Swap Space

Chapter 21

Previously in CS212…

- With multi-level page tables, we can drastically shrink the amount of memory required for a page table
  - On a TLB hit, we essentially pay no penalty, on a miss, we must perform extra expensive memory lookup operations
- While we have saved space, there are lots of processes being run both by the OS and the user
- We have assumed so far that everything can fit into RAM
  - What happens when we can’t fit everything?

Memory is not infinite

- We need a place to put pages that aren’t in high demand to free up space
  - Needs to have a larger capacity than RAM (Primary Memory/Storage)
  - This means it will be slower as well
  - HDD/SDD (Secondary Memory/Storage)

- We also need to maintain the illusion of single large address space across primary and secondary storage locations

Swap Space

- A portion of reserved disk space where we can offload pages from and restore them to memory
  - We ***"Swap"*** them in and out
- OS writes and reads from swap space and will need to remember the disk address of the given page
- The size of the swap space ultimately determines the maximum number of memory pages the system can use at a given time
- Swap space is NOT the only on-disk location for swapping
  - Program binaries are generally on disk can have their code loaded into memory
  - We can reuse the data at this location and not swap space for this as the binaries are not likely to change

Example

If we only had enough space in physical memory for four pages, we could utilize swap space to off load the pages increasing the number of pages we can have “in memory”.

How do we know when data is in swap space and where it is located on disk?

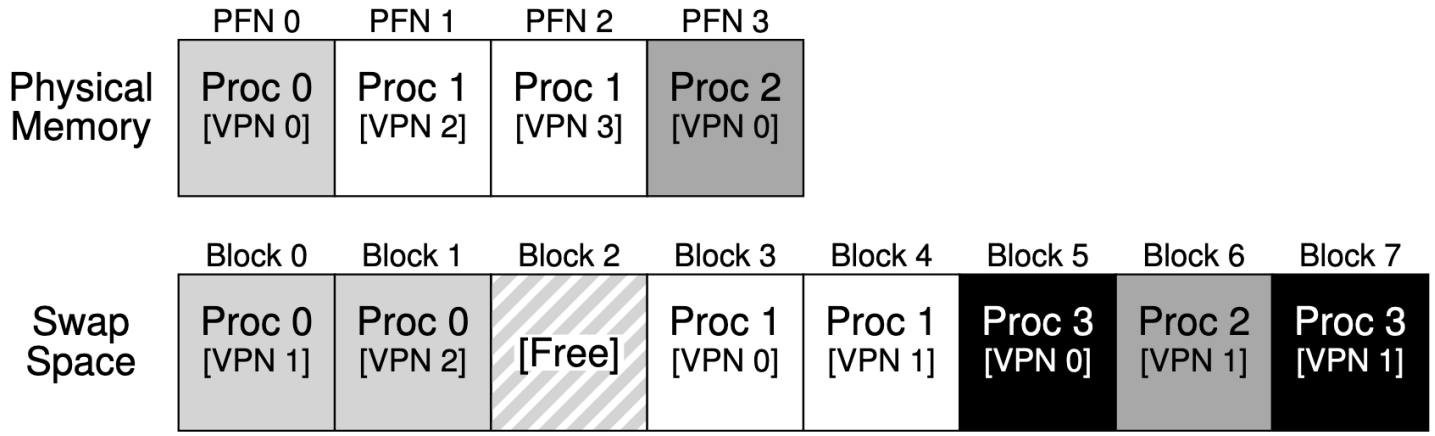

Figure 21.1: **Physical Memory and Swap Space**

Present Bit

- Remember that the page table entry stores some information about the data contained in the page

PFN	valid	prot	present	dirty
10	1	r-x	1	0

- When trying to access a page from the page table we can check the **present bit** to make sure that the page is physically present in RAM

- If the page is not present, then we have a **page fault**

Page Fault

- When a page fault occurs, we trap to the OS and run a page-fault handler
  - Virtually all page faults are handled by software
- How does the OS know where to find the page?
  - The disk address is placed in the PTE as the PFN
- While the OS works to retrieve the page from disk, the process is blocked
  - This will take quite a bit of time, so it’s best to overlap this request with execution of another processes

Page Faults and the TLB

- If the TLB miss occurs, we have some additional checks to do now with swap space
- If the page is **both** present and valid, the TLB handler finds PFN from the PTE and retries the instruction
- If the page is valid, but not present, then the page fault handler (OS) needs to be run find a free page (swapping out others if necessary) and retrieve the needed page from swap space (disk)
- If the page is not valid, nothing else matters and we have an invalid access (process is likely terminated)

To swap or not to swap

- The OS generally does not wait until all of memory is full to start swapping pages
  - We use high watermark (HW) and low watermark (LW) thresholds
- A page-replacement policy determines how pages are exchanged
  - This is critical for high performance
  - Bad replacement decisions could cause programs to run orders of magnitude slower
- Swapping pages is generally done in a group or cluster to optimize the process of reading/writing to disk
  - Handled by the swap/page daemon

Next Time

- We looked at mechanisms to avoid having to physically hold all the information from our processes in memory at once by swapping content to and from the disk

- Next time, we discuss policies used to determine which pages are removed and added to memory when it’s time to swap