Outline for Today

• Administrative
  – Talk to me about being a UTA for 110 next fall!
  – Midterms to distribute (still)

• Objective: Discussion about Performance!

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 Developed in Class

- Thrashing due to bad spatial locality in the program
  - Hashing randomizes accesses
  - Go counter to array layout (row-wise vs. column-wise)
  - Pointer-based data structure – “interesting” memory allocator
- High overhead deadlock checking algorithm
- Starvation due to priority inversion
- Starvation due to scheduling (shortest job first with constantly arriving short jobs)
- No synchronization – race conditions
- Too much synchronization – around stuff that doesn’t need it (very coarse grain – large lock scope)
- Too much synchronization – fine grain locking – overhead of acquiring and releasing all the time.
- Too small quantum in the scheduling - too much overhead in context switching

- Paging daemon – pageout most useful pages – page out the working set
- Thrashing because too many processes running – memory overcommitted – nobody’s working set can fit,
- Buffer overflow – can cause security breaches
- No security – weak security
- Spindown disk too aggressively. Spinup latency (wears out disk – reliability)
- Disk layout was bad – lots of seeking
- Contiguous layout on disk – requires compaction daemon to run
- Don’t readahead on disk accesses
- Use write through on disk cache or force every write to disk
- Swapping instead of virtual memory – huge context switching
- Allow only single threading – not exploiting idle times in interactive task with “background” activity – no I/O overlap
• No inodes (no indexing structure for disk blocks)
• Use really long pathnames
• No relative pathnames – always absolute
• No file sharing
• Use remote resources for everything
• No caching – of files, no tlb, no inodes in memory
• Bad temporal locality – no looping, no reuse of functions.
• Write a virus – don’t check for viruses – let the virus destroy the system.
• Run at low priority – nice
• Allow remote tasks to use up all your local resources – denial of service
• Loop overwhelms possibility of reuse – exec inside in loop
• Polling instead of using interrupts.
• Base timing (e.g. timeouts) on processor clock.
• Using busywaiting instead of blocking synchronization (esp. for critical sections that are big)

• Use any one of the 4 conditions for deadlock to make sure you do cause deadlock.
• Priority inversion (scheduler that does it deliberately)
• Fork a process inside a loop – lots of processes!
• No disk scheduling of any sort.
• Disk layout – spread blocks across the disk
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 $\rightarrow$ 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