CS 471 Operating Systems

Yue Cheng

George Mason University

Spring 2019

Midterm II

- Wednesday, April 10, 9:00am – 10:15am
  - 75 min, closed book, closed note

- Covering topics from lec-3a to lec-4d (**not** including “Workload Examples” at the end of lec-4d)
  - CPU scheduling: FIFO/SJF/SRTF/RR/Priority/MLFQ
  - Memory management: Paging/PT/TLB/Page Replacement

CPU Scheduling Policies

- FIFO
  - How it works?
  - Its inherent issues (why we need SJF)?
- SJF
  - How it works?
  - Any limitations (why we need SRTF)?
- SRTF (preemptive SJF)
  - How it works? How it solves SJF’s limitations?
  - Optimality of SJF & SRTF (under what scenarios)
    - Any implication or problem with SJF & SRTF?

Various Metrics

- Average waiting time
- Average turnaround time
- Average response time

- How to calculate each metric under a specific schedule (Gantt chart)

CPU Scheduling Policies (cont.)

- RR
  - How it works?
  - Why it is needed (compared to SJF & SRTF)?
    - The turnaround time vs. response time tradeoff
  - Impact of quantum tuning on turnaround time
- Priority
  - How it works?
  - Problems of Priority sched and solution?
- MLFQ
  - How it works?
  - Rules that are discussed in lecture. Which rule solves what problem?

Virtual Memory Accesses

- Virtual memory accesses involved in one assembly instruction
  - Fetch instruction at addr XYZ
  - Mem load/store if any

- Segfault vs. page fault
  - Basic concepts of memory segmentation

Paging

- Virtual addresses and physical addresses
  - VPN, PFN, page offset

- Virtual to physical address translation
  - (Basic) linear page table: index of the array as VPN
    - Each PTE contains PFN and other status info

- The baseline linear page table has problems
  - Performance problem – TLB to the rescue
  - Memory consumption problem – various solutions

Translation Lookaside Buffer (TLB)

- Address translation steps
  - Each virtual address access requires **two** physical memory accesses: one for PTE access, one for data access
  - Two much performance overhead!

- Having TLB as a hardware cache to accelerate address translation
  - PT access workloads exhibit **data locality**
  - To save the 1st memory access for fetching PTE

PT-related calculation

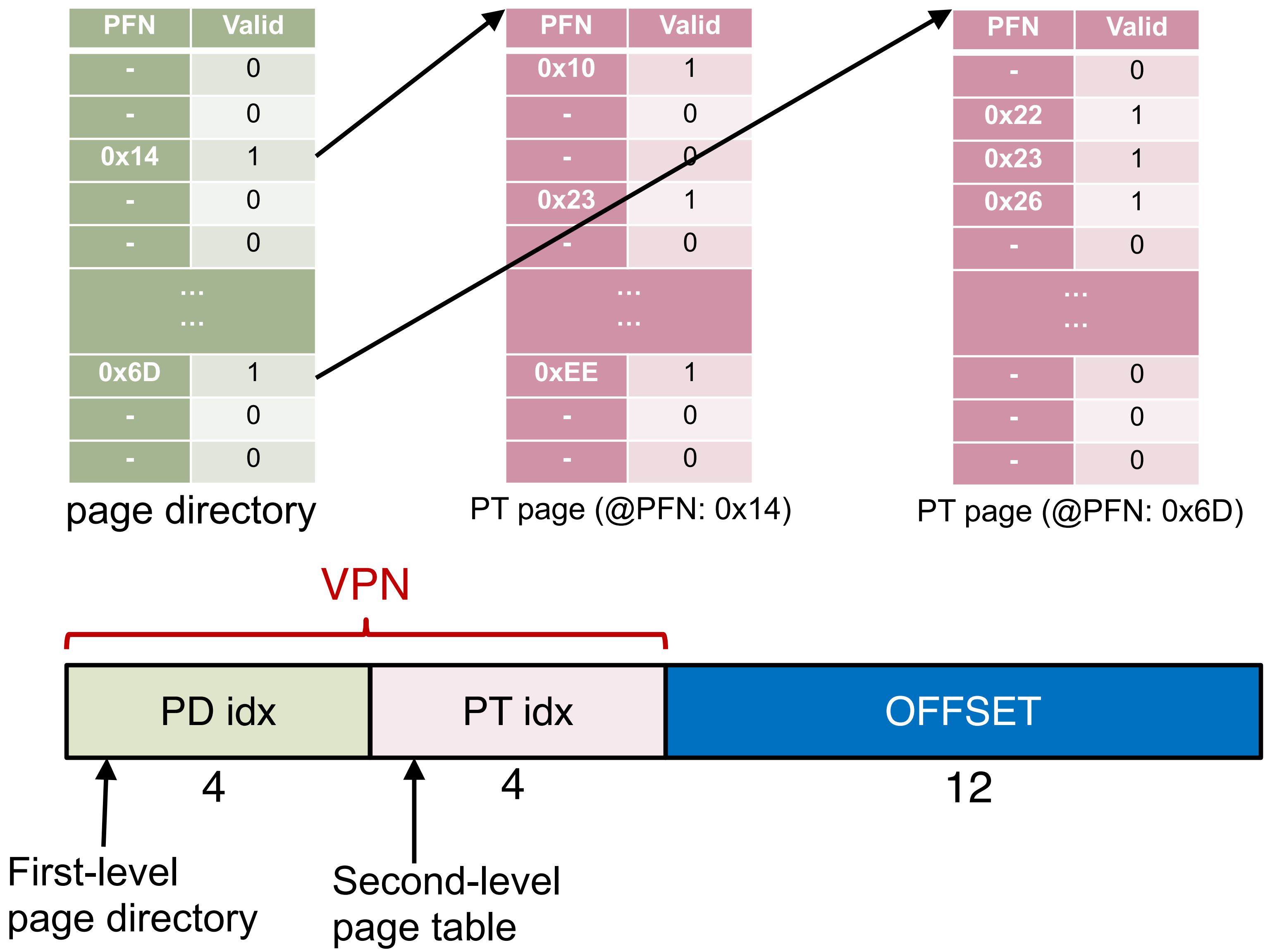
- Assume a linear page table array

- Virtual address: VPN + offset
  - Assume a 32-bit virtual address space
  - 4KB pages: offset has 12 bits
  - How many pages: 2^{20} pages since VPN has 20 bits
  - How much memory: **4MB** assuming each page-table entry is of 4 bytes
    - 2 ^ (32-log(4KB)) * 4 = 4MB

PT's are Too Big

- Approach 1: Linear Inverted Page Table
  - Whole system maintains only **one** PT
  - Performs a whole-table linear search using `pid+VPN` to get the index
    - The index corresponds to the PFN: An "**Inverted**" process compared against the baseline linear page table array
- Approach 2: Hash Inverted Page Table
  - Leverages hashing to reduce the algorithmic time complexity from `O(N)` to `O(1)`
- Approach 3: Multi-Level Page Table
  - Uses hierarchy to reduce the overall memory usage

Multi-Level Page Tables (cont.)



PFN	Valid
-	0
-	0
0x14	1
-	0
-	0
… …	
0x6D	1
-	0
-	0

page directory

PFN	Valid
0x10	1
-	0
-	0
0x23	1
-	0
… …	
0xEE	1
-	0
-	0

PT page (@PFN: 0x14)

PFN	Valid
-	0
0x22	1
0x23	1
0x26	1
-	0
… …	
-	0
-	0
-	0

PT page (@PFN: 0x6D)

Page Replacement Policies

- FIFO
  - Why it might work? Maybe the one brought in the longest ago is one we are not using now
  - Why it might not work? No real info to tell if it's being used or not
  - Suffers "**Belady's Anomaly**"
- Random
  - Sometimes non intelligence is better
- OPT
  - Assume we know about the future
  - Not practical in real cases: **offline** policy
  - However, can be used as a **best case baseline** for comparison purpose
- LRU
  - Intuition: we can't look into the future, but let's look at past experience to make a good guess
  - Our "bet" is that pages used recently are ones which will be used again (**principle of locality**)

Average Memory Access Time

- Metric to quantify the effective memory access performance
  - Assume a more generic two-tier memory (or storage) model
  - `AMAT = Hit_rate*`T_{T1} `+ Miss_rate*`T_{T2}
  - T_{T1} : Performance cost given existence of Tier 1
  - T_{T2} : Performance cost given absence of Tier 1

- If Tier 1 is TLB, then Tier 2 can be memory
- If Tier 1 is memory, then Tier 2 can be disk