Final Review

CS 571: Operating Systems (Spring 2020)
Lecture 12

Yue Cheng

Some material taken/derived from:
- Wisconsin CS-537 materials created by Remzi Arpaci-Dusseau.

Final Exam Logistics

- Monday, **May 18, 7:20pm – 10:00pm**
  - 160 min, **open book, open notes**

- Covering topics from *lec-1 to lec-11*
  - CV, classic sync problems (~18%)
  - CPU job scheduling (~18%)
  - Memory management and paging (~18%)
  - Cache replacement policies (~16%)
  - I/O and storage (~30%)

Final Exam Logistics (cont.)

- Like midterm, the final exam sheet will be available on BB (under "Assignment") for downloading at 7:20 pm

- Only pdf format will be provided
  - You can directly work on the pdf
  - Or, print out the pdf, write on printed papers, and scan to pdf with visible resolution

- Submission closes at 10 pm, so please make sure to submit before the deadline

Condition Variables

- CV: an explicit queue that threads can put themselves when some condition is not as desired (by waiting on that condition)

- `cond_wait(cond_t *cv, mutex_t *lock)`
  - assume the lock is held when `cond_wait()` is called
  - puts caller to sleep + **release** the lock (atomically)
  - when awaken, **reacquires** lock before returning

- `cond_signal(cond_t *cv)`
  - wake a single waiting thread (if >= 1 thread is waiting)
  - if there is no waiting thread, just return, **doing nothing**

Condition Variables (cont.)

- Traps when using CV
  - A **`cond_signal()`** may only wake one thread, though multiple are waiting
  - Signal on a CV with no thread waiting results in a lost signal

- Good rules of thumb when using CV
  - Always do wait and signal while holding the lock
  - Lock is used to provide mutual exclusive access to the shared variable
  - **`while()`** is used to always guarantee to re-check if the condition is being updated by other thread

Classic Problems of Synchronization

- Producer-consumer problem (CV-based version)
- Readers-writers problem
- Five dining philosophers problem

CPU Job Scheduling

- FIFO
  - How it works?
  - Its inherent issues (why we need SJF)?
- SJF
  - How it works?
  - Any limitations (why we need STCF)?
- STCF (preemptive SJF)
  - How it works? How it solves SJF's limitations?
- RR
  - How it works (time quantum or slice)?
  - Why it is needed (compared to SJF & STCF)?
    - The turnaround time vs. response time tradeoff

CPU Scheduling Metrics

- Average waiting time
- Average turnaround time

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

Memory Management: Addresses & PT

- Virtual addresses and physical addresses
  - VPN, PFN, page offset
  - Virtual address = VPN | offset

- Virtual to physical address translation
  - (Basic) linear page table: using VPN as index of array

Advanced Page Tables

- 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
- Approach 2: Hash inverted page table
  - Leverages hashing to reduce the time complexity from `O(N)` to `O(1)`
- Approach 3: Multi-level page table
  - Uses hierarchy to reduce the overall memory usage

Cache 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
- 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)

Cache Locality

- Spatial locality
  - Access to a single byte on disk brings in the whole page

- Temporal locality
  - Repetitive accesses to the same data

I/O and Storage Basics

- Disk scheduling policies
  - FIFO, SPTF, SCAN, C-SCAN, C-LOOK

- Hardware storage mediums
  - HDDs:
    - Internal mechanical pieces
    - Performance model: seek, rotate, data transfer
  - Flash SSDs:
    - Asymmetric read-write performance
    - Due to inherently different architecture

RAID

- Redundant array of inexpensive disks
  - Tradeoffs of different RAID configurations
  - RAID-0: No redundancy, perf-capacity upper bound
  - RAID-1: Mirroring
  - RAID-4: A disk is solely used for storing parity
  - RAID-5: Rotating parity across disks

Question Types

- Multi-choice questions
- Problem solving

Good Luck!