Lecturer SOE
Dan Garcia

inst.eecs.berkeley.edu/~cs61c

UCB CS61C : Machine Structures

Lecture 33 – Virtual Memory I
2011-11-14

8 TB SOLID STATE DRIVE (SSD)!

OCZ has showcased an 8 TB solid state drive (the biggest HDD is only 4 TB, they've caught up!) Unfortunately, it's not released yet and the price will be astonomical.

http://news.softpedia.com/news/OCZ-Showcases-4TB-and-8TB-SSDs-at-CeBIT-2011-187631.shtml

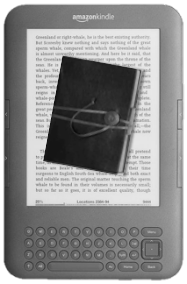
Review

- Pipelining is an important form of ILP
- Challenges are hazards
  - Forwarding helps w/many data hazards
  - Delayed branch helps with control hazard in 5 stage pipeline
  - Load delay slot / interlock necessary
- More aggressive performance:
  - Longer pipelines
  - Superscalar
  - Out-of-order execution
  - Speculation

Designing an e-journal in 1970

SPEC

- Want to be able to read and write words on a page
- Start with a blank journal, also want to be able to write anywhere in journal
- Problem is, only enough physical memory on device for 4 pages!



More details on our 1970 e-reader

- Each page only 32 B
  - 5 bits to specify the byte within a particular page
  - The "page offset"
- 4 physical pages
- What if you had a wireless connection to a disk that could hold 8 pages…
  - What illusion / abstraction could we provide to the user?

BINGO! Make them think they have 8!

- We'll distinguish
  - "physical" memory "resident" to the device
    - E.g., 4 pages
  - "virtual" memory that the user should use
    - E.g., 8 pages
- What's needed to keep track of which page is in memory & where

#	Virtual
0	Dear diary
1	One day…
2	Yesterday…
3	
4	OMG, did u…
5	
6	Occupy Cal!
7	CS61C is great!

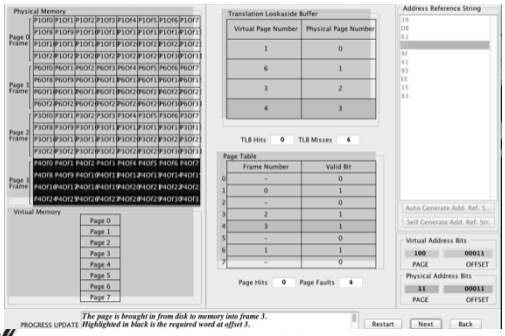
We need a "page table"

#	Frame # (physical page)	Valid (resident)
0		
1	3	True
2	1	True
3		
4		
5		
6	0	True
7	2	True

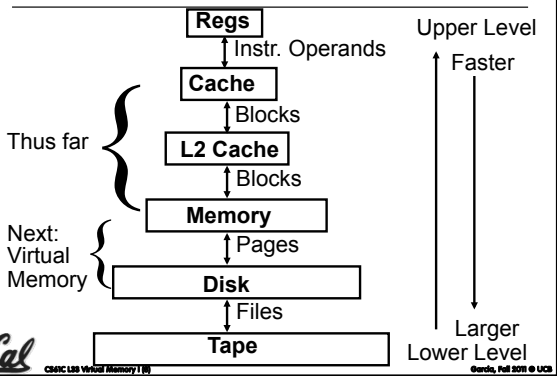
Page Table

#	Virtual
0	Dear diary
1	One day…
2	Yesterday…
3	
4	OMG, did u…
5	
6	Occupy Cal!
7	CS61C is great!

Let's see a simulation of our e-journal!



Another View of the Memory Hierarchy



Memory Hierarchy Requirements

- **If Principle of Locality allows caches to offer (close to) speed of cache memory with size of DRAM memory, then recursively why not use at next level to give speed of DRAM memory, size of Disk memory?**
- **While we're at it, what other things do we need from our memory system?**

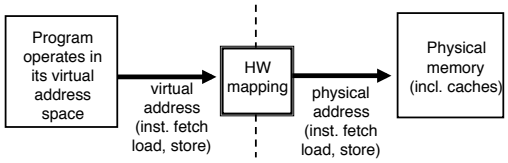
Memory Hierarchy Requirements

- **Allow multiple processes to simultaneously occupy memory and provide protection – don't let one program read/write memory from another**
- **Address space – give each program the illusion that it has its own private memory**
  - Suppose code starts at address 0x40000000. But different processes have different code, both residing at the same address. So each program has a different view of memory.

Virtual Memory

- **Next level in the memory hierarchy:**
  - Provides program with illusion of a very large main memory:
  - Working set of "pages" reside in main memory - others reside on disk.
- **Also allows OS to share memory, protect programs from each other**
- **Today, more important for protection vs. just another level of memory hierarchy**
- **Each process thinks it has all the memory to itself**
- **(Historically, it predates caches)**

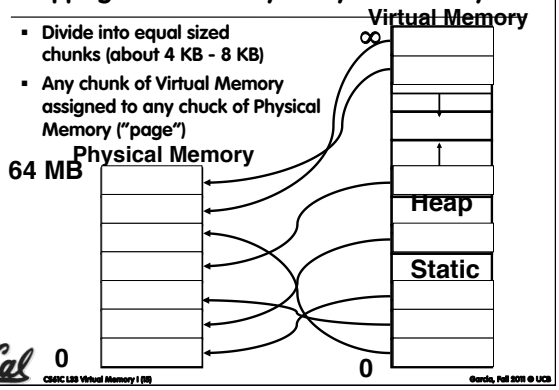
Virtual to Physical Address Translation



- **Each program operates in its own virtual address space; ~only program running**
- **Each is protected from the other**
- **OS can decide where each goes in memory**
- **Hardware gives virtual ⇒ physical mapping**

Mapping Virtual Memory to Physical Memory

- Divide into equal sized chunks (about 4 KB - 8 KB)
- Any chunk of Virtual Memory assigned to any chuck of Physical Memory ("page")



Virtual Memory Mapping Function

- Cannot have simple function to predict arbitrary mapping
- Use table lookup of mappings

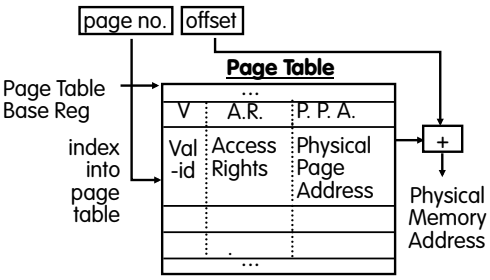
Page Number	Offset

- Use table lookup ("Page Table") for mappings: Page number is index
- Virtual Memory Mapping Function
  - Physical Offset = Virtual Offset
  - Physical Page Number = PageTable[Virtual Page Number]

(P.P.N. also called "Page Frame")

Address Mapping: Page Table

Virtual Address:



Page Table located in physical memory

Page Table

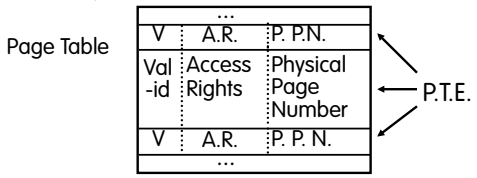
- A page table is an operating system structure which contains the mapping of virtual addresses to physical locations
  - There are several different ways, all up to the operating system, to keep this data around
- Each process running in the operating system has its own page table
  - "State" of process is PC, all registers, plus page table
  - OS changes page tables by changing contents of Page Table Base Register

Requirements revisited

- Remember the motivation for VM:
- Sharing memory with protection
  - Different physical pages can be allocated to different processes (sharing)
  - A process can only touch pages in its own page table (protection)
- Separate address spaces
  - Since programs work only with virtual addresses, different programs can have different data/code at the same address!
- What about the memory hierarchy?

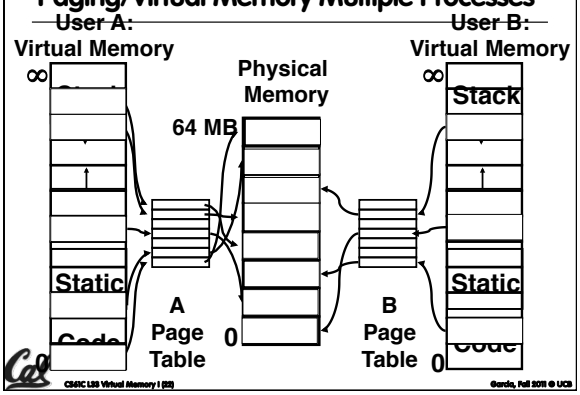
Page Table Entry (PTE) Format

- Contains either Physical Page Number or indication not in Main Memory
- OS maps to disk if Not Valid (V = 0)



- If valid, also check if have permission to use page: Access Rights (A.R.) may be Read Only, Read/Write, Executable

Paging/Virtual Memory Multiple Processes



Comparing the 2 levels of hierarchy

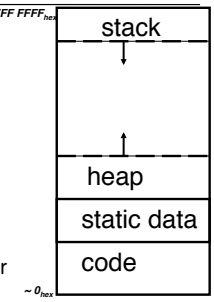
Cache version	Virtual Memory vers.
Block or Line	Page
Miss	Page Fault
Block Size: 32-64B	Page Size: 4K-8KB
Placement: Direct Mapped, N-way Set Associative	Fully Associative
Replacement: LRU or Random	Least Recently Used (LRU)
Write Thru or Back	Write Back

Notes on Page Table

- Solves Fragmentation problem: all chunks same size, so all holes can be used
- OS must reserve "Swap Space" on disk for each process
- To grow a process, ask Operating System
  - If unused pages, OS uses them first
  - If not, OS swaps some old pages to disk
  - (Least Recently Used to pick pages to swap)
- Each process has own Page Table
- Will add details, but Page Table is essence of Virtual Memory

Why would a process need to "grow"?

- A program's *address space* contains 4 regions:
  - stack: local variables, grows downward
  - heap: space requested for pointers via `malloc();` resizes dynamically, grows upward
  - static data: variables declared outside main, does not grow or shrink
  - code: loaded when program starts, does not change



For now, OS somehow prevents accesses between stack and heap (gray hash lines).

Peer Instruction

1) Locality is important yet different for cache and virtual memory (VM): temporal locality for caches but spatial locality for VM
2) VM helps both with security and cost

	12
a)	FF
b)	FT
c)	TF
d)	TT

And in conclusion…

- Manage memory to disk? Treat as cache
  - Included protection as bonus, now critical
  - Use Page Table of mappings for each user vs. tag/data in cache
  - TLB is cache of Virtual ⇒ Physical addr trans
- Virtual Memory allows protected sharing of memory between processes
- Spatial Locality means Working Set of Pages is all that must be in memory for process to run fairly well