CS 537 Part 1 Study Guide By Leping Li

Tuesday, June 28, 2022 9:41 PM

Context Switching

OS @ run (kernel mode)	Hardware	Program (user mode)
To start process A:		
allocate entry in process table		
alloc memory for process		
set base/bound registers		
return-from-trap (into A)		
	restore registers of A	
	move to **user mode**	
	jump to A's (initial) PC	
		Process A runs
		Fetch instruction
	translate virtual address	
	perform fetch	
		Execute instruction
	if explicit load/store:	
	ensure address is legal	
	translate virtual address	
	perform load/store	
		(A runs...)
	Timer interrupt	
	move to **kernel mode**	
	jump to handler	
Handle timer		
decide: stop A, run B		
call `switch()` routine		
save regs(A) to proc-struct(A) (including base/bounds)		
restore regs(B) from proc-struct(B) (including base/bounds)		
return-from-trap (into B)		
	restore registers of B	
	move to **user mode**	
	jump to B's PC	
		Process B runs
		Execute bad load
	Load is out-of-bounds;	
	move to **kernel mode**	
	jump to trap handler	
Handle the trap		
decide to kill process B		
deallocate B's memory		
free B's entry in process table		

System Call

OS @ boot (kernel mode)	Hardware	
initialize trap table		
	remember address of... syscall handler	

OS @ run (kernel mode)	Hardware	Program (user mode)
Create entry for process list		
Allocate memory for program		
Load program into memory		
Setup user stack with argv		
Fill kernel stack with reg/PC		
return-from-trap		
	restore regs (from kernel stack)	
	move to user mode	
	jump to main	
		Run main()
		...
		Call system call **trap** into OS
	save regs (to kernel stack)	
	move to kernel mode	
	jump to trap handler	
Handle trap		
Do work of syscall		
return-from-trap		
	restore regs (from kernel stack)	
	move to user mode	
	jump to PC after trap	
		...
		return from main
		trap (via `exit()`)
Free memory of process		
Remove from process list		

Scheduling Policy

- First In First Out
  - Short-running jobs wait for long-running jobs, terrible average turnaround time, as known as the **convoy effect**
  - Easy to implement, ok when jobs have the same length
- Shortest Job First
  - runs the shortest job first, then the next shortest, and so on
  - Assume OS has perfect information
  - Better at minimizing average turnaround time
  - non-preemptive, shorter job arrives when a job is running
  - Starvation, long-running jobs may never run
- Shortest Time-To-Completion First
  - Schedule different jobs by taking CPU away from running job
  - preemptive
  - Always run job that complete quickest
  - Starvation
- Round Robin
  - Preemptive

    - Alternate ready processes for a fixed-length time-slice
    - Run jobs for a time slice, then switch to the next
    - The cost of context switching is high
    - Fair, good average respond time but cost of turnaround time
- Multi-Level Feedback Queue
    - Multi-level, each level is a queue, different time slice for different levels, lower level with longer time slice
    - Jobs with high priority run, Round Robin if the same priority
    - When a job arrives, give it the highest priority
    - If a job uses whole slice, demote it
    - If a job gives up the CPU before the time slice is up, stays at the same priority
    - Boost priority of all jobs after a period of time

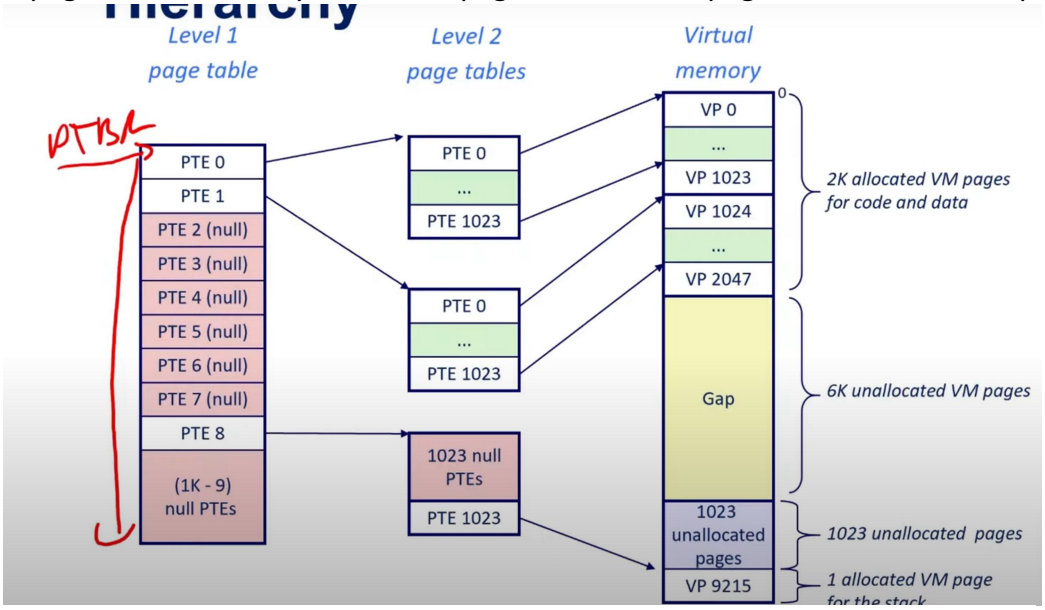
Memory Virtualization

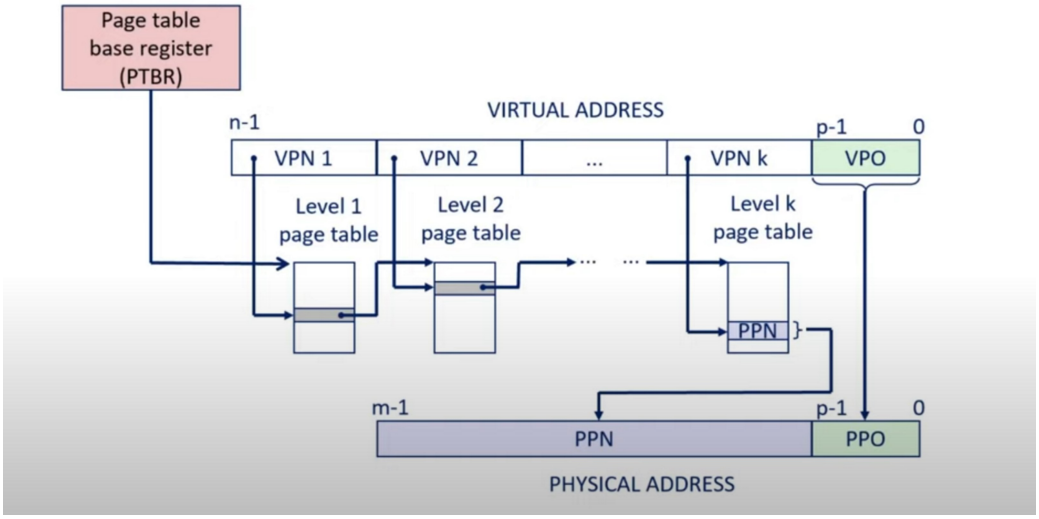
- Time sharing
    - One process use the memory at a time, swap out, then switch to another process
    - Ridiculously poor performance
- Static Relocation
    - OS picks static location for each process when started
    - Rewrite each code segment before loading to the memory
    - Each rewrite for different processes to different addresses and pointers
    - OS does not rewrite stack address because each process has its own stack pointer, no collision
    - No protection
    - Cannot move after placed
- Dynamic Relocation, Base approach
    - Translate virtual to physical address by adding a fixed offset each time
    - Hardware MMU required, store offset in base register
    - Each process has different value in base register
    - Physical address = Virtual address + offset
    - No protection
- Dynamic Relocation, Base and Bound approach
    - Bound: size of the process' virtual address space
    - If process loads/stores beyond the bound, OS kills process
    - Need to track base and bound in PCB, extra context switch work
    - Protection
    - OS can place process at different locations initially
    - OS can move processes if need
    - Simple, fast, efficient
    - Entire address space must be allocated contiguously in physical memory
    - Reserve large address space may not be used by process
    - Cannot share limited parts of address space with process
- Segmentation
    - Each segment has base and bound register
    - Each segment can be independent, no need to be contiguously (sparse allocation)
    - grow and shrink. For heap, system call sbrk(). For stack, if OS recognizes reference near outside legal segment, extend stack implicitly

    - Set read/write bits to protect, enable sharing
    - Two part of logical address, top bits indicate segment and rest bits indicate offset
    - Each segment must be allocated contiguously
    - External Fragmentation
- Paging
    - Eliminate requirement that address space is contiguous
    - Eliminate external fragmentation
    - Ability to grown if needed
    - Divide address and physical space into fixed-size pages
    - High-order bits of address designate page number
    - Low-order bits of address designate offset within page
    - Number of bits in virtual address need not equal number of bits in physical address
    - Linear page table to store physical frame number, virtual page number is the index
    - Size of page table = number of entries * entry size
    - Address of page table is stored in page table base register so hardware can find it
    - Save page table base register when context switch
    - Valid bit(VPN to PFN valid or not) and protection bit(read/write)
    - Does not have to coalesce with adjacent free space
    - Internal fragmentation
    - Large page table
    - Additional memory reference
    - Each page table must be allocated contiguously
    - Process of translation
        1. Extract VPN from virtual address
        2. Calculate address of PTE
        3. Read PTE from memory
        4. Extract PFN
        5. Build PA
        6. Read contents of PA
- Paging with TLB
    - Cache VPN to PFN translation, part of MMU
    - TLB entry
        1. Tag(valid virtual page number)
        2. Physical page number(page table entry)
        3. Protection bit
    - Hardware and software can both handle TLB miss
    - Replacement policy
        1. LRU, good for sequential workload such as code segment. 100% miss rate when repeatedly access same offset across n pages but having n-1 TLB entries
        2. Random, good for temporal locality
    - Empty TLB when context switch or add address space identifier in TLB(similar to process identifier), context switching affects performance
- Paging and Segmentation
    - Divide address into segments
    - Divide each segment into fixed-size pages
    - Logical address divides into 3 parts: seg, page number, page offset
    - Base register holds the base of page table
    - Support sparse address spaces, decrease size of page table(only need PTE for allocated portions)

- No external fragmentation, segments can grow
- Share either page or segment
- Must allocate each page table contiguously
- Page table size can be very large

- Multi-level page table
  - Allow each page table to allocated non-contiguously
  - Only allocate portion of page tables if corresponding pages in use
  - Page the page tables
  - Logical address divides into 3 parts: outer page, inner page, page offset
  - Outer level is as known as page directory
  - A page directory entry has a valid bit, valid bit is set to 1 if corresponding page in use
  - If page directories may not fit in a page, use another page dir to refer to the page dir pieces





- Inverted Page Table
  - Only need entries for virtual page with valid physical page mapping, have entries based on physical pages
  - Use hash table to store entries <vpn + asid, ppn> to find match
  - Faster look up
  - Required software managed TLB because hardware does not understand hash table

- Swapping Page Table
  - Swap page to disk when not enough memory
  - Present bit is set to 1 when a page is in memory, 0 in disk
  - If present bit is 0, PTE points to the disk, and causes trap to OS when page is referenced
  - Dirty bit is set to 1 when a page in memory, but content does not match with content on disk, store new content to disk
- First In First Out Page Replacement Policy
  - Have a queue, replace page that has been in memory longest
  - Fair, all pages receive equal residency, easy to implement
  - Some pages may always be needed
  - Belady's anomaly, increasing the number of page frames results in an increase in the number of page faults
- LRU Page Replacement Policy
  - Replace page not used longest in the past
  - Use past to predict future
  - With locality, behave the same as OPT
  - Smaller memory sizes are guaranteed to contain a subset of larger memory size, reason why Belady's anomaly not occurs
  - Hard to implement, does not handle all workloads well
  - Clock algorithm
    1. Look for page with use bit cleared
    2. There is a pointer pointing to the last examined page frame
    3. Traverse pages in circular buffer
    4. Clear use bits as search
    5. Stop when find page with use bit 0
    6. Replace multiple pages at once because expensive to run replacement algorithm and write single block to disk
    7. Use dirty bit to give preference to dirty pages because it is more expensive to replace dirty page, have to write it to disk, replace page that have use bit and dirty bit cleared
    8. Hand moving slowly(good), hand moving fast(bad, means keeps finding that the use bit is set on different pages, so many different pages are being accessed)
- Optimal Page Replacement Policy
  - Predict the future
  - Replace page that will be accessed furthest in the future
  - Potential approach: machine learning, train the model to predict the future
  - Impractical, only use for comparison with other policies