ECE 344: Operating Systems

Lecture 27

Quiz 3 Review

1.0.2

Jon Eyolfson

November 21, 2022

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License

The Quiz Covers 4 General Topics

- Scheduling
- Page Replacement
- Page Tables and TLB
- Virtual Memory

No sockets

Scheduling Involves Trade-Offs

We looked at few different algorithms:

- First Come First Served (FCFS) is the most basic scheduling algorithm
- Shortest Job First (SJF) is a tweak that reduces waiting time
- Shortest Remaining Time First (SRTF) uses SJF ideas with preemptions
- SRTF optimizes lowest waiting time (or turnaround time)
- Round-robin (RR) optimizes fairness and response time

Scheduling Gets Even More Complex

There are more solutions, and more issues:

- Introducing priority also introduces priority inversion
  - We saw dynamic priority scheduling
- Some processes need good interactivity, others not so much
- Multiprocessors may require per-CPU queues
- Real-time requires predictability
- Completely Fair Scheduler (CFS) tries to model the ideal fairness

Page Tables Translate Virtual to Physical Addresses

The MMU is the hardware that uses page tables, which may:

- Be a single large table (wasteful, even for 32-bit machines)
- Be a multi-level to save space for sparse allocations
- Use the kernel allocate pages from a free list
- Use a TLB to speed up memory accesses

Page Replacement Algorithms Aim to Reduce Page Faults

We saw the following:

- Optimal (good for comparison but not realistic)
- Random (actually works surprisingly well, avoids the worst case)
- FIFO (easy to implement but Bélády's anomaly)
- LRU (gets close to optimal but expensive to implement)

The pages go to a swap file on disk

A Bitmap Is What a Slab Allocator Uses

It’s a contiguous block of memory that tracks slots (1 bit each)

Assume we have a bitmap that’s 512 bytes large

- That’s 4096 bits

The bitmap could track 4096 slots (e.g. bit 500 tracks slot 500)

The Coremap Data Structure Manages the Physical Pages (Frames)

It keeps track of what processes can access which frames

  Recall: two processes could map different virtual pages to the same frame

For each frame it would track what processes (pid) have access to it

It would be able to free the frame if no processes have it mapped

Rough Memory Heirarchy Latencies

CPU cache: order of nanoseconds (1 ns)

Memory: 100s of nanoseconds, or single digit microseconds (1 μs)

SSD: double digit microseconds (10 μs)

Disk (Hard Drive): possibly up to milliseconds (1 ms)

Spatial vs Temporal Locality

Spatial locality means accesses are close to each other in memory
(accessed contiguously or almost)

Temporal locality means accesses are close to each other in a small time duration
(accessed together or almost)

Thrashing Is When There is a High Number of Page Faults

This occurs when processes cannot have their working sets of memory loaded

- A working set is an amount of memory a process requires in a time interval

Processes constantly have to evict pages they are currently using

This causes the system to be very slow and almost unresponsive

If the page fault frequency (PFF) is too high, the process
doesn’t have enough memory for its working set

Demand Paging

Mark a page in the page table as invalid
- This generates a page fault on the first access

The kernel can then allocate a page and update the page table with a valid entry

Page fault may happen because a process accesses an unmapped virtual address
- Instead send a segfault to the process

Note, the alternative is to load everything into memory when the program starts

Copy-on-Write (COW) is a Useful Technique

Similar to data races, we can share data as long as there's only reads

If two processes are sharing the same frame, and either could write (e.g. right after a fork)

Only copy the frame after one process modifies its page

We can also do other optimizations with read-only frames (e.g. one representing the executable code)

Instead of evicting page to disk, just re-read again when you need it

Virtual Memory Example

This problem is from Question 6 of the 2016 Fall Final

Done live