ECE 598 – Advanced Operating Systems
Lecture 12

Vince Weaver

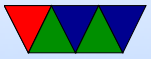
http://www.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

24 February 2015

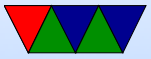
Announcements

- Homework #1 and #2 grades were sent out
- Don't forget Homework #3
- Also don't forget Midterm Exam on Thursday



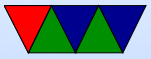
HW1 Review – code

- Part1 I meant to have it blink forever, just like the C code, but I see I worded it poorly.
- Comment your code!



HW2 Review

- Makefile didn't auto-include README.
- Serial: Divider settings for 115200 were 1 / 40 (or 41)
- Boot message to avoid stairstep, use \r as well as n.
- Hex printing, like decimal but %16 and special case if above ten to print letter.
  Capital vs lowercase (%X vs %x) doesn't really matter.
  Don't modify the 10 char printing routine (that's a 4GB limit). If you do it will double-print things.



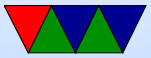
- Atags. Most of code there for you.
  Meant to ask you print memory in MB but forgot.
  Can get model number from the cmdline, board revision.

- Why serial? It's simple.

- Why parity? To catch errors.

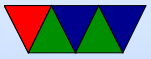
- Why not used? Wasted bandwidth (nearly 10%) and assume cables are good, etc.
  Unrelated to parity memory.

- What is inline assembly?



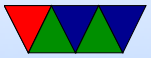
Lets you include assembly (which can be faster) and also lets you insert code that doesn't map directly to C (swi instructions, etc).

- Why no strtok()
  Because we don't have a C library. It is also considered semi-unsafe to use for various reasons, but that's not why we can't easily use it.

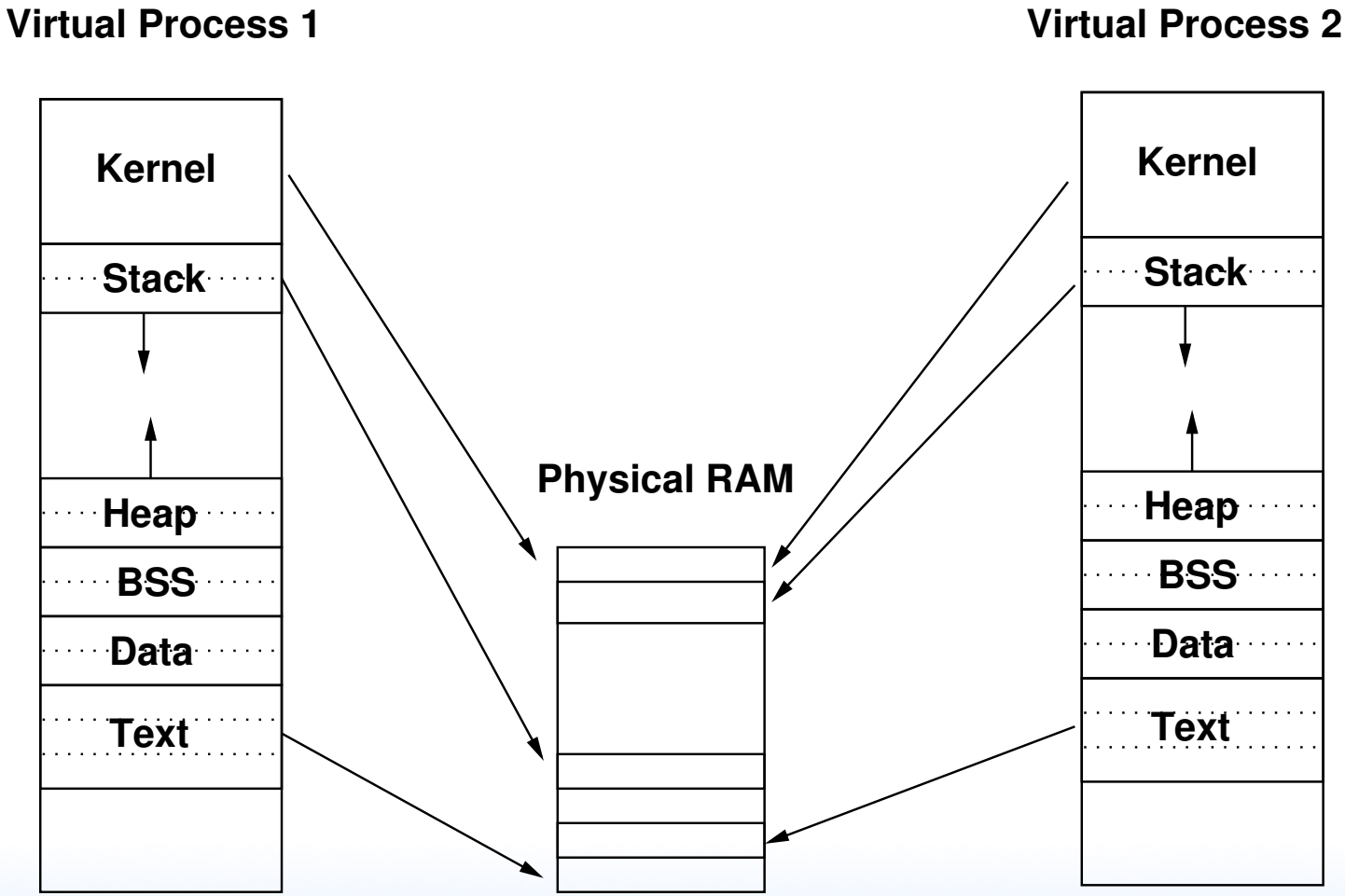


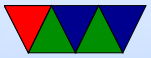
Virtual Memory Redux

- 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



Diagram

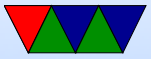

Virtual Process 1
Virtual Process 2
Kernel
Stack
Heap
BSS
Data
Text
Physical RAM
Kernel
Stack
Heap
BSS
Data
Text



Memory Management Unit

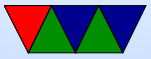
Can run without MMU. There's even MMU-less Linux.

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



Page Table

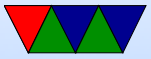
- Collection of Page Table Entries (PTE)
- Some common components: ID of owner, Virtual Page Number, valid bit, location of page (memory, disk, etc), protection info (read only, etc), page is dirty, age (how recent updated, for LRU)



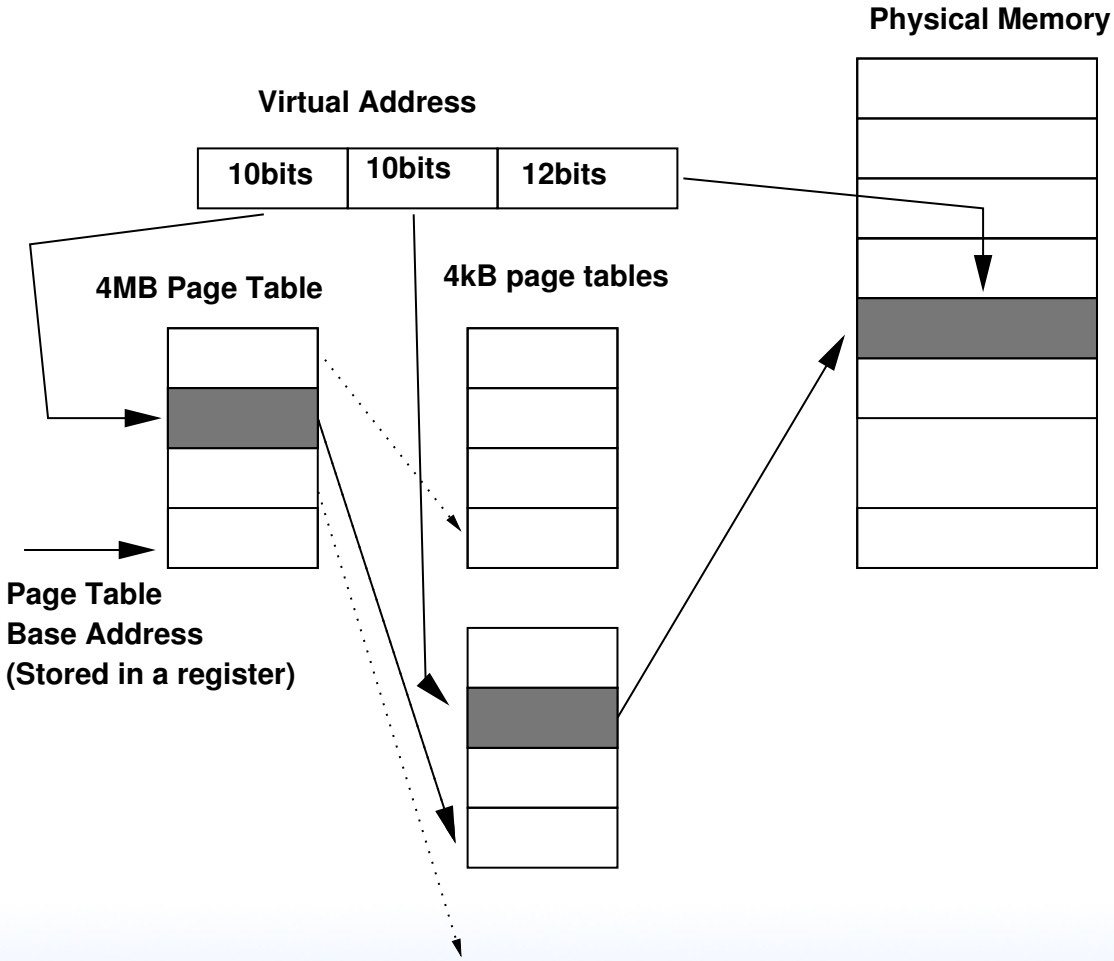
Hierarchical Page Tables

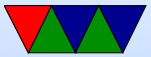
- With 4GB memory and 4kb pages, you have 1 Million pages per process. If each has 4-byte PTE then 4MB of page tables per-process. Too big.

- 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.



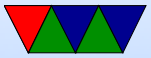
Hierarchical Page Table Diagram





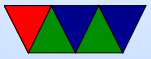
Hierarchical Page Table Diagram

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

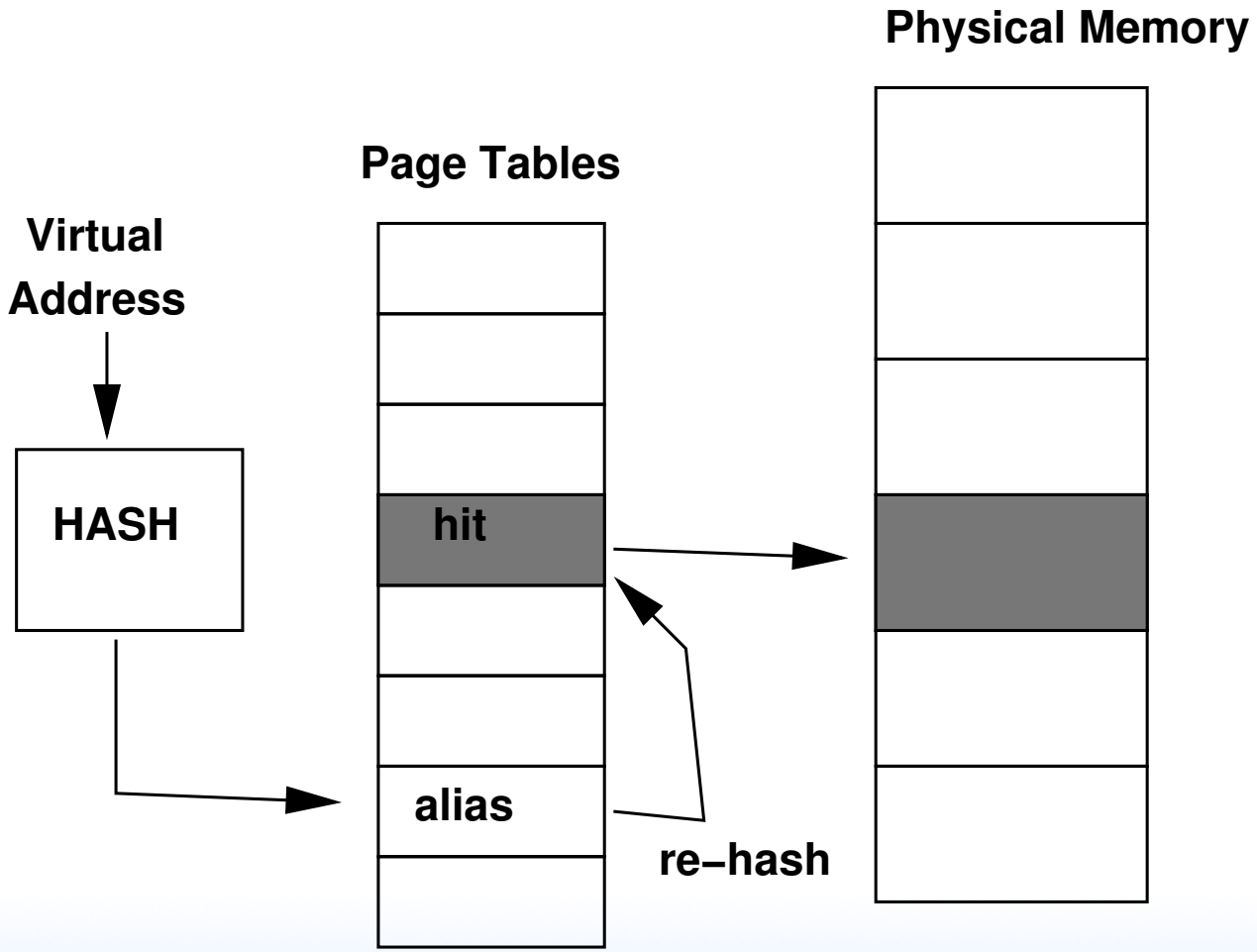


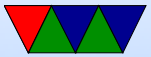
Inverted Page Table

- 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.



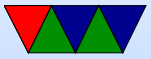
Inverted Page Table Diagram





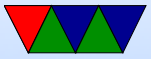
Walking the Page Table

- 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?

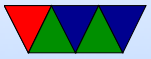


TLB

- 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

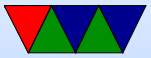


and TLB invalidated on all other processors



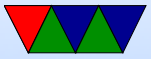
Flushing the TLB

- May need to do this on context switch if doesn't store ASID or ASIDs run out (ASID=Address Space ID)
- 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



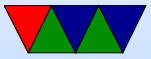
What happens on a memory access

- 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



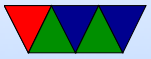
What happens on a page fault

- 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



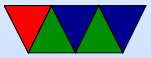
available, needs to kick something out. Disk-backed (and not dirty) just discarded. Disk-backed and dirty, written back. Memory can be paged to disk. Eventually can OOM. Memory is then loaded, or zeroed, and PTE updated. Can it be shared? (zero page)

- "invalid" – segfault



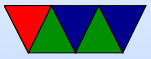
What happens on a fork?

- Do you actually copy all of memory?
  Why would that be bad? (slow, also often exec() right away)
- Page table marked read-only, then shared
- Only if writes happen, take page fault, then copy made
  Copy-on-write (COW)

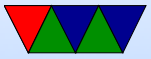


Large Pages

- Another way to avoid problems with 64-bit address space
- Larger page size (64kB? 1MB? 2MB? 2GB?)
- Less granularity. Potentially waste space
- Fewer TLB entries needed to map large data structures
- Compromise: multiple page sizes.
  Complicate O/S and hardware. OS have to find free blocks of contiguous memory when allocating large page.

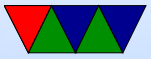


- Transparent usage? Transparent Huge Pages? Alternative to making people using special interfaces to allocate.

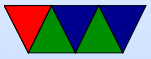


Having Larger Physical than Virtual Address Space

- 32-bit processors cannot address more than 4GB
  x86 hit this problem a while ago, ARM just now
- Real solution is to move to 64-bit
- As a hack, can include extra bits in page tables, address more memory (though still limited to 4GB per-process)
- Linus Torvalds hates this.

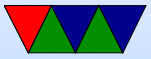


- Hit an upper limit around 16-32GB because entire low 4GB of kernel addressable memory fills with page tables



ARM 1176 (ARMv6) MMU

- See Chapter 6 of the ARM1176 Technical Reference Manual `http://infocenter.arm.com/help/topic/com.arm.doc.ddi0301h/DDI0301H_arm1176jzfs_r0p7_trm.pdf`
- page sizes: 4KB, 64KB, 1MB, and 16MB
- you can specify access permissions for 64KB large pages and 4KB small pages separately (Subpages)
- one 64-entry unified TLB and a lockdown region of eight

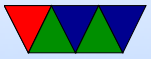


entries

- you can mark entries as a global mapping, or associated with a specific application space identifier to eliminate the requirement for TLB flushes on most context switches

- access permissions extended to enable Privileged read-only and Privileged or User read-only modes to be simultaneously supported

- hardware page table walks



- separate Secure and Non-secure entries and page tables
- Small fast microTLB. 1 cycle. Has ASID
- Larger slower main TLB, 64-entry 2-way
- Enable MMU by writing to special register in CP15