
 - Lock the whole computation with a single monolithic lock → very big critical sections
 - Fine grain locks that cause lots of context switching
- Deadlocks or Starvation
 - Build a user-level thread package with blocking synch.
- Using busywaiting (spinlocks) where it ties up resources that are needed.
 - Expected long waits, RR scheduling
- Ignore opportunities for I/O overlap.

Interactions with Scheduler

- Priority Inversions
- Load Imbalances*
 - Create n threads for $n-1$ processors
- Creating too many processes to do a single task*
 - synchronizing among them creates serious contention
 - context switching overhead
 - overcommits resources (memory - see next slide)
- Very frequent daemons performing maintenance activity
- Choose inappropriate quantum values
 - Too long – response time impact
 - Too short – context switch overhead

Memory

- Lousy locality in a virtual memory environment*
 - pointer-based hither-and-yon data structures
 - access patterns that don't match layout in pages (column major order layout / row-wise access)
 - hashing (randomizes accesses)
- Overcommitted memory
 - large footprint*
 - creating lots and lots of processes*
 - not doing admission control
- Page allocation at odds with cache associativity (conflict misses in cache)
- Inappropriate page size

Files*

- Use really long absolute pathnames
- Lousy locality of file accesses
 - lousy spatial (block-grain transfers unjustified)
 - no reuse (lousy temporal locality - defeats cache effectiveness)
- Sync to disk early and often
- Structure data for an application as lots of individual files
 - each mail message being a separate file, for example
- Share data within a highly parallel job through files in a DFS rather than, say, messages.
 - Work the cache consistency mechanism very hard

Files

- No caching
- Lousy spatial locality
 - Separate related data: metadata, directories, blocks
 - Encourage fragmentation
- Offer only synchronous I/O operations (or implementation)
- Write through cache
- Assume all data is created equal – ignore semantic differences that could be detected.

IPC and Networks

- Use RPC when inappropriate
 - not client-server communication pattern

General

- Optimize for the uncommon case