CS 3204
Operating Systems

Lecture 23

Godmar Back

Announcements

- Project 2 due Fri March 24
  - 2 day extension to stay in synch with other CS 3204 sections
- Midterm Fri March 24
  - We start at 10:00am!
- Project 3 page table design document due Wed March 29
  - See project page

Recap

- Page tables store mapping information from virtual to physical addresses, or to find non-resident pages
  - Conditioned by process id, current mode (user/kernel) and kind of access (read/write/execute)
- TLBs cache such mappings
- Page tables are consulted when TLB miss occurs
  - Either all software, or in hardware
- OS must maintain its page table(s) and, if hardware TLB reload is used, the page table (on x86 aka "page directory + table") that is consulted by MMU
  - These two tables may or may not be one and the same
- The OS page table must have sufficient information to load a page's content from disk

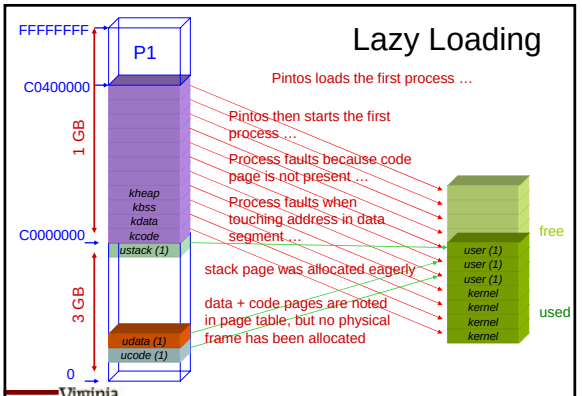
Virtual Memory

Paging Techniques

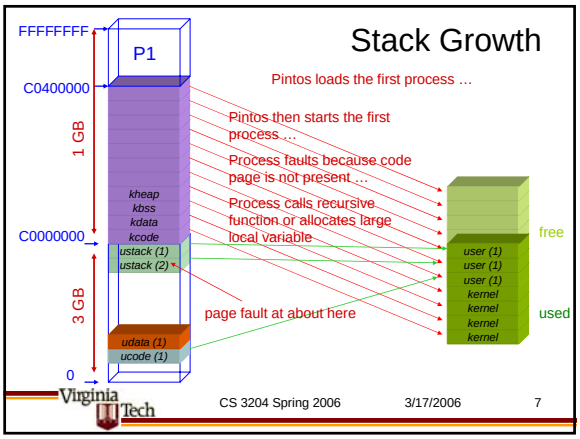
Demand paging

- Idea: only keep data in memory that's being used
  - Needed for virtualization – don't use up physical memory for data processes don't access
- Important optimization:
  - Lazy (on-demand) loading of pages the first time they're accessed

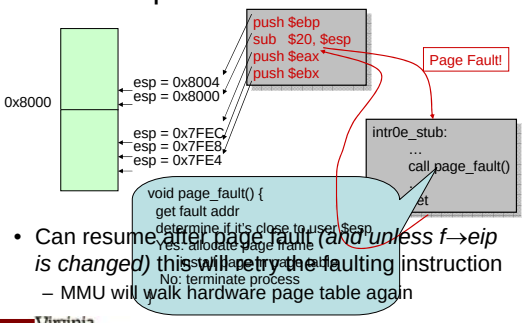
Lazy Loading



Stack Growth



Microscopic View of Stack Growth



- Can resume after page fault *(and unless f→eip is changed)* this will retry the faulting instruction
  - MMU will walk hardware page table again

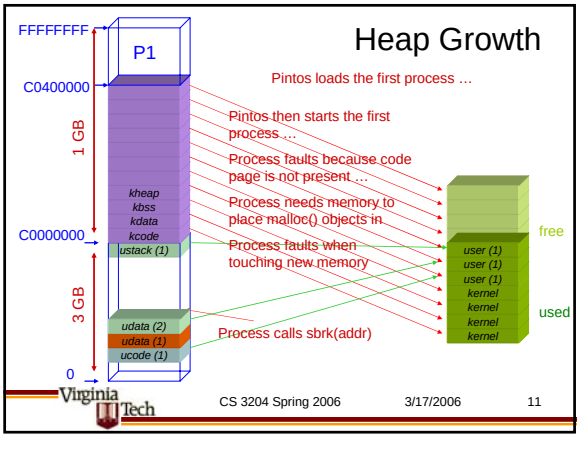
Fault Resumption

- Requires that faulting CPU instruction be restartable
  - Most CPUs are designed this way
- Very powerful technique
  - Entirely transparent to user program: user program is frozen in time until OS decides what to do
- Can be used to emulate lots of things
  - Programs that just ignore segmentation violations (!?) (here: resume with next instruction – retrying would fault again)
  - Subpage protection (protect entire page, take fault on access, check if address was to an valid subpage region)
  - Virtual machines (vmware, qemu – run entire OS on top of another OS)
  - Garbage collection
  - Distributed Shared Memory

Distributed Shared Memory

- Idea: allows accessing other machine's memory as if it were local
- Augment page table to be able to keep track of network locations:
  - local virtual address → (remote machine, remote address)
- On page fault, send request for data to owning machine, receive data, allocate & write to local page, map local page, and resume
  - Process will be able to just use pointers to access all memory distributed across machines – fully transparent
- Q.: how do you guarantee consistency?
  - Lots of options

Heap Growth



mmap()