

Midterm Announcements

- **Midterm is today 3/18 155 Dwinelle 5:30-7:00**
- You will all do well. ☺

Today's Menu: Requested Review Topics

- TLBs
- 2-level Page Tables
- Locks, Semaphores, Condition Variables, and Monitors
- Banker's Algorithm
- Q and A

Caching and TLBs

**QUESTION:** Multilevel paging makes storage efficient, but what's the drawback?  
What's the solution?

**ANSWER:** Waaay too many memory accesses to do this for every instruction. Things would crawl! Solution: why not cache translations to make this fast? Use TLB, Translation Lookaside Buffer: caching applied to address translation.

- Types of cache as applied to TLB:
  1. Direct Mapped - restrict each virtual page to use specific TLB entry
    - Kinda like hashing—1 to 1 mapping.
    - Fast and simple, but have to deal with collisions.
  2. Set Associative - partition TLB into N separate banks. Select entry with low order virtual page number bits. Compare all N entries in parallel.
    - Solves collisions partially—each entry has a specific bank it can be in.
    - However, more complex, still have collisions.
    - Slow unless done in parallel.
  3. Fully Associative - Compare entire TLB in parallel. Only one entry per bank.
    - Entry can be anywhere in cache.
      - No collisions—easy to place in cache.
    - However, must search in parallel to make fast enough
      - **HARD** to implement in hardware.
- Effective access time in a TLB:
  1.  $P(\text{hit}) * \text{cost of hit} + P(\text{miss}) * \text{cost of miss}$

**QUESTION:** Given the following TLB characteristics:

2. 20 ns ==> TLB access
3. 100 ns ==> Memory access

4.  $P(\text{hit}) = .80$

What would the effective access time be?

**ANSWER:**  $EAT = .80(120) + .20(220) = 140 \text{ ns}$

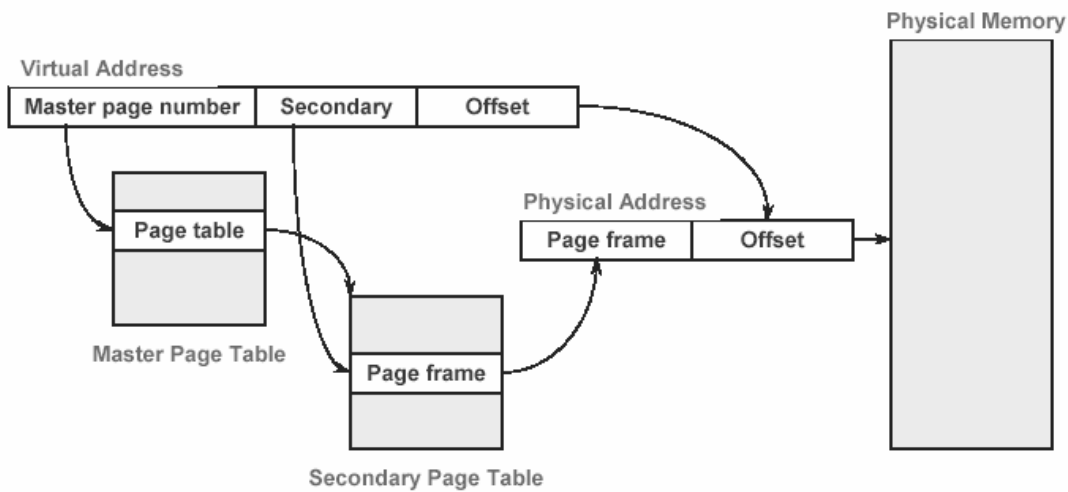
- Is this good?

1. Hierarchy of access times for all things memory:

	Latency	Size	Cost
Registers	2.5ns	32-128 bytes	on chip
On-chip cache	5ns	32KB	on chip
Off-chip cache	20ns	512KB	\$2000/MB
Main memory	40ns	512MB	\$4/MB
Disk	10ms (10M ns)	100 GB	\$0.001/MB
Robotic tape	10s (10B ns)	100 TB	Factor of 3-5 less than disk.

Multilevel Paging

- Solves the problem by using a hierarchy of page tables



- Key observation: since at any given time a process is only using a small portion of its address space, only need page table entries corresponding to what's actually in use

1. Note that later on we will talk about DEMAND PAGING: Moving memory to and from disk
2. This is going to allow us to allocate more virtual memory than we actually have physical RAM
3. With multilevel page tables, the PAGE TABLES can be PAGED—meaning that if the process has a portion of its address space which hasn't been used in a long time, those page tables (and the associated pages) go out to disk

- This is project phase 3!

- Multilevel paging on the x86

1. Uses two-level page tables
2. Virtual address is divided into 3 components:

10 bits (31 .. 22): Page directory index  
 10 bits (21 .. 12): Page table index  
 12 bits (11 .. 0): Offset

- Example: Virtual address

0x13a49F01 =	0001001110	1001001001	11110000001
	0x04e	0x249	0xF01
	pgdirind.	pgtblind.	offset

- First, use page dir index as index into "page directory."
  1. Physical base address of the page directory is stored in a special register (called "CR3") on the x86. It is always page-aligned, so the base address might be 0x0001c000.

**QUESTION:** How many entries in the page directory if each entry is 4 bytes (why 4 bytes?)?

**ANSWER:** Well 10 bits of index, so  $2^{10}$  entries \* 4 bytes each entry = 4Kb (One page!)

- (use picture on board) At offset 0x04e in the page directory we find the physical address of page table, e.g. 0x000a2000. (Note that the page table is also page-aligned and holds just one page.)
  1. We use the "page table index" as an index into this second-level table.
  2. At index 0x249 into the page table at 0xa2000 we find the physical page number of the address we are looking for, e.g., 0x0341b000.
  3. We add to this the offset (0xf01) to get the final physical address: 0x0341bf01.
- Generally the page directory entries contain PHYSICAL addresses for page tables.
  1. However, if page not in memory, then the entry would contain an OS-specific pointer (such as a disk block address) used to record where the page table is stored on disk.
  2. In this way, page directory entries always use PHYSICAL addresses, but the page tables can still be swapped out.
- Note that the page directory is always resident in physical memory, meaning that each process has 1 page always "pinned"

### Locks, Semaphores, Monitors, and Condition Variables

- Locks
  - o Lock the critical section, wait if can't get lock.
  - o On release, wait up anyone waiting to acquire.
- Semaphores
  - o Like a generalized lock, with a counter

- You can't get the value out of the counter.
  - Two functions:
    - P() – Decrement if semaphore is positive
      - Sleep otherwise
    - V() – Increment, waking up anyone waiting
- **QUESTION:** What do 'P' and 'V' really mean?
- **ANSWER:** 'proberen' means 'try' or 'attempt' in Dutch, and 'verhogen' means 'increase' or 'raise'
  - Mention opposite from down trick!
  - Semaphores are great for resource control
    - Example: Producer/Consumer problem
- Condition variables
  - Let us sleep within a critical section by atomically releasing lock when we sleep.
    - Obviously, must use with a lock
  - Three functions:
    - Wait() – Release lock, sleep, must re-acquire on wake.
    - Signal() – Wake up waiter
    - Broadcast() – Wake up ALL waiters
- **QUESTION:** Do condition variables keep state? Do semaphores?
- **ANSWER:** By themselves, semaphores keep state: i.e. if you increment a semaphore, that means a thread can continue in the future even if no thread is waiting. A signal when no thread is waiting is lost.
- Monitor
  - High-level synchronization primitive.
  - Consists of a lock and zero or more condition variables
  - Encapsulates shared data or method and regulates access.
  - Mesa vs. Hoare:
    - Mesa: signaler keeps lock, waking thread must wait on acquire
    - Hoare: signaler releases lock, waking thread acquires and runs.
- These things can be built out of each other
- **QUESTION:** How to use semaphores to implement a lock?
- **ANSWER:** Make a Lock class using a binary semaphore, and map acquire() and release() accordingly. Start semaphore at 1.
- **QUESTION:** How to use semaphores to implement condition variables?
- **ANSWER:**
  - Keep a counter of how many threads are trying to get this condition variable. Have a semaphore that starts at 0
  - Wait: update counter, call semaphore->P()

- Signal: increment semaphore only if there is a thread waiting.
    - Why? (otherwise signal doesn't behave correctly)
  - Broadcast: increment semaphore for each thread waiting.
- **QUESTION:** How to use lock to implement semaphores?
- Brainteaser for your week. 😊

### The Banker's Algorithm

- **QUESTION:** What are the four conditions for deadlock?
- **ANSWER:**
  1. Limited resources
  2. No preemption (can't take away resources)
  3. Multiple independent requests
  4. Circular chain of requests
- Preventing deadlock: the Banker's Algorithm
  1. Important stuff; might be a midterm question!
  2. Source of name: Can be used by banks so they don't allocate their available cache without being able to satisfy the needs of all customers.
- When a process enters system, it declares maximum resources it may need.
- Example:
 

Suppose we have 12 available tape drives. After the following allocations...

Job	Needs	Allocated
1	8	3
2	7	2
3	7	1
4	2	0

... we have 6 tape drives left. Now suppose the following request from job 5 comes in. The request is for 4 tape drives immediately, and 2 more possibly.

Job	Needs	Needs Now
5	6	4

Using the Banker's Algorithm, show that the system is or isn't in a safe state.

- **QUESTION:** Is it safe? Can we finish safely?
- **ANSWER:** Yup. Here's how:
  1. Allocate the 4 drives
    - a. Available drives now = 2
  2. Can anybody finish? Yes. Job 4 can finish

- a. Available drives now = 2
- 3. Can anybody finish? Yes. Job 5 can finish
  - a. Available drives now = 6
- 4. Can anybody finish? Yes. Job 1 can finish
  - a. Available drives now = 9
- 5. Can anybody finish? Yes. Job 2 can finish
  - a. Available drives now = 11
- 6. Can anybody finish? Yes. Job 3 can finish
  - a. Available drives now = 12