Virtual Memory Management

CS-3013 Operating Systems
Hugh C. Lauer

(Slides include materials from Modern Operating Systems, 3rd ed., by Andrew Tanenbaum and from Operating System Concepts, 7th ed., by Silbershatz, Galvin, & Gagne)
Caching issues

- When to put something in the cache
- What to throw out to create cache space for new items
- How to keep cached item and stored item in sync after one or the other is updated
- How to keep multiple caches in sync across processors or machines
- Size of cache needed to be effective
- Size of cache items for efficiency
- ...

Virtual Memory Management
Physical Memory is a Cache of Virtual Memory, so ...

- When to swap in a page
  - On demand? or in anticipation?
- What to throw out
  - Page Replacement Policy
- Keeping dirty pages in sync with disk
  - Flushing strategy
- Keeping pages in sync across processors or machines
  - Defer to another time
- Size of physical memory to be effective
  - See previous discussion
- Size of pages for efficiency
  - One size fits all, or multiple sizes?
Physical Memory as cache of Virtual Memory

- When to swap in a page
  - On demand? or in anticipation?
- What to throw out
  - Page Replacement Policy
- Keeping dirty pages in sync with disk
  - Flushing strategy
- Keeping pages in sync across processors or machines
  - Defer to another time
- Size of physical memory to be effective
  - See previous discussion
- Size of pages for efficiency
  - One size fits all, or multiple sizes?
Reading Assignment

• Tanenbaum:–
  • §3.4 — Page Replacement Algorithms
  • §3.5 — Design Issues for Paging Systems
  • §3.6 — Implementation Issues
  • §3.7 — Segmentation
VM Page Replacement

- If there is an unused frame, use it.
- If there are no unused frames available, select a victim (according to policy) and
  - If it contains a dirty page (M == 1)
    - write it to disk
    - Invalidate its PTE and TLB entry
    - Load in new page from disk (or create new page)
    - Update the PTE and TLB entry!
    - Restart the faulting instruction
- What is cost of replacing a page?
- How does the OS select the page to be evicted?
Page Replacement Algorithms

• Want lowest page-fault rate.
• Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.
• Reference string – ordered list of pages accessed as process executes

Ex. Reference String is A B C A B D A D B C B
The Best Page to Replace

• The best page to replace is the one that will *never* be accessed again

• Optimal Algorithm – *Belady’s Rule*
  – Lowest fault rate for any reference string
  – Basically, replace the page *that will not be used for the longest time in the future.*
  – Belady’s Rule is a yardstick
  – We want to find close approximations
Some Page Replacement Algorithms

- Not Recently Used
- First in, First out
- Second Chance
- Clock
- Not Frequently Used
- Aging
- Working Set
- WSClock
- ...
Page Replacement – NRU
(Not Recently Used)

• Periodically (e.g., on a clock interrupt)
  • Clear R bit from all PTE’s
• When needed, rank order pages as follows
  1. R = 0, M = 0
  2. R = 0, M = 1
  3. R = 1, M = 0
  4. R = 1, M = 1
• Evict a page at random from lowest non-empty class
  • Write out if M = 1; clear M when written
• Characteristics
  • Easy to understand and implement
  • Not optimal, but adequate in some cases
Typical Page Table Entry

- **Valid** bit gives state of this entry
  - says whether or not a virtual address is valid – in memory and VA range
  - If not set, page might not be in memory or *may not even exist!*
- **Reference** bit says whether the page has been accessed
  - it is set by hardware *whenever* a page has been read or written to
- **Modify** bit says whether or not the page is *dirty*
  - it is set by hardware during *every* write to the page
- **Protection** bits control which operations are allowed
  - read, write, execute, etc.
- **Page** frame number (PFN) determines the physical page
  - physical page start address
- **Other** bits dependent upon machine architecture
Page Replacement – FIFO
(First In, First Out)

• Easy to implement
  • When swapping a page in, place its page id on end of list
  • Evict page at head of list

• Page to be evicted has been in memory the longest time, but …
  • Maybe it is being used, very active even
  • We just don’t know

• A weird phenomenon:– Belady’s Anomaly
  • fault rate may increase when there is more physical memory!
FIFO Illustrating Belady’s Anomaly

![Graph showing number of page faults vs. number of frames. The x-axis represents the number of frames (1 to 7), and the y-axis represents the number of page faults (0 to 16). The graph shows a decrease in page faults as the number of frames increases.]

CS-3013, A-Term 2011
Virtual Memory Management
Page Replacement – FIFO
(First In, First Out)

• Easy to implement
  • When swapping a page in, place its page id on end of list
  • Evict page at head of list
• Page to be evicted has been in memory the longest time, but …
  • Maybe it is being used, very active even
  • We just don’t know
• A weird phenomenon:— Belady’s Anomaly
  • fault rate may increase when there is more physical memory!
• FIFO is rarely used in practice
Second Chance

- Maintain FIFO page list
- When a page frame is needed, check reference bit of top page in list
  - If R == 1 then move page to end of list and clear R, repeat
  - If R == 0 then evict page

- I.e., a page has to move to top of list at least twice
  - I.e., once after the last time R-bit was cleared

- Disadvantage
  - Moves pages around on list a lot (bookkeeping overhead)
Clock Replacement
(A slight variation of Second Chance)

• Create circular list of PTEs in FIFO Order
• One-handed Clock – pointer starts at oldest page
  – Algorithm – FIFO, but check Reference bit
    • If $R = 1$, set $R = 0$ and advance hand
    • evict first page with $R = 0$
  – Looks like a clock hand sweeping PTE entries
  – Fast, but worst case may take a lot of time
Clock Algorithm (illustrated)

Reference bits:
- 0
- 0
- 1
- 0
- ...
- 1
- 1

Pages:
- (a) circular queue of pages
- (b) circular queue of pages

Next victim:
- 1

Reference bits:
- 0
- 0
- 0
- 0
- ...
- 0
- 1

Pages:
- (a) circular queue of pages
- (b) circular queue of pages
Enhanced Clock Algorithm

- Two-handed clock – add another hand that is $n$ PTEs ahead
  - Extra hand clears Reference bit
  - Allows very active pages to stay in longer
- Also rank order the frames
  1. $R = 0, M = 0$
  2. $R = 0, M = 1$
  3. $R = 1, M = 0$
  4. $R = 1, M = 1$

Select first entry in lowest category
- May require multiple passes
- Gives preference to modified pages
Least Recently Used (LRU)

- Replace the page that has not been used for the longest time

Reference String: A B C A B D A D B C

LRU - 5 faults

A B C A B D A D B C

On the assumption that it is least likely to be needed again soon
LRU

• Past experience may indicate future behavior
• Perfect LRU requires some form of timestamp to be associated with a PTE on every memory reference !!!
• Counter implementation
  – Every page entry has a counter; each time page is referenced through this entry, copy the clock into the counter.
  – When a page needs to be changed, look at the counters to determine which to select
• Stack implementation – keep a stack of page numbers in a double link form:
  – Page referenced: move it to the top
  – No search for replacement
LRU Approximations

- **Aging**
  - Keep a counter for each PTE
  - Periodically (clock interrupt) – check R-bit
    - If $R = 0$ increment counter (page has not been used)
    - If $R = 1$ clear the counter (page has been used)
    - Clear $R = 0$
  - Counter contains # of intervals since last access
  - Replace page having largest counter value

- **Alternatives**
  - §§3.4.6-3.4.7 in Tanenbaum
When to Evict Pages
(Cleaning Policy)

- An OS thread called the *paging daemon*
  - wakes periodically to inspect pool of frames
  - if insufficient # of free frames
    - Mark pages for eviction according to policy, set valid bit to zero
    - Schedule disk to write dirty pages
  - on page fault
    - If desired page is *marked* but still in memory, use it
    - Otherwise, replace first clean marked page in pool

- Advantage
  - *Writing out dirty pages is not in critical path to swapping in*
Physical Memory as cache of Virtual Memory

- When to swap in a page
  - On demand? or in anticipation?
- What to throw out
  - Page Replacement Policy
- Keeping dirty pages in sync with disk
  - Flushing strategy
- Keeping pages in sync across processors or machines
  - Defer to another time
- Size of physical memory to be effective
  - See previous discussion
- Size of pages for efficiency
  - One size fits all, or multiple sizes?
What to Page in

- Demand paging brings in the faulting page
  - To bring in more pages, we need to know the future
- Users don’t really know the future, but a few OSs have user-controlled pre-fetching
- In real systems,
  - load the initial page
  - Start running
  - Some systems (e.g. WinNT & WinXP) will bring in additional neighboring pages (clustering)
- Alternatively
  - Figure out *working set* from previous activity
  - Page in entire working set of a swapped out process
Working Set

- A working set of a process is used to model the dynamic locality of its memory usage
  - Working set = set of pages a process currently needs to execute without too many page faults
  - Denning in late 60’s

- Definition:
  - \( WS(t,w) = \) set of pages referenced in the interval between time \( t-w \) and time \( t \)
    - \( t \) is time and \( w \) is working set window (measured in page refs)
    - Page is in working set only if it was referenced in last \( w \) references
Working Set

- \( w \equiv \text{working-set window} \equiv \text{a fixed number of page references} \)
  Example: \( 10,000 \text{ – } 2,000,000 \text{ instructions} \)
- \( WS_i (\text{working set of Process } P_i) = \text{set of pages referenced in the most recent } w \) (varies in time)
  - if \( w \) too small will not encompass entire locality.
  - if \( w \) too large will encompass several localities.
  - as \( w \to \infty \), encompasses entire program.
Working Set Example

• Assume 3 page frames
• Let interval be \( w = 5 \)
• \( 1 2 3 2 3 1 2 4 3 4 7 4 3 3 4 1 1 2 2 2 1 \)
  \( w=\{1,2,3\} \quad w=\{3,4,7\} \quad w=\{1,2\} \)
  – if \( w \) too small, will not encompass locality
  – if \( w \) too large, will encompass several localities
  – if \( w \to \infty \), will encompass entire program
• if Total WS \( > \) physical memory \( \Rightarrow \) thrashing
  – Need to free up some physical memory
  – E.g., suspend a process, swap all of its pages out
Working Set Page Replacement

- In practice, convert references to time:
  - E.g. 100ns/ref, 100,000 references \( \approx \) 10msec

- WS algorithm in practice:
  - On each clock tick, clear all R bits and record process virtual time \( t \)
  - When looking for eviction candidates, scan all pages of process in physical memory
    - If \( R == 1 \)
      - Store \( t \) in LTU (last time used) of PTE and clear \( R \)
    - If \( R == 0 \)
      - If \((t - LTU) > WS\_Interval\) (i.e., \( w \)), evict the page (because it is not in working set)
    - Else select page with the largest difference

See Tanenbaum, §3.4.8
WSClock
(combines Clock and WS algorithm)

• WSClock
  – Circular list of entries containing
    • R, M, *time of last use*
    • R and *time* are updated on each clock tick
  – Clock “hand” progresses around list
    • If R = 1, reset and update *time*
    • If R = 0, and if *age > WS_interval*, and if clean, then claim it.
    • If R = 0, and if *age > WS_interval*, and if dirty, then schedule a disk write
    • Step “hand” to next entry on list

• Very common in practice

See Tanenbaum, §3.4.9
### Review of Page Replacement Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>Not implementable, but useful as a benchmark</td>
</tr>
<tr>
<td>NRU (Not Recently Used)</td>
<td>Very crude</td>
</tr>
<tr>
<td>FIFO (First-In, First-Out)</td>
<td>Might throw out important pages</td>
</tr>
<tr>
<td>Second chance</td>
<td>Big improvement over FIFO</td>
</tr>
<tr>
<td>Clock</td>
<td>Realistic</td>
</tr>
<tr>
<td>LRU (Least Recently Used)</td>
<td>Excellent, but difficult to implement exactly</td>
</tr>
<tr>
<td>NFU (Not Frequently Used)</td>
<td>Fairly crude approximation to LRU</td>
</tr>
<tr>
<td>Aging</td>
<td>Efficient algorithm that approximates LRU well</td>
</tr>
<tr>
<td>Working set</td>
<td>Somewhat expensive to implement</td>
</tr>
<tr>
<td>WSClock</td>
<td>Good efficient algorithm</td>
</tr>
</tbody>
</table>

Tanenbaum, Fig 3-22
Virtual Memory Subsystem

- All about managing the page cache in RAM of virtual memory …
- … which lives primarily on disk
- See also Chapter 15 of *Linux Kernel Development*, by Robert Love
More on Segmentation

• **Paging** is (mostly) invisible to programmer, but **segmentation** is not
  • Even paging with two-level page tables is invisible

• **Segment**: an open-ended piece of VM
  • Multics (H6000): $2^{18}$ segments of 64K words each
  • Pentium: 16K segments of $2^{30}$ bytes each
    – 8K *global* segments, plus 8K *local* segments per process
    – Each segment may be paged or not
    – Each segment assigned to one of four protection levels

• Program consciously loads **segment descriptors** when accessing a new segment
  • Only OS/2 used full power of Pentium segments
  • Linux concatenates 3 segments to simulate contiguous VM
OS Design Issue — Where does Kernel execute?

- In physical memory
  - Old systems (e.g., IBM 360/67)
  - Extra effort needed to look inside of VM of any process

- In virtual memory
  - Most modern systems
  - Shared segment among all processes

- Advantages of kernel in virtual memory
  - Easy to access, transfer to/from VM of any process
  - No context switch needed for traps, page faults
  - No context switch needed for purely kernel interrupts
Kernel Memory Requirements

- **Interrupt handlers**
  - Must be *pinned* into physical memory
  - At locations known to the hardware
- **Critical kernel code**
  - *Pinned*, never swapped out
- **I/O buffers** (user and kernel)
  - Must be *pinned* and in *contiguous* physical memory
- **Kernel data** (e.g., PCB’s, file objects, semaphores, etc.)
  - Pinned in physical memory
  - Dynamically allocated & freed
  - Not multiples of page size; fragmentation is an issue

**Definition:** Pinned – not subject to being swapped out!

Reason:– I/O and other devices don’t recognize paging!
**Kernel Memory Allocation**

- E.g., Linux PCB \(\text{struct task_struct}\)
  - \(> 1.7\) Kbytes each, pinned
  - Created on every `fork` and every thread create
    - `clone()`
  - Deleted on every `exit`

- Kernel memory allocators
  - Buddy system
  - Slab allocation
Buddy System

- Maintain a segment of contiguous pinned VM
- Round up each request to nearest power of 2
- Recursively divide a chunk of size $2^k$ into two “buddies” of size $2^{k-1}$ to reach desired size
- When freeing an object, recursively coalesce its block with adjacent free buddies

- Problem, still a lot of internal fragmentation
  - E.g., 11 Kbyte page table requires 16 Kbytes
Buddy System (illustrated)

physically contiguous pages

256 KB

128 KB

128 KB

64 KB

64 KB

32 KB

32 KB
Slab Allocation

- Maintain a separate “cache” for each major data type
  - E.g., task_struct, inode in Linux
- **Slab**: fixed number of contiguous physical pages assigned to one particular “cache”
- Upon kernel memory allocation request
  - Recycle an existing object if possible
  - Allocate a new one within a slab if possible
  - Else, create an additional slab for that cache
- When finished with an object
  - Return it to “cache” for recycling
- **Benefits**
  - Minimize fragmentation of kernel memory
  - Most kernel memory requests can be satisfied quickly
Slab Allocation (illustrated)

- **kernel objects**
- **caches**
- **slabs**

3 KB objects

7 KB objects

Physical contiguous pages
Note

- We use this allocation system in the Linux kernel in Project #4
Classical Unix

• Physical Memory
  – Core map (pinned) – page frame info
  – Kernel (pinned) – rest of kernel
  – Frames – remainder of memory

• Page replacement
  – Page daemon
    • runs periodically to free up page frames
    • Global replacement – multiple parameters
    • Current BSD system uses 2-handed clock
  – Swapper – helps paging daemon
    • Look for processes idle 20 sec. or more and swap out longest idle
    • Next, swap out one of 4 largest – one in memory the longest
    • Check for processes to swap in
Linux VM

- Kernel is pinned
- Rest of frames used
  - Processes
  - Buffer cache
  - Page Cache
- Multilevel paging
  - 3 levels
  - Contiguous slab memory allocation using Buddy Algorithm
- Replacement – goal keep a certain number of pages free
  - Daemon (kswapd) runs once per second
    - Clock algorithm on page and buffer caches
    - Clock on unused shared pages
    - Modified clock (by VA order) on user processes (by # of frames)

From Robert Love, for Linux aficionados:
- Chapter 11: Kernel memory mgmt.
- Chapters 12-13: (about file systems)
- Chapter 14: Process address space
- Chapter 15: Page Cache and writeback
Windows

- Uses demand paging with *clustering*. Clustering brings in pages surrounding the faulting page.
- Processes are assigned *working set minimum* (20-50) and *working set maximum* (45-345)
- Working set minimum is the minimum number of pages the process is guaranteed to have in memory.
- A process may be assigned as many pages up to its working set maximum.
- When the amount of free memory in the system falls below a threshold, *automatic working set trimming* is performed to restore the amount of free memory. (Balance set manager)
- Working set trimming removes pages from processes that have pages in excess of their working set minimum
VM Summary

- Memory Management – from simple multiprogramming support to efficient use of multiple system resources
- Models and measurement exist to determine the goodness of an implementation
- In real systems, must tradeoff
  - Implementation complexity
  - Management overhead
  - Access time overhead
Virtual Memory Summary (continued)

- When to swap in a page
  - On demand? or in anticipation?
- What to throw out
  - Page Replacement Policy
- Keeping dirty pages in sync with disk
  - Flushing strategy
- Keeping pages in sync across processors or machines
  - Defer to another time
- Size of physical memory to be effective
  - See previous discussion
- Size of pages for efficiency
  - One size fits all, or multiple sizes?
Reading Assignment

- Tanenbaum
  - §§ 3.1–3.3 (previous topics)
    - Memory Management
    - Paging

- §§ 3.4–3.6 (this topic)
  - Page Replacement Algorithms
  - Design Issues for Paging Systems
  - Implementation Issues for Paging Systems

- § 3.7
  - More on Segmentation
Questions?