Slide 1

Administrivia

- Midterm grades mailed. Letter grades are approximate and conservative.

Slide 2

Recap — Paging

- Recall basic ideas of paging:
  - Divide address spaces into pages, memory into page frames; allocate memory page (frame) by page (frame).
  - Use page tables (one per process) to keep track of things.
  - Use MMU to translate program (virtual) addresses into memory locations — using page table for current process. Generate "page fault" interrupt if impossible.

    Notice that we get memory protection for free; can also get memory sharing. Related issue — might be nice to have "read-only" bit in page table.
  - Performance is an issue — MMU usually just points to start of page table — but can solve that with caching (TLB, practical because of "locality of reference").

Slide 3

Paging Recap, Continued

- Things to look at more:
  - Dealing with large address spaces.
  - Extending this idea to provide "virtual memory" (by extending swapping idea).

Slide 4

Minute Essay From Last Lecture

- Given a page size of 64K (2^{16}), 64-bit addresses, and 4G (2^{32}) of main memory, at least how much space is required for a page table?

  Number of entries is $2^{64}/2^{16} = 2^{48}$.

  Size of each is at least enough for page frame.

  Number of page frames is 2^{16}, so we need 16 bits for page frame number, plus a valid/invalid bit — 3 bytes.

  Total is 3×2^{48} — i.e., really big!

Slide 5

Large Address Spaces

- Clearly page tables can be big. How to make this feasible?
- One approach — multilevel page tables.
- Another approach — inverted page tables (one entry per page frame).

Slide 6

Paging and Virtual Memory

- Idea — if we don't have room for all pages of all processes in main memory, keep some on disk ("pretend we have more memory than we really do").
- Or a simpler view: All address spaces live in secondary memory / swap space / backing store, and we "page in" as needed (demand paging).
- Consider an example . . .

Slide 7

Page Tables, Revisited

- What do we need for each entry in a page table?
  - Page frame number.
  - Present/absent bit (was valid/invalid).
  - Protection bit(s).
  - "Modified since last page-in?" bit.
  - "Referenced recently?" bit.
  - "Okay to cache?" bit.
- Goal is to keep this somewhat minimal — mostly data the MMU needs.

  If present/absent bit says "absent", two cases — error and "page not in memory right now" — MMU should generate "page fault" interrupt, let page fault interrupt handler decide.

Slide 8

Page Fault Processing

- If MMU finds "absent" bit on, it generates a "page fault" interrupt.
- Interrupt handler must:
  - Decide whether page is invalid or just not in main memory.
  - If page is invalid, error — maybe terminate process.
  - If page is valid but not in main memory?
    Find a free page frame and schedule I/O to get page from disk.
    When disk I/O completes, retry instruction that caused page fault.
- What details do we need to fill in here?
  - How to keep track of pages on disk.
  - How to keep track of which page frames are free.
  - (How to "schedule I/O", but that's later.)
  - What to do if there aren't any free pages.

Slide 9

Keeping Track of Pages on Disk

- To implement virtual memory, need space on disk to keep pages not in main memory. Reserve part of disk for this purpose (“swap space”); (conceptually) divide it into page-sized chunks. How to keep track of which pages are where?
- One approach — give each process a contiguous piece of swap space. Advantages/disadvantages?
- Another approach — assign chunks of swap space individually. Advantages/disadvantages?
- Either way — processes must know where “their” pages are (via page table and some other data structure), operating system must know where free slots are (in memory and in swap space).

Slide 10

Page Fault Processing, Revisited

- What happens again during a page fault?
- Interrupt handler examines process tables, etc., to decide whether page is “paged out” or invalid.
- If page is “paged out”, page it in and try again:
  - Try to find a free frame. If none, pick one to steal (discuss later how to choose). If it needs to be saved to disk, start I/O to do that. Update process table, etc., for “victim” process. Block process until I/O is done.
  - When we have a free frame, start I/O to bring needed page in from swap space. Block process until done.
  - Update process table, etc., for process that caused the page fault, and restart it at instruction that generated page fault.

Slide 11

Page Replacement Algorithms

- “Find a free page frame” would be easy if the current set of processes aren’t taking up all of main memory, but what if they are?

  Must steal a page frame from someone. How to choose one? “page replacement algorithm” — several choices.
- What makes a “good” p.r.a.?

Slide 12

Minute Essay

- How did the midterm compare to your expectations? easier or more difficult? shorter or longer? topics?