

CSE 120

Summer, 2005

Day 5
Virtual Memory

Instructor: Neil Rhodes

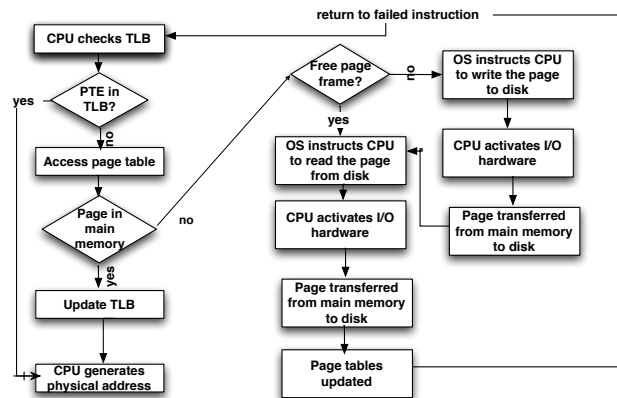
Page Fault Handling

MMU generates Page Fault (protection violation or page not present)

- Save registers
- Figure out virtual address that caused fault
 - Often in hardware register
- If protection problem, signal or kill process
- If no free page table entry, evict a page from memory (which one?)
 - If modified, write to backing store (dedicated paging space or normal file)
 - Keep disk location of this page (**not** in page table, but some other data structure).
 - MMU doesn't need to know disk location
 - Suspend faulting process (resume when write is complete)
- Read data from backing store for faulting page
 - From backing store or application code or fill-with-zero
 - Suspend faulting process (resume when read complete)
- Update page table
- Restart instruction for faulting process
 - Must undo any partial effects

2

Paging and Translation Lookaside Buffer



3

Segmentation vs. Paging

	Segmentation	Paging
Need the programmer be aware the technique is being used?		
How many linear address spaces are there?		
Can the total address space exceed the size of phys. mem?		
Can procedures and data be distinguished and separately protected?		
Can tables whose size fluctuates be accommodated easily?		
Is sharing of procedures between users facilitated?		
Why was this technique invented?		

4

Combining Segmentation and Paging

Virtual address contains <Segment number, Page number, offset>

- Segment number used to reference segment table
 - Segment table entry contains
 - Page Table pointer: reference to page table
 - Bounds: used to check Page number + offset
 - Protections: read/write, for example
- Page table converts page number to page frame
 - Offset is used within page frame

Virtual address space of a process uses high bits as segment number

- With different protection and ranges

5

Page Replacement Policy

Resident Set Management

- How many page frames are allocated to each active process?
 - Fixed
 - Variable
- What existing pages can be considered for replacement
 - Local: only the process that caused the page fault
 - Global: all processes
- Cleaning policy
 - Pre-Cleaning: Write dirty pages out prospectively
 - Demand-Cleaning: Write dirty pages out only as needed
- Fetch policy
 - Demand paging
 - Prepaging. Load extra pages speculatively while you're loading others
 - Copy-on-write
 - Lazy duplicate of pages. For example, on fork, don't copy data page until write occurs.

Replacement Policy

- Which page, among those eligible, should be replaced
 - All policies want to replace those that won't be needed for a long time
 - Since most processes exhibit *locality*, recent behavior helps predict future behavior
- Eligibility may be limited based on locked frames
 - Kernel pages
 - I/O buffers in kernel space
 - Page frames with scheduled DMA

6

Page References

Assumption is that the sequence of page references exhibits *locality*

Reference string is list of page numbers used by program

- For example, <0 1 2 3 0 1 4 0 1 2 3 4>
- Consecutive references to the same page are removed
 - That page better still be in memory!
- Reference means read or write

7

Opt: the Optimal Page Replacement Policy

Swap out the page that will be used farthest in the future

- Difficult to implement:)

Example reference string: <2 3 5 3 1 5 1 0 5 1 0>

- Three page frames

	2	3	5	3	1	5	1	0	5	1	0

8

FIFO: First-In First-Out

Swap out the page that's been in memory the longest

- Works well for swapping out initialization code
- Not so good for often-used code

	2	3	5	3	1	5	1	0	5	1	0

9

FIFO: Belady's anomaly

For FIFO, adding extra page frames can cause *more* page faults

Example reference string: <0 1 2 3 0 1 4 0 1 2 3 4>

- Three page frames

	0	1	2	3	0	1	4	0	1	2	3	4

- Four page frames

	0	1	2	3	0	1	4	0	1	2	3	4

10

Clock (or Second-chance)

Choose the oldest page that hasn't been referenced

- Implementation:
 - Pages in circular list
 - R bit maintained by hardware in the PTE
 - HW: Whenever a PTE is accessed (read or write for that page), R bit is set to 1
 - SW: can set R bit to 0 or 1
 - Hand points to particular page. When a page is needed, it checks R bit of that page
 - If set, clear and move to next page
 - If not set, replace and move to next page

	2	3	5	3	1	5	1	0	5	1	0

11

Not Recently Used (NRU)

Like Clock, but rather than having only two states (R=0, R=1), has *four states*

- Use Referenced/Modify bits
- Remove from (in order):
 - Not Referenced, not Modified
 - Not Referenced, Modified
 - Referenced, not Modified
 - Referenced, Modified
- Implementation:
 - Step 1: search for not Referenced, not Modified. If found, done
 - Step 2: search for not Referenced, Modified. If found, done (During scan, set R to 0)
 - go back to Step 1 and 2 (will find some page frame, since R is now 0)

12

Least Recently Used (LRU)

Remove the page that has been unused the longest

Hardware

- Keep counter in PTE. Increment on use. Find PTE with lowest counter to evict

	2	3	5	3	1	5	1	0	5	1	0

13

Working-Set Model

$W(\Delta, t)$

- Developed by Deming
- Set of pages a process has accessed from time $t-\Delta$ to time t
 - t is virtual time (last t memory accesses)
- Δ is size of the window (a larger window means a possibly larger set of pages)
- Working set can grow and shrink over time

Idea for algorithm

- Monitor the working set of each process
- Shrink/Grow the page frames allocated to a process down/up to that of its working set
- If there is not enough space for the working set, swap this process to disk

Difficulties

- What size of Δ to use?
- Keeping track of working set very difficult

Approximation

- Monitor *page fault frequency* of process.
 - Exceed upper threshold: add page frame
 - Below lower threshold: remove page frame

14

Keeping Free Pages

Keeping some clean free pages makes page faults faster

- Don't need to run page replacement algorithm: just go to free list
- Only need to wait for page to be brought in (instead of first waiting for dirty page to be written out).

Retain contents of freed page frames

- If requested again, reuse page frame without I/O

Write modified page frames lazily

- Save in modified page list
- Write out in groups (based on disk locality)

15

Thrashing

What it is:

- Spending more time paging than doing real work

Why it happens:

- If the degree of multiprogramming gets too high, each process' working set is not resident
- With local replacement, the number of frames allocated to this process isn't enough. (Fighting within a process)
- With global replacement, one process causes pages from other processes working sets to be evicted. (Fighting among processes)

Solution

- Reduce the degree of multiprogramming. Swap processes out to disk

How to determine good degree of multiprogramming

- Look at utilization of page device (50% utilization optimal)
- Look at mean time between faults versus mean time to service a fault (equal maximizes CPU utilization)
- For clock algorithm, look at rate that hand scans through the clock
 - Too low
 - Few page faults: not many requests to move the pointer
 - Not scanning many pages per request: most pages not referenced
 - Too high
 - High fault rate
 - Scanning many pages per request: most pages are referenced

16

Memory-Mapped Files

A file can be mapped into an address space

- Pager must read/write from file, similar to the way it pages in from an executable

Processes can read and write using memory access rather than file read/file write

- Written data is cached in page frame
- Difficult to change EOF

Can be shared between processes

17

Page Sizes

Advantages of smaller page size

- Less internal fragmentation
 - On average, the address space of each process wastes P/2 space

Advantages of larger page size

- TLB covers more bytes (TLB size * P), so better TBL hit rate
- Smaller page tables (need address space/P PTEs)

As memory has become cheaper and address space has become larger, page sizes have increased

- 1970s: Vax: 512 bytes
- 1990s: PowerPC: 4KB
- 1990s: Pentium: 4KB or 4MB (defined per secondary page table)
- 1990s: MIPS: 16KB

18

Summary

Some page replacement algorithms are better than others

- $OPT \geq LRU \geq \text{Clock} \geq \text{FIFO}$

Locality is what makes VM (or any caching) work

- Physical memory is a cache for logical memory

Keep working set in memory

- Otherwise, thrashing

19