Segmenting and Paging of Memory

Segmentation and paging are ways of
• partitioning memory for convenient allocation
• reorganizing memory for convenient usage

How?
• Relocation via address translation
• Protection via matching operations with objects

Result: A logically organized memory

Implications

Offers possibility that not all pieces need be in memory
• Need only piece being referenced
• Other pieces can be on disk
• Bring pieces in only when needed

Result: illusion that there is more memory than there is

What’s needed to support this idea?
• A way to identify whether a piece is in memory
• A way to bring in pieces (from where, to where?)
• Relocation (which we have)
From Logical to Virtual Memory

Logical memory becomes virtual memory
- Still logical (separate organization from physical)
- Memory exists, for all practical purposes

Virtual Memory: illusion of a large physical memory
- Keep only portion of logical memory in physical
- Rest is kept on disk (larger, slower, cheaper)
- Unit of memory is segment or page (or both)

Logical address space becomes virtual address space

Sample Contents of Page Table Entry

<table>
<thead>
<tr>
<th>Valid</th>
<th>Ref</th>
<th>Mod</th>
<th>Frame number</th>
<th>Prot: rwx</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Valid: is this entry valid (is page in physical memory)?
Ref: has this page been referenced yet?
Mod: has this page been modified (dirty)?
Frame: what frame is this page in (assuming in memory)?
Prot: what are the allowable operations?

Address Translation and Page Faults

Get entry by using page number to index into page table

If valid bit is off, page fault
- Trap into operating system
- Find page on disk (kept in kernel data structure)
- Read it into a free frame
  - may need to make room: page replacement
- Record frame number in page table entry, set valid
- Retry instruction (return from page-fault trap)
Faults under Segmentation/Paging

Virtual address: \(<\text{segment } s, \text{ page } p, \text{ offset } i>\>

Use \(s\) to index into segment table (to get page table)
- May get a segment fault

Check bound (Is \(p < \text{ bound?}\))
- May get a segmentation violation

Use \(p\) to index into page table (to get frame \(f\))
- May get a page fault

Concatenate \(f\) and offset \(i\) to get physical address

Problem: Page Faults are EXPENSIVE

Disk: 5-6 orders magnitude slower than memory
- Very expensive; but if very rare, tolerable

Example
- Memory access time: 100 nsec
- Disk access time: 10 msec
- \(p\) = page fault probability
- Effective access time: \(100 + p \times 10,000,000\) nsec
- If \(p = 0.1\%\), effective access time = \(10,100\) nsec

Principle of Locality

Not all pieces are referenced uniformly over time
- Make sure most referenced pieces in memory
- If not, thrashing: constant fetching of pieces

References cluster in time/space
- Will be to same or neighboring areas
- Allows prediction based on past

Policies for Virtual Memory

Fetch policy: when to bring in, how many, which ones
- Demand paging vs. prepaging

Placement policy: where to place in memory
- Relevant for segmentation, irrelevant for paging

Replacement policy: which to remove to make room
- Resident set management, replacement

Cleaning policy: when to write out
- Demand cleaning vs. precleaning
Page Replacement

Goal: kick out page outside of locality of reference

Page replacement is about
• Which page(s) to kick out
• When to kick them out

How to do it in the cheapest way possible
• Least amount of additional hardware
• Least amount of software overhead

Basic Page Replacement Algorithms

FIFO: select page that is oldest
• Simple: use frame ordering
• Doesn't perform very well (oldest may be popular)

OPT: select page that to be used furthest in future
• Optimal, but requires future knowledge
• Establishes best case, good for comparisons

LRU: select page that was least recently used
• Predict future based on past; works given locality
• Costly: time-stamp pages each access, find least

Example Comparing FIFO, OPT, LRU

Ref string: 2 3 2 1 5 2 4 5 3 2 5 2

FIFO 2* 2 2 2 5* 5 5 5 3* 3 3 3
   3* 3 3 3 2* 2 2 2 2 2 5* 5
   6 faults 1* 1 1 4* 4 4 4 4 2*

OPT 2* 2 2 2 2 4* 4 4 2* 2 2
   3* 3 3 3 3 3 3 3 3 3 3
   3 faults 1* 5* 5 5 5 5 5 5 5

LRU 2* 2 2 2 2 2 2 2 2 3* 3 3 3
   3* 3 3 5* 5 5 5 5 5 5 5
   4 faults 1* 1 1 4* 4 4 2* 2 2

Approximating LRU: Clock Algorithm

Idea: select page that is old and also not recently used
• Clock (second chance) is approximation of LRU

Hardware support: reference bit
• Associated with each frame is a reference bit
• Actually, reference bit is in page table entry

How reference bit is used
• When frame filled with page, set bit to 0 (by OS)
• If frame is accessed, set bit to 1 (by hardware)
How Clock Works

Arrange all frames in circle (clock)
Clock hand: next frame to consider
Page fault: do until frame found
  • If ref bit 0, select frame
  • Else, set ref bit to 0
  • Advance clock hand
  • If frame found, break out of loop (else repeat)
If frame found had modified page, must first write out

Example of Clock

Reference page 5: page fault (unavoidable)
  • Hand points to an unreferenced page: use it
  • Advance hand

Example of Clock, continued

Reference page 9: page fault (unavoidable)
  • Hand points to an unreferenced page: use it
  • Advance hand

Reference page 7: page fault (unavoidable)
  • Hand points to an unreferenced page: use it
  • Advance hand
Example of Clock, continued

Ref string: 5 9 7 1 9 5 9
Reference page 1: page fault (1)
• Hand points to a referenced page: skip it
• Set ref bit to 0, advance hand, try again

Example of Clock, continued

Ref string: 5 9 7 1 9 5 9
Trying to find unreferenced page
• Hand points to a referenced page: skip it
• Set ref bit to 0, advance hand, try again

Example of Clock, continued

Ref string: 5 9 7 1 9 5 9
Trying to find unreferenced page
• Hand points to a referenced page: skip it
• Set ref bit to 0, advance hand, try again

Example of Clock, continued

Ref string: 5 9 7 1 9 5 9
Trying to find unreferenced page
• Hand points to an unreferenced page: use it
• Advance hand
Example of Clock, continued

Ref string: 5 9 7 1 9 5 9
Reference page 9
• Page 9 is already in memory: no page fault
• OS does nothing, but hardware sets ref bit to 1

Example of Clock, continued

Ref string: 5 9 7 1 9 5 9
Reference page 5: page fault (2)
• Hand points to a referenced page: skip it
• Set ref bit to 0, advance hand, try again

Example of Clock, continued

Ref string: 5 9 7 1 9 5 9
Trying to find unreferenced page
• Hand points to an unreferenced page: use it
• Advance hand

Example of Clock, continued

Ref string: 5 9 7 1 9 5 9
Reference page 9
• Page 9 already in memory: no page fault
• OS does nothing, but hardware sets ref bit to 1
Comparing to OPT, LRU, FIFO

Ref string: 5 9 7 1 9 5 9
FIFO 5* 5 5 1* 1 1 1
3 faults 9* 9 9 9 5* 5
OPT 5* 5 5 5 5 5 5
9* 9 9 9 9 9
1 faults 7* 1* 1 1 1
LRU 5* 5 5 1* 1 1 1
2 faults 9* 9 9 9 9 9
9* 7* 7 7 5* 5

Resident Set Management

Resident set: pages of process resident in memory
- How big should resident set be? Which pages?
- Who provides frame (same process or another)?

Local: limit frame selection to requesting process
- Isolates effects of page behavior on processes
- Inefficient: some processes have unused frames

Global: select any page frame (from any process)
- Efficient: resident sets grow/shrink accordingly
- No isolation: process can negatively affect another

Two-Handed Clock

Front hand
- Sets ref bits to 0
Back hand
- Selects unreferenced pages
Parameters
- Scan rate
- Hand gap

Multiprogramming Level

Smaller resident sets implies more processes in memory
- Increases multiprogramming level
- Good for processor utilization
- However, beyond some point, thrashing occurs

Resident set should contain the working set
A Process’s Phases of Locality

Process memory reference pattern is a cycle
• Period of locality: few faults outside working set
• Locality change: many faults, change in working set

Denning’s Working Set Model

Working set: \( W(t, \Delta) \)
• Pages referenced during last delta (process time)
Add/remove pages according to \( W(t, \Delta) \)
If working set cannot be in memory, swap process out
Problem: difficult to implement
• Must timestamp pages in working set
• Must determine if timestamp older than \( t - \Delta \)
• Must determine \( \Delta \)

Monitor Page Fault Frequency

If frequency too high, working set not present
• Give process more frames; if none, swap out
If frequency too low, resident set has too many pages
• Take away page frames
Problem: no concept of transitions in locality

Page Size: How Large Should It Be?

Waste due to internal fragmentation: \( p/2 \) per process
• Small \( p \) reduces waste: use small page size
Page table size: proportional to number of pages \( M/p \)
• Large \( p \) reduces page table size
TLB hit rate: proportional to \( p \) * number of entries
• Large \( p \) increases hit rate
Given reduction in memory costs, favor large page sizes
• VAX (’70s technology): 512B; today: 8KB-32KB
Managing Disk Space

Disk is used for swapping/paging

Swapping: allocate region for entire process
  • Variable allocation: e.g., use first fit
  • Within region, can transfer individual pages
  • Determine address by region start + page number

Paging (assuming no per-process regions)
  • On page out, find a free block to hold page
  • Fixed allocation: any free block will do
  • Must keep track of each page address on disk

Paging Daemons

On page fault, there may be no free pages
  • Must make room by paging something out
  • In meantime, process must wait for pageout/pagein

Better strategy: always maintain a pool of free pages
  • Faulting process must only wait for page in

Paging daemon periodically replenishes free page pool
  • When pool goes below threshold, replenish
  • If freed page is accessed, remove from pool

Summary

VM is efficient because of principle of locality

OPT ≥ LRU typically ≥ Clock typically ≥ FIFO

Goal: keep working set in memory

If working set cannot be resident, swap process out

Other factors
  • Use large page size
  • Maintain free pool of page frames