ECE 598 – Advanced Operating Systems
Lecture 11

Vince Weaver

http://www.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

19 February 2015

Announcements

- Homework #1 and #2 will be graded soon.
- Homework #3 was posted.

OS Monitor Aside / HW#3

- Writing a simple shell
- Many old system or OSes would have a simple "monitor" you'd drop into when something went wrong
- Our HW#2 code was just echoing and printing a single char at a time.
- Usually input shells (like the Linux console) are line-oriented.

- Input, especially serial input, a pain on Linux. Back to the old teletype days. Character or line oriented, complex ioctls to set up.
- In HW#3, design a "monitor" that takes simple commands.
  Read the keypresses (echoing). Then when press enter pass the string to a function that parses it.
- How to parse?
  Traditionally you use `strtok()` or `strcmp()` but we don't have that. So you can either write one, or cheat

and look at individual chars to match.

- How to handle backspace? Always a challenge. Look for Control-H in the input stream?

HW1 Review

1. What does the OS provide?
   A layer of abstraction, etc.

2. What is the downside to an OS?
   Various types of overhead.

3. Comparative sizes of ASM/C/Linux?
   80 bytes to 270 bytes to 4MB.

4. What does volatile mean?
   Tells the compiler not to optimize away reads/writes.

5. Why are the C defines 1/4 the value of the assembly ones?
   We were accessing the GPIOs as an integer array, so the C compilers was indexing by 4-byte chunks.

6. What happens if you set ALT4 output instead of GPIO out?
   The pins would be configures for SPI_CE2_N (SPI chip select2).

Overlays

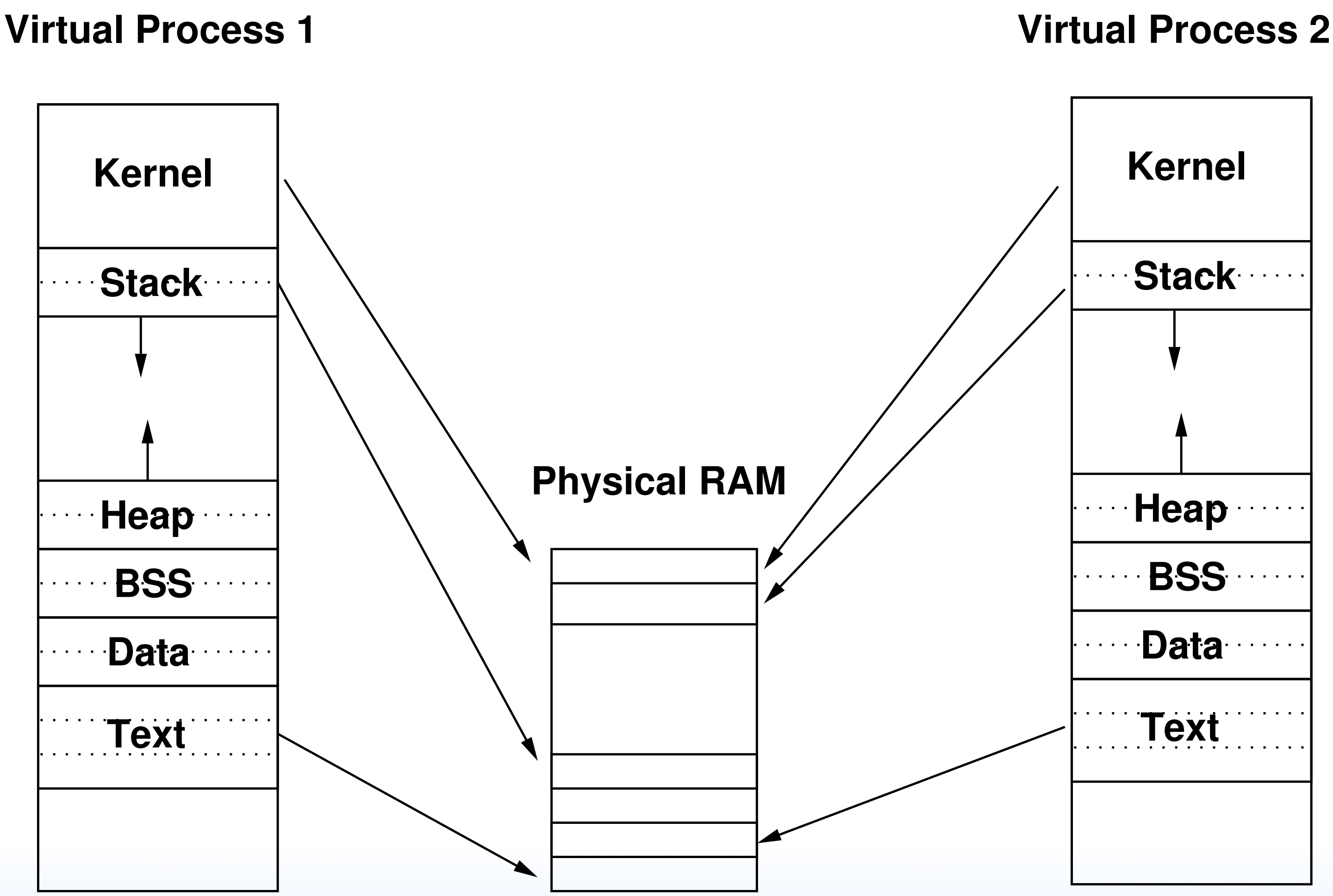
- How can you have a program that accesses more RAM than available in physical memory?
- Can handle this in software with a technique called overlays.
- Split program in parts. Only load the part currently running at any given time.
- Can we have hardware do this automatically? This is part of the idea of virtual memory.

Virtual Memory

- Original purpose was to give the illusion of more main memory than available, with disk as backing store.
- Give each process own linear view of memory.
- Demand paging (no swapping out whole processes).
- Execution of processes only partly in memory, effectively a cache.
- Memory protection

- Reduces fragmentation

Diagram


Virtual Process 1
Virtual Process 2
Kernel
Stack
Heap
BSS
Data
Text
Physical RAM
Kernel
Stack
Heap
BSS
Data
Text

Memory Management Unit

Can run without MMU. There's even MMU-less Linux.

How do you keep processes separate? Very carefully...

Page Table

- Collection of Page Table Entries (PTE)

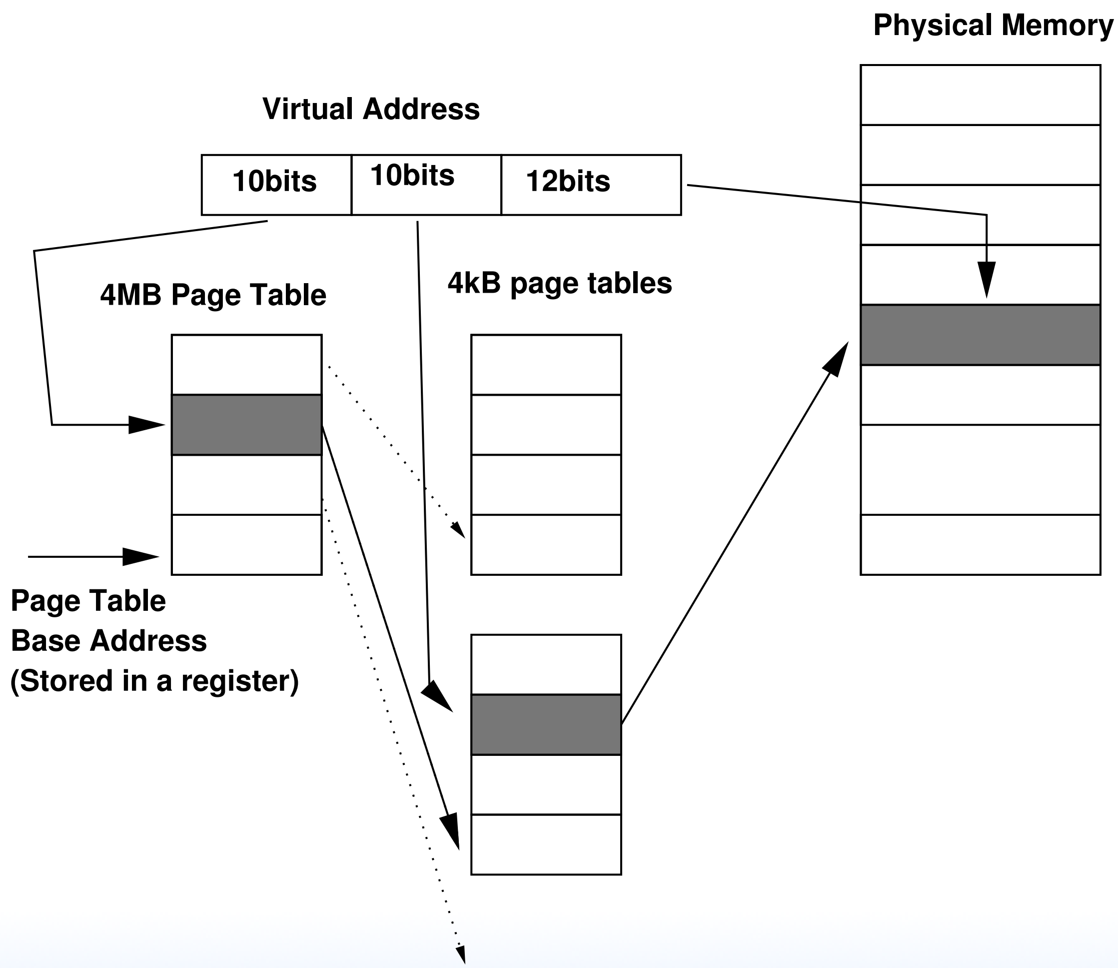
- Some common components: ID of owner, Virtual Page Number, valid bit, location of page (memory, disk, etc), protection info (read only, etc), page is dirty, age (how recent updated, for LRU)

Hierarchical Page Tables

- With 4GB memory and 4kb pages, you have 1 Million pages per process. If each has 4-byte PTE then 4MB of page tables per-process. Too big.

- It is likely each process does not use all 4GB at once. (sparse) So put page tables in swappable virtual memory themselves!
  4MB page table is 1024 pages which can be mapped in 1 4KB page.

Hierarchical Page Table Diagram



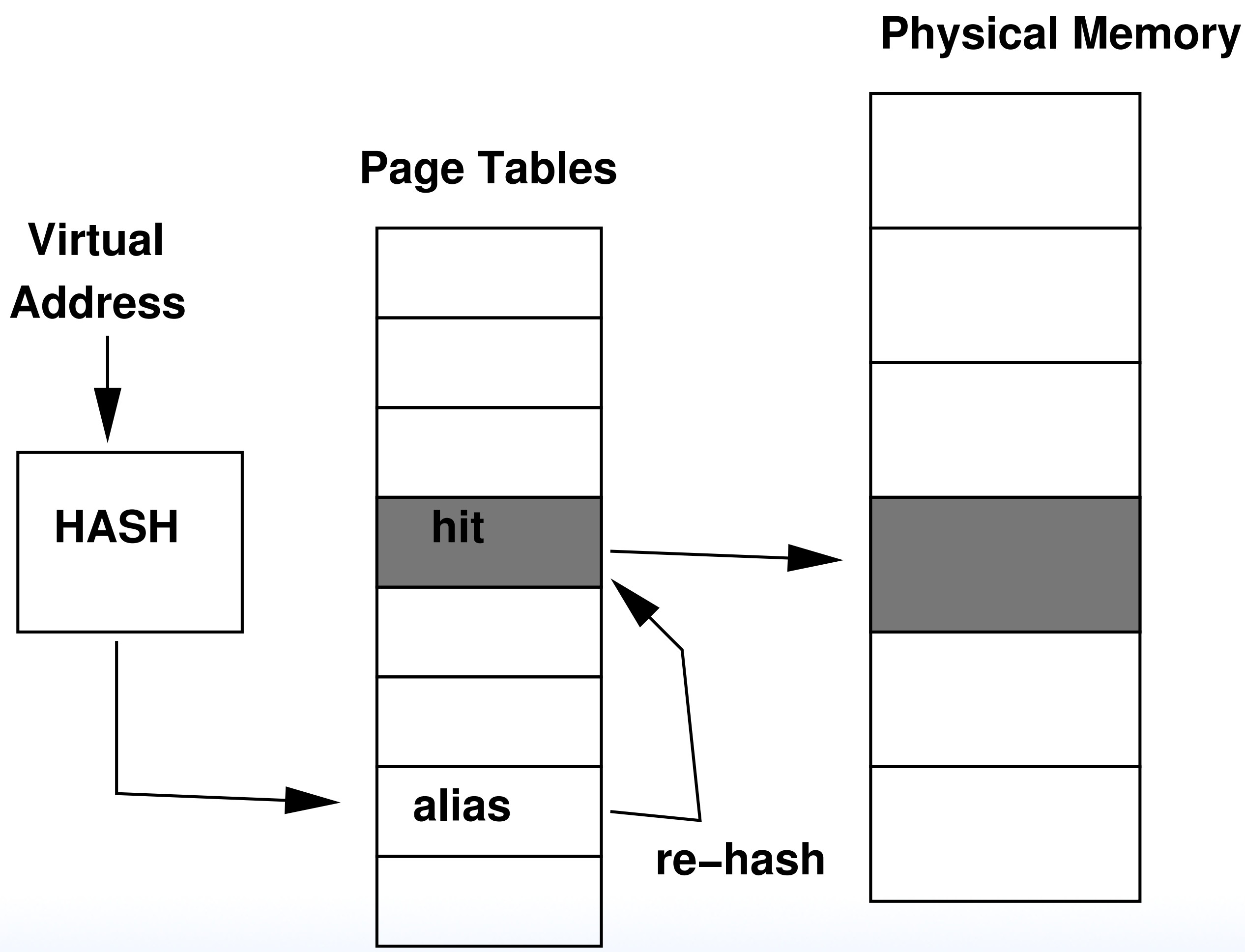
Hierarchical Page Table Diagram

- 32-bit x86 chips have hardware 2-level page tables
- ARM 2-level page tables

Inverted Page Table

- How to handle larger 64-bit address spaces?
- Can add more levels of page tables (4? 5?) but that becomes very slow
- Can use hash to find page. Better best case performance, can perform poorly if hash algorithm has lots of aliasing.

Inverted Page Table Diagram



Walking the Page Table

- Can be walked in Hardware or Software
- Hardware is more common
- Early RISC machines would do it in Software. Can be slow. Has complications: what if the page-walking code was swapped out?

TLB

- Translation Lookaside Buffer
  (Lookaside Buffer is an obsolete term meaning cache)
- Caches page tables
- Much faster than doing a page-table walk.
- Historically fully associative, recently multi-level multi-way
- TLB shootdown – when change a setting on a mapping

and TLB invalidated on all other processors

Flushing the TLB

- May need to do this on context switch if doesn't store ASID or ASIDs run out.
- Sometimes called a "TLB Shootdown"
- Hurts performance as the TLB gradually refills
- Avoiding this is why the top part is mapped to kernel under Linux

What happens on a memory access

- Cache hit, generally not a problem, see later. To be in cache had to have gone through the whole VM process. Although some architectures do a lookup anyway in case permissions have changed.

- Cache miss, then send access out to memory

- If in TLB, not a problem, right page fetched from physical memory, TLB updated

- If not in TLB, then the page tables are walked

- It no physical mapping in page table, then page fault happens

What happens on a page fault

- Walk the page table and see if the page is valid and there
- "minor" – page is already in memory, just need to point a PTE at it. For example, shared memory, shared libraries, etc.
- "major" – page needs to be created or brought in from disk. Demand paging.
  Needs to find room in physical memory. If no free space

available, needs to kick something out. Disk-backed (and not dirty) just discarded. Disk-backed and dirty, written back. Memory can be paged to disk. Eventually can OOM. Memory is then loaded, or zeroed, and PTE updated. Can it be shared? (zero page)

- "invalid" – segfault