CSCI-GA.2250-001

Operating Systems

Lecture 6: Memory Management II

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com
In case of page fault: which page to remove from Memory?
Replacement Policies

• Used in many contexts when storage is not enough
  – caches
  – web servers
  – pages

• Things to take into account when designing a replacement policy
  – measure of success
  – cost
Optimal Page Replacement Algorithm

- Each page labeled with the number of instructions that will be executed before this page is referenced
- Page with the highest label should be removed
- Impossible to implement
The Not Recently Used Replacement Algorithm

- Two status bits with each page
  - R: Set whenever the page is referenced
  - M: Set when the page is written
- R and M bits are available in most computers implementing virtual memory
- Those bits are updated with each memory reference
  - Must be updated by hardware
  - Reset only by the OS
- Periodically (e.g. on each clock interrupt) the R bit is cleared
  - To distinguish pages that have been referenced recently
The Not Recently Used Replacement Algorithm

<table>
<thead>
<tr>
<th>Class</th>
<th>R</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class 0:</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Class 1:</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Class 2:</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Class 3:</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

NRU algorithm removes a page at random from the lowest numbered unempty class.
The FIFO Replacement Algorithm

- OS maintains a list of the pages currently in memory
- The most recent arrival at the tail
- On a page fault, the page at the head is removed
The Second-Chance Page Replacement Algorithm

• Modification to FIFO
• Inspect the R bit of the oldest page
  – If R=0 page is old and unused -> replace
  – If R=1 then
    • bit is cleared
    • page is put at the end of the list
    • the search continues
• If all pages have R=1, the algorithm degenerates to FIFO
The Second-Chance Page Replacement Algorithm

Moving pages around on the lists is inefficient
The Clock Page Replacement Policy

- Keep page frames on a circular list in the form of a clock
- The hand points to the oldest page
- When page fault occurs
  - The page pointed to by the hand is inspected
  - If R=0
    - page evicted
    - new page inserted into its place
    - hand is advanced
  - If R = 1
    - R is set to 0
    - hand is advanced
The Clock Page Replacement Policy

When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:

- R = 0: Evict the page
- R = 1: Clear R and advance hand
The Least Recently Used (LRU) Page Replacement Algorithm

• Good approximation to optimal
• When page fault occurs, through out the page that has been unused for the longest time
• Realizable but not cheap
LRU
Hardware Implementation 1

• 64-bit counter increment after each instruction
• Each page table entry has a field large enough to include the value of the counter
• After each memory reference, the value of the counter is stored in the corresponding page entry.
• At page fault, the page with lowest value is discarded
LRU Hardware Implementation 1

- 64-bit counter increment after each instruction
- Each page table entry has a field large enough to include the value of the counter
- After each memory reference, the value of the counter is stored in the corresponding page entry
- At page fault, the page with lowest value is discarded

Too expensive!
Too Slow!
LRU: Hardware Implementation 2

- Machine with n page frames
- Hardware maintains a matrix of nxn bits
- Matrix initialized to all 0s
- Whenever page frame k is referenced
  - Set all bits of row k to 1
  - Set all bits of column k to 0
- The row with lowest value is the LRU
LRU: Hardware Implementation 2

Pages referenced: 0 1 2 3 2 1 0 3 2 3
LRU Implementation

- Slow
- Few machines have required hardware
Simulating LRU in Software

• Not Frequently Used (NFU) algorithm
• Software counter associated with each page, initially zero
• At each clock interrupt, the OS scans all pages and add the R bit to the counter
• At page fault: the page with lowest counter is replaced
Enhancing NRU

• NRU never forgets anything -> high inertia
• Modifications:
  – shift counter right 1 bit before adding R
  – R is added to the leftmost
• This modified algorithm is called aging
• The counter whose counter is lowest is replaced at page fault
## Aging Algorithm

Aging Algorithm

R bits for pages 0-5, clock tick 0: 101011
R bits for pages 0-5, clock tick 1: 110010
R bits for pages 0-5, clock tick 2: 110101
R bits for pages 0-5, clock tick 3: 100010
R bits for pages 0-5, clock tick 4: 011000

<table>
<thead>
<tr>
<th>Page</th>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(d)</th>
<th>(e)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1000000</td>
<td>1100000</td>
<td>1110000</td>
<td>1111000</td>
<td>01111000</td>
</tr>
<tr>
<td>1</td>
<td>0000000</td>
<td>1000000</td>
<td>1100000</td>
<td>0110000</td>
<td>10110000</td>
</tr>
<tr>
<td>2</td>
<td>1000000</td>
<td>0100000</td>
<td>0010000</td>
<td>0010000</td>
<td>10010000</td>
</tr>
<tr>
<td>3</td>
<td>0000000</td>
<td>0000000</td>
<td>1000000</td>
<td>0100000</td>
<td>00100000</td>
</tr>
<tr>
<td>4</td>
<td>1000000</td>
<td>1100000</td>
<td>0110000</td>
<td>1011000</td>
<td>01011000</td>
</tr>
<tr>
<td>5</td>
<td>1000000</td>
<td>0100000</td>
<td>1010000</td>
<td>0101000</td>
<td>00101000</td>
</tr>
</tbody>
</table>
The Working Set Model

- **Working set**: the set of pages that a process is currently using
- **Thrashing**: a program causing page faults every few instructions

An important question:

In multiprogramming systems, processes are sometimes swapped to disk (i.e. all their pages are removed from memory). When they are brought back, which pages to bring?
The Working Set Model

- Try to keep track of each process' working set and make sure it is in memory before letting the process run.

\[ w(k, t) \]: the set of pages accessed in the last \( k \) references at instant \( t \)
The Working Set Model

• OS must keep track of which pages are in the working set
• Replacement algorithm: evict pages not in the working set
• Possible implementation (but expensive):
  – working set = set of pages accessed in the last $k$ memory references
• Approximations
  – working set = pages used in the last 100 msec
Working Set
Page Replacement Algorithm

Scan all pages examining R bit:
- if (R == 1)
  set time of last use to current virtual time
- if (R == 0 and age > \( \tau \))
  remove this page
- if (R == 0 and age \( \leq \tau \))
  remember the smallest time
The WSClock Page Replacement Algorithm

- Based on the clock algorithm and uses working set
- data structure: circular list of page frames
- Each entry contains: time of last use, R bit, and M bit
- At page fault: page pointed by hand is examined
  - If R = 1, the hand advances to next page
  - If R = 0
    - If age > threshold and page is clean → it is reclaimed
    - If page is dirty → write to disk is schedule and hand advances
The WSClock Page Replacement Algorithm
Design Issues for Paging: Local vs Global Allocation

- How memory should be allocated among the competing runnable processes?
- **Local algorithms**: allocating every process a fixed fraction of the memory
- **Global algorithms**: dynamically allocate page frames
- Global algorithms work better
  - If local algorithm used and working set grows -> thrashing will result
  - If working set shrinks -> local algorithms waste memory
Global Allocation

• Method 1: Periodically determine the number of running processes and allocate each process an equal share
• Method 2 (better): Pages allocated in proportion to each process total size
• Page Fault Frequency (PFF) algorithm: tells when to increase/decrease page allocation but says nothing about which page to replace.
Global Allocation: PFF

Unacceptable high process given more pages

so low, process has too much memory
page frames may be taken away
Design Issues: Load Control

- What if PFF indicates that some processes need more memory but none need less?
- **Swap** some processes to disk and free up all the pages they are holding.
- Which process(es) to swap?
  - Strive to make CPU busy (I/O bound vs CPU bound processes)
  - Process size
Design Issues: Page Size

- Large page size -> internal fragmentation
- Small page size ->
  - larger page table
  - More overhead transferring from disk
Design Issues: Page Size

• Assume:
  – \( s \) = process size
  – \( p \) = page size
  – \( e \) = size of each page table entry

• So:
  – number of pages needed = \( s/p \)
  – occupying: \( se/p \) bytes of page table space
  – wasted memory due to fragmentation: \( p/2 \)
  – overhead = \( se/p + p/2 \)

• We want to minimize the overhead:
  – Take derivative of overhead and equate to 0:
  – \(-se/p^2 + \frac{1}{2} = 0\) \( \rightarrow \) \( p = \sqrt{2se} \)
Design Issues:  
Separate Instruction and Data Spaces

- The linker must know about it  
- Paging can be used in each separately
Design Issues: Shared Pages

- To save space, when same program is running by several users for example
- If separate I and D spaces: process table has pointers to Instruction page table and Data page table
- In case of common I and D spaces:
  - Special data structure is needed to keep track of shared pages
  - Copy on write for data pages
Design Issues: Shared Libraries

• Dynamically linked
• Loaded when program is loaded or when functions in them are called for the first time
• Compilers must not produce instructions using absolute addresses in libraries -> position-independent code
Design Issues: Cleaning Policy

- Paging daemon
- Sleeps most of the time
- Awakened periodically to inspect state of the memory
- If too few pages are free -> daemon begins selecting pages to evict
Conclusions

• Virtual memory is very widely used

• Many design issues for paging systems:
  – Page replacement algorithm
    • The two best ones: aging and WSClock
  – Page size
  – Local vs Global Allocation
    • Global algorithms work better
  – Load control
  – Dealing with shared pages

• Sections 3.4 and 3.5 from the textbook