Linux VM System

Chapter 23

Previously in CS212…

- We've spent a bunch of time looking at all the different ways to virtualize memory

- Now we look at one specific example of an implementation of virtual memory in the Linux operating system.

Linux Address Virtual Space (32-bit)

- Page 0 is left Invalid for detecting **null-pointer** accesses
- Two different types of kernel addresses
  - **Kernel logical** – normal virtual address space that can be allocated and holds data structures, pages tables, per-process kernel stacks, etc.
    - Maps directly to the first portion of physical addresses: 0xC0000000 => 0x00000000
    - Useful for direct memory access (I/O to and from devices via specific memory locations)
  - **Kernel virtual** - virtually contiguous, but physically non–contiguous. Easy to allocate, so used for large buffers and lets the kernel access more than the 1 GB of RAM reserved for kernel memory (useful in 32-bit).

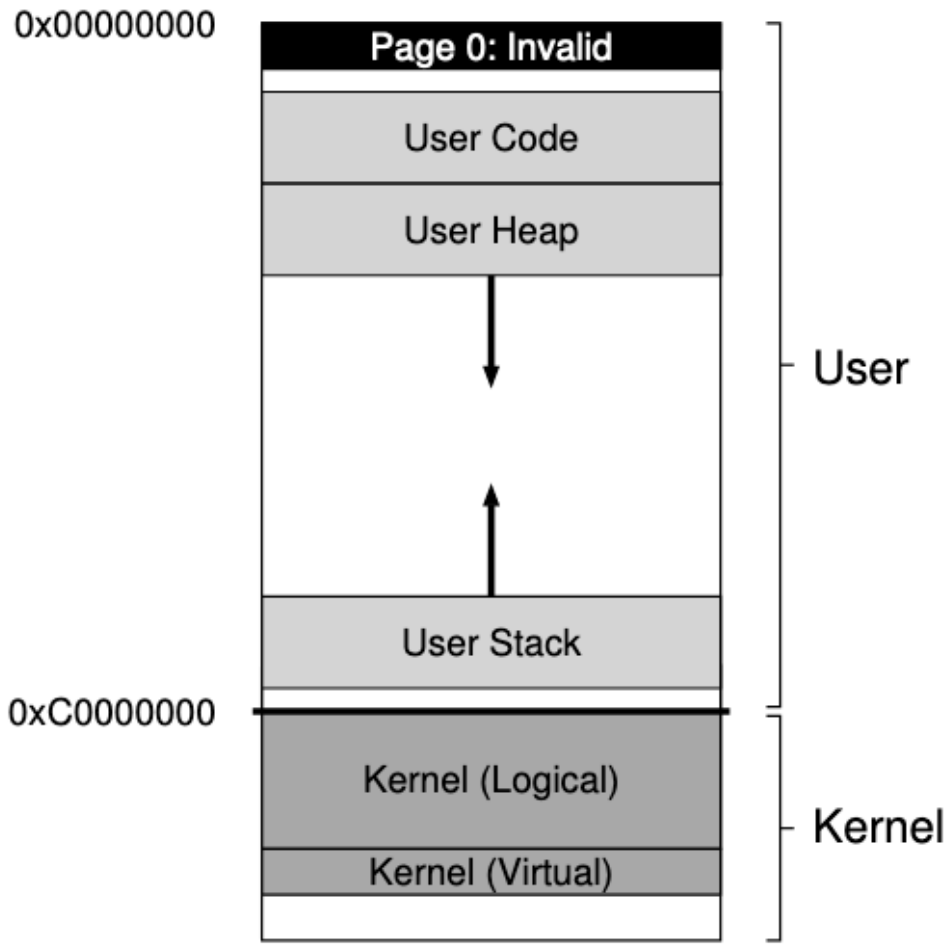
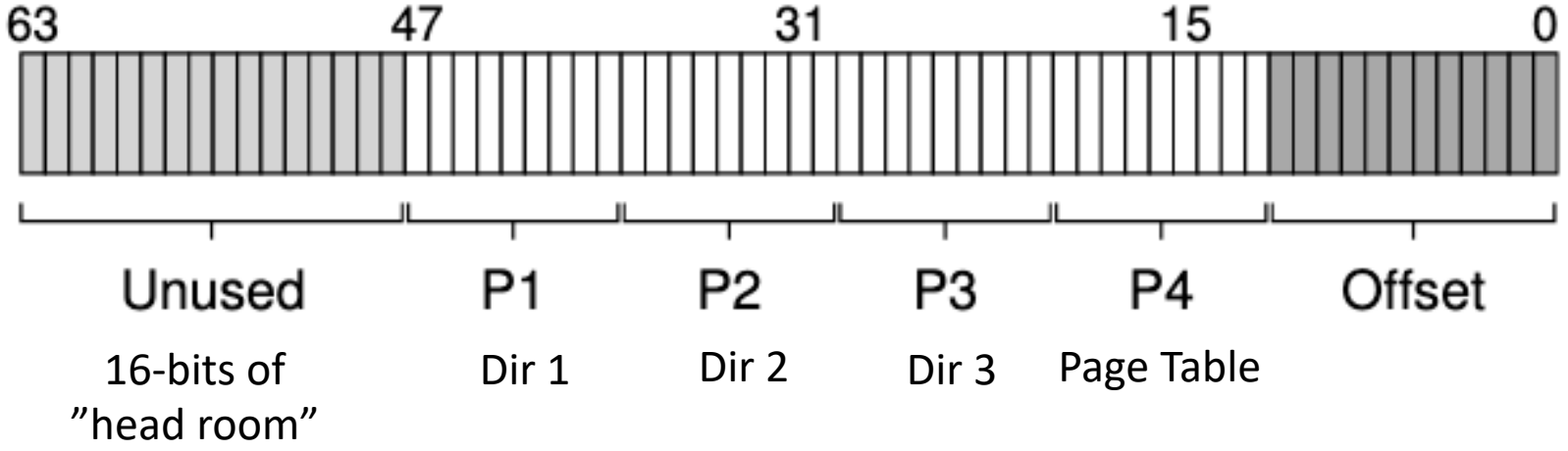


Figure 23.2: **The Linux Address Space**

Page Table Structure (64-bit)

- A 32-bit address space has become too limiting
  - Ram >= 4GB cannot be addressed in a 32-bit system (some addresses are used for hardware, so you never get the full 4GB)
- Why most operating systems are now 64-bit

- This is the Virtual Address format for 64-bit Linux:



Large Page Support

- Linux can support not just 4KB pages, but 2MB to even 1GB pages in hardware
- Why?
  - Larger pages == less mappings in the page table
  - **** Better TLB Performance ****
- Remember that the TLB cached page table mappings, if you have larger pages, you can reduce the number of page table entries and hold more data
  - Less page table entries means a better chance of having TLB Hits for better performance
- Downsides:
  - Internal fragmentation
  - Swapping large pages can increase the amount of I/O needed

Page Cache

- Page cache entries come from:
  - Memory-mapped files – mapping virtual memory directly to another part of memory
  - File data and metadata from devices (read() and write())
  - Anonymous memory - process heap and stack pages
- Page cache hash table holds entries
- If entries are dirty, they are periodically written out to their source by a background thread
  - This takes place after a certain period or if too many pages are dirty

Cache Replacement

- Standard LRU can be subverted with access to a large file
  - LRU kicks out other pages to make room for the large file's pages
  - Worse yet if those pages aren't needed multiple time
- 2Q Replacement is used instead
  - Has two queues: inactive and active
- When something is accessed for the first time it goes in the first inactive queue (A1)
- If that content is needed again, it goes in the second active queue (Aq)
- Replacement candidates are taken from the inactive queue and periodically, active queue pages are demoted to the inactive queue

Next Time

- Buckle up…it’s time for concurrency and threading