Week5 (Feb 1 - 5)

Memory

In an idea world: * Lots of memory * Fast * Non-volatile

Real World Memory is: * Very large * Very fast * Affordable * Pick TWO

Memory Management Goal: Make real world look as much like ideal world as possible

Hierarchy

- What is the memory Hierarchy
  - Different levels
  - Some small fast
  - Others large slow
- What levels are usually included
  - Cache: Small fast epensive
    - L1 cache: usually on CPU chip
    - L2: may be on or off chip
    - L3 cache: off-chip, made of SRAM
  - Main Memory: medium speed/price
  - Disk: Super big, super slow, super cheap
- Memory manager handles memory hierarchy

Basic Mem management (old) Compoenents include * Operating System * Single process

Goal: Lay these out in memory * Mem protection may not be an issue (only one prog) * Flexibiltiy may still be useful (allow OS changes, etc)

No Swapping or paging

FIxed Partitions (multiple programs) *Fixed memm parittions* Devide into fixed spaces * Assign a process to a space when it's free * Mechanisms * Separate iniput queues for each partition * Single input queueu: better ability to optimize CPU usage

How many processes are engouh? * Several memory partitons (fixed or variable size) * Lots of processes wanting to use CPU * Tradeoff * More process utilize the CPU better * Fewer processes use less mem (cheaper) * How many processes do we need to keep the CPU fully utilized? * this will help determine hwo much mem we need * Is this still relevant with mem costing $5/GB?

Modeling multiprogramming

- More I/O wait means less processor utilization
  - At 20% IO wait, 3-4 procs fullly utilize CPU
  - At 80% I/O wait, even 10 processes aren't enough
- this means OS should havemore processes if they're IO bound
- More procs -> memory management and protection more important

Memory and Multiprogramming

- Mem needs two things for multiprogramming
  - Relocation *Addresses should be relative. Compiler should lie and tell programs that memory starts at 0. Consistent renaming.
    - This allows us to move programs within memory without screwing up addresses
  - Protection
    - Programs cannot clobber the memory of other programs
- The OS cannot be certain where a program will be loeaded in memory
  - variables and procedures can't use abslute locaions in memory
  - Seeral ways to guarantee this
- The OS must keep processes memory state separate
  - Protect a process from mthe other processes reading or modifying its own memory
  - Protect a process from moodifying its own memory in undesirable ways (such as writing to program code)

Base and Limit Registers (old)

- Special CPU registers: Base and LImitt
  - Access to registers limited to system mode
  - Regiesters contian
    - Base: Start of process's memory partition
    - Limit: Lenth of procs memory partition
- Address generation
  - PHysical address: location in actual mem
  - Logical address: location from procs point of view
  - Physical address = base + logical address
  - Logical address larger than limit -> error

PHysical: Actual byte address

Logical: Where the program things it is

Swappiong

Mem allocation changes as Processes come into memory * Processes leave memory * Swapped to disk * Complete execution Need to find spaces of contiguous memory

Swapping leaving rooom to grow

- Need to allow programs to grow
  - Allocate more memory for data
  - Larger stack
- Handled by allocating more space than is necessary at the start
  - Inefficient: Wastes memory that's not currently in use
  - What if procs rrequire too much memory? Take segment, swap it out, put it in a larger area

Tracking memory usage

- Operating systemm needs to track allocation state of memory
  - Regions that are available to hand out
  - Regions that are in use
    * Possibly what they're being used for
- Multiple approaches
  - Bitmap
  - Linked list
  - Buddy allocation
  - slab allocation

Bitmaps

- Keep track of free/allocated mem with a bitmap
  - One bit in map corresponds to a fixed size region of memory
  - Bitmap is a constat size for a igven amount of memory
  - Bitmap is constat size for a given amount of memory regardless of how much is allocated at a particular time
- Chunk size determines efficiency
  - At 1 bit per 4KB chunk, we just ned 256 bits (32 bytes) per MB of memory
  - For smaller chunks weneed more memory for the bitmap
  - can be difficult to find larg continuous free ares in bitmap

Bitmap keeps track of pages

Concept: INternal and External fragmentation

External: Space outside of blocks

Internal: Wasted space inside blocks

Tracking mem usage: Linked List

- Keep track of free/allocated memory regions with a linked list
  - Each entry in the list corresponds to a contiguous region of memory
  - Entry can indicate either allocated or free (and, optionally, owning process)
  - May have separate lists for free and allocated areas
- Efficient if chunks are large
  - Fixed-size representation for each region
  - More regions -> More space needed for free lists

Allocating Memory

- Search through greion list to find a large enough space
- Suppose there are several coices: which one to use?
  - First fit: the first suitable hole in the list
  - next fit: the first suitable after the previously allocated hole
  - Best fit: the smallest hole that is larger than the desired region (wastes least space)
  - Worst fit: largest available hole (leaves largest fragment)
- Option: Maintain separate queues for different-size holes

Freeing memory

- Allocation structures must be updated when momory is freed
- Easy with bitmaps: just set the appropriate bits in the bitmap
- Linked lists: modify adjacent elements as needed
  - Merge adjacent free regions into a single region
  - May invovlve mergign two regions with the just-freed area

Buddy alloction

- Allocate memory in powers of two
  - Good for objects of varied sizes
- Split larger chunks to create two smaller chunks
- When chunk is freed, see if it can be combined with its buddy to rebuild a larger chunk
  - This is recursive

Slab Allocation is a thing

Limitations of Swapping

- Problems with swapping
  - Process must fit into physical memory (impossible to run larger processes)
  - Memory becomes fragmented
    * External fragmentatin: lots of small free areas
    * Compaction needed to reassemble larger free areas
  - Proceses are either in memory or on disk: half and half doesn't do any good
- Overlays solved the first problem
  - Bring in pieces of the process over time (typically data)
  - Still doesn't solve the problem of fragmentation or parially resident process

Overlays: Only hold part of program you need. Program is like a tree, hold what parts of tree you need

Virtual Memory

- Basic idea: allow OS to hand out more memory than existt on the system
- Keep recently used stuff in physical memory
- Move less recently used tuff to disk
- Keep all of this hidden from processes
  - Processes still see an address space from 0 - max_address
  - movement of information to and from disk handled by the OS without process help
- Virtual Memory (VM) especially helpful in multiprogrammed systems
  - CPU schedules B while process A waits for its memory to be retrieved from disk

virtual and physical addresses

- Program uses virtual addresses
  - Addresses local to the process

  - Hardware translates virtual adress to phhsical address
- Translation done by the memory management unit
  - Usually on the same chip as the CPU
  - Only physical addresses leave the CPU/MMU chip
- Physical memory indexed by physical address

If instr give page fault (page not in memory) grab it from disk

Paiging and Page tables

- Virtual addressesmapped to physical addresses
  - Unitof mapping is called a page
  - All addresses in same virtual page are in the same physical page
  - Page table entry PTE contains translation for a single page
- Table translates virtual page number to physical page number
  - not all virtual memory has a physical page
  - Not every physical page need be used
- Exsample
  - 64KB virtual mem
  - 16KB phys mem

What's in a page table entry?

- Each entry contains:
  - Valid bit: set if this logical page number hasa corresponding physical frame inmemoery
    * if not valid, remainder of PTE is invalid
  - Page frame number: page in physical memory
  - Referencedbit: set if data on the page has been accessed
  - Dirty (modified) bit: set if data on page has been modified
  - Protection information

If valid, has a physical page frame number, else is a hint to where thing is held on disk

Mapping logical addresses to physical addresses

- Split address from CPU into two pieces
  - Page number (p)
  - page offset (d)
- Page number
  - Index into page table
  - Page table contains base address of page in physical memory
- Page offset
  - Added to base address to get actual phyiscal memory address
- Page size = 2ˆd bytes

32 bit logical address

20 bits page number 12 bits page offset

page frame number is index into the page table

Two level page tables

Problem: page tables can be too large

- 2^32 in 4KB pages = 1 Million PTEs
- Worse for 64 bit

Solution: use multi level page tables Page Size in first page table is arge (megabytes) * PTE marked invalid in first page table needs no 2nd level page table * Top level pages page tables * 1st level has pointers to second level * 2nd level has actual physical page numbers in it

Page table resides in Main memory

- CPU uses special regiesters for paging
  - Page table base register (PTBR) points to the page table
  - Page table length register (PTLR) contains len of page table; retricts max legal logical address
- Translating an addressrequires two memory accesses
  - First reads page table entry (PTE)
  - Second reads the data/instruction from mem
- Reduce num of mem accesses
  - Can't avoid second access (we need the val from mem)
  - Eliminate first access by eeping a hardware cache (called translation lookaside bufer or TLB) of recently used page table entries

TLB (Translation lookaside buffer)

- Search the tlb for the desired logical page number
  - Serach entries in paralel (of TLB and memory)
  - Use standared cache techniques
- If desired logical page number is found, get frame number from TLB
- If desired logical page number isn't found
  - Get frame number from page table in memory
  - Replace an entry in the TLB with the logical and physical page numbers from this reference

Handling TLB misses

- If PTE isn't found in TLB, OS needs to do the lookup in the page table
- Lookup can be done in hardware or software
- Hardware TLB replacement
  - CPU hardware does page table lookup
  - Can be faster than software
  - Less flexible than software,and more complex than hardware
- Software TLB replacement
  - OS gets TLB exception

  - Exception handler does page table lookup and places the result into the TLB
  - LArger TLB (lower miss rate) can make this feasible

How long do mem accesses take?

- Assume the following times
  - tlB lookup = a (often 0 - overlalped in cpu)
  - mem access time = m
- Hi ratio (h) is percent of time that a logical page number is found in the tlb
  - larger TLB usually means higher h
  - TLB structure can affect h as well

Inverted page table

- Reducec page table size further: keep on entry for each frame in memory
  - alternative: merge tables for pages in mem on dist
- PTE Contains
  - Virtual addr pointingtothis frame
- Search page table by
  - Hashing virtual page number and proces ID
  - starting at the entry correspondong to the hash result
  - Search until either the entry or a limit is reached
- Page fram number is index of PTE
- Improve performance by using more advanced hashing algorithms

Page replacement algorithms

- Page fault forces a choice
  - No room for new page (steady state)
  - Which page must be removed to make room for an incoming page
- How isa page removed form physica memory?
  - If the page is unmodified, simply overwrite it: a copy already exists on disk
  - If the page has been modified, it must be written back to disk: prefer unmodified pages
- Better not to choose an often used page
  - It'll probably be brought back in soon

Optimal Page Replacement

- What's the best we can poissibley do?
  - assume perfec knowledge of the future
  - not realizable in practice (usually)
  - useful for comparison: if another algorithm is within 5% of optimal, not much more can be done
- Algorithm: replace the page that will be used furthest into the future
  - Only works if we know the whole sequence

  - Can be approximated by running the program twice
    * onece to generate the reference trace
    * once to apply optimal algorithm
- Nice but not realistic

Not recently used (NRU) algorithm

- each bit has reference and dirty bit
- FOur classifications
  - Not ref, not dirty 0
  - not ref, dirty 1
  - ref, not dirty 2
  - ref, dirty 3
- Clear ref bit for all pages periodically
  - Can't clear dirty bit: needed to indicate which pages need to be flushed
  - Class one contains dirty pages where ref has been cleared
- Algorithm remove a page from lowest non-empty class
  - Select a page at random from that class
- Easy to understand and implement
- Performance adequate (though not optimal)

FIFO

- Maintain linked list of all pages
- Page at front replaced
- Easy to implement
- Disadvantage: Page in mem longest may be often used
  - Alg forces out regardless of usage
  - usage may be helpful in determining which to keep

second chance

- Modify FIFO to avoid throwing heavily used pages
  - If ref 0 throw out
  - If ref bit is 1
    * Reset to 0
    * Move page to tail
    * Continue search
- Still easy to implement, and better than plain FIFO

Clock Algorithm

- Same functionality as second chance
- Simpler implementation
  - Clock hand pointsto next page to replace
  - If R=0, replace

  - IF R=1, set R=0 and advance
- Continue until page with R=0 is found
  - May involve going around the clock

Least Recently Used (LRU)

- Assume pages used recently will be used again soon
  - throw out page that has been unused for longest time
- Must keep alinked list of pages
  - Most recent used at front, least at rear
  - UPdate this list every mem reference
    * THis can be somewhat slow: hardware has to update a linked list on every reference
- Alternatively, keep counter in each page table entry
  - Global counter incremembts with each CPU cycle
  - Copy global counter to PTE counter on a reference to the page
  - For replacement, evict page with lowest counter value