Page Replacement

Michael E. Locasto
CPSC 457
Agenda

Tour of the wiki (5 minutes)

Summary of Memory Management (5 minutes)

Page Replacement Algorithms (25 minutes)

Coding Activity (30 minutes)

Informal Anonymous Survey (5 minutes)

Discussion Arising (5 minutes)
Memory Management Recap

Memory Addressing :: why doesn’t programs’ manipulation of variables conflict / collide?

Process Address Space :: how does the kernel support the ABI memory abstraction?

Virtual Memory :: how does a kernel keep all processes in memory at once?
locality rides to the rescue; program data may actually reside anywhere here, but is still logically in the registers.
Virtual memory is predicated on a mechanism for extending a set of virtual addresses over a collection of multiple physical devices (mostly primary RAM and disk).

In essence, this is the *actual* operation of virtual memory.

$\text{VM} = \text{page table translation} + \text{frame management}$
Page Replacement Focus Question

How do you select which page frame to evict in favor of an incoming (from disk) memory page?
Recall: Picking a Destination Frame

Case A: Free Frame
There is a free page frame.

The OS selects it, fetches the data from disk, and writes the page into the frame, updating the indexing as needed.

Case B: No free frame.
The OS will:
1. select some frame (via a page replacement algorithm)
2. evicts its contents (if it is dirty) to disk (and mark its indexing information in the page tables as 'absent')
3. write the new page contents to frame.
Recall: Picking a Destination Frame

**Case A: Free Frame**
There is a free page frame.

The OS selects it, fetches the data from disk, and writes the page into the frame, updating the indexing as needed.

**Case B: No free frame.**
The OS will:

1. select some frame (via a page replacement algorithm)
2. evicts its contents (if it is dirty) to disk (and mark its indexing information in the page tables as 'absent')
3. write the new page contents to frame.
Bits: Contents of Page Table Entries

20 bits: page frame index
Present bit: is this logical page in the frame?
Protection bits: R/W/X (R sometimes implies X)
Dirty bit: was this page written?
Accessed bit: hardware sets this; OS must unset.
  used to determine candidates for swapping
User/Supervisor: ring 3 or everything else
Simple Approach: Random

**Approach:** Select a random frame to evict

**Pro:** Simple to implement & understand

**Con:** May select a heavily-used frame, will soon cause a page fault

**Question:** can we measure this? How bad is it in practice?
Best Approach: Optimal

**Approach:** Select frame that will be referenced furthest in the future

**Pro:** Optimal; maximally delays each page fault

**Con:** Fantasy

**Question:** Might this be on a test?
Simple Approach: FIFO

**Approach:** Evict frame with “oldest” contents

**Pro:** Simple to implement & understand, low overhead

**Con:** May select an important frame

**Question:** can we improve FIFO’s knowledge of page frame content semantics?
Second Chance: Improve FIFO

**Approach:** Evict in FIFO order, but check if page is in use before evicting (2\textsuperscript{nd} chance)

**Pro:** Vastly improves FIFO

**Con:** Degenerates to FIFO; shuffles pages on list

**Note:** Mostly equivalent to “clock” (clock doesn’t shuffle pages around a list); relies on hardware providing a “Referenced” bit
Good Approach: Least Recently Used

**Approach:** Evict frame that has not been referenced for a long time

**Pro:** Semantics closely match optimal

**Con:** Costly implementation; hardware requirements

**Question:** Can we avoid the need for updating the linked list? Or specialized hardware?
Simulate LRU: NFU+aging

**Approach:** Keep a small counter per-frame, shift right to age, add R to MSb

**Pro:** Efficient simulation of LRU

**Con:** No strict ordering / tiebreak; small horizon
EXAMPLES
FIFO Page Replacement

4 Physical Page Frames

Memory Reference String: 0 2 1 4 3 9 1 1 2 0

8 page faults
2 hits

<table>
<thead>
<tr>
<th>Page Ref.</th>
<th>Frame 1</th>
<th>Frame 2</th>
<th>Frame 3</th>
<th>Frame 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>init</td>
<td>0*</td>
<td>2*</td>
<td>1*</td>
<td>4*</td>
</tr>
<tr>
<td>3</td>
<td>3*</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>9*</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1+</td>
<td>3</td>
<td>9</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1+</td>
<td>3</td>
<td>9</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>9</td>
<td>2*</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>9</td>
<td>2</td>
<td>0*</td>
</tr>
</tbody>
</table>
**FIFO Page Replacement**

5 Physical Page Frames

Memory Reference String: 0 2 1 4 3 9 1 1 2 0

<table>
<thead>
<tr>
<th>Page Ref.</th>
<th>Frame 1</th>
<th>Frame 2</th>
<th>Frame 3</th>
<th>Frame 4</th>
<th>Frame 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>init</td>
<td>0*</td>
<td>2*</td>
<td>1*</td>
<td>4*</td>
<td>3*</td>
</tr>
<tr>
<td>9</td>
<td>9*</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>1+</td>
<td>9</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>1+</td>
<td>9</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2+</td>
<td>9</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>9</td>
<td>0*</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

7 page faults
3 hits
Optimal Page Replacement

4 Physical Page Frames

Memory Reference String:

0 2 1 4 3 9 1 1 2 0

6 page faults
4 hits

<table>
<thead>
<tr>
<th>Page Ref.</th>
<th>Frame 1</th>
<th>Frame 2</th>
<th>Frame 3</th>
<th>Frame 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>init</td>
<td>0*</td>
<td>2*</td>
<td>1*</td>
<td>4*</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>3*</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>9*</td>
</tr>
<tr>
<td>1+</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>1+</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>2+</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>0+</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>9</td>
</tr>
</tbody>
</table>
### LRU Page Replacement

- **4 Physical Page Frames**
- **Memory Reference String:** 0214391120
- **8 page faults**
- **2 hits**

Note different eviction choices from FIFO

<table>
<thead>
<tr>
<th>Page Ref.</th>
<th>Frame 1</th>
<th>Frame 2</th>
<th>Frame 3</th>
<th>Frame 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>init</td>
<td>0*</td>
<td>2*</td>
<td>1*</td>
<td>4*</td>
</tr>
<tr>
<td>3</td>
<td>3*</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>9*</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1+</td>
<td>3</td>
<td>9</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1+</td>
<td>3</td>
<td>9</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>9</td>
<td>1</td>
<td>2*</td>
</tr>
<tr>
<td>0</td>
<td>0*</td>
<td>9</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Other Algorithms

NRU: track R/M bits, remove random from lowest non-empty class
  00: not referenced, not modified
  01: not referenced, modified
  10: referenced, not modified
  11: referenced, modified

Pure NFU
2-bit replacement (aging)
Working Set: prepaging, scan whole page table
WSClock
Notes

Linux PFRA: see ULK, Chapter 17
Processes that have been sleeping for a while may have (all) their frames stolen.
Some PRAs make sense only in the context of the current process’s working set.

One critical goal of page frame reclaiming: preserve enough frames for the kernel to recover from low-memory conditions.
Activity One: Page Fault Plotter

Using SystemTap script, record all page faults on the system

Plot with Google Chart API

(this depends on network connectivity)
Activity Two: P.R. Simulator

struct page
struct frame (size plus ID of page in it)
declare memory for 100 frames
generate endless random page reference string
record number of “page faults”
compare FIFO() vs. LRU()
REFERENCE SLIDES
Figure 3-1. Segmentation and Paging

“hidden” segment register state bits

TLB caching