Today

• Algorithm that approximates the OPT replacement algorithm.
Least Recently Used (LRU) Page Replacement

- A recently used page is likely to be used again in the future.
- Replace the page that has not been used for the longest period of time.
- Use the recent past as an approximation of the near future.

Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
Distinguish recently used from not recently used pages

- **Counters**
  - Each page-table entry is associated with a time-of-use field.
  - Add to the CPU a logical clock.
    - The clock is incremented at every memory access.
  - At every memory access, the field of the referenced page is updated with the clock.
  - Scan the page table to find the LRU page.

- **Stack**
  - Whenever a page is referenced, it is removed from the stack and put on the top.
  - The LRU page is always at the bottom.
  - The update is expensive.
LRU: Clock Algorithm

- Each page entry is associated with a reference bit.
  - Set on use, reset periodically by the OS.
- Algorithm:
  - Scan: if ref bit is 1, set to 0, and proceed. If ref bit is 0, stop and evict.
- Problem:
  - Low accuracy for large memory
LRU: Clock Algorithm

- Solution: Add another hand
  - Leading edge clears ref bits
  - Trailing edge evicts pages with ref bit 0

- What if angle small?
- What if angle big?
Global vs Local Allocation

- **Global replacement**
  - Single memory pool for entire system.
  - On page fault, evict oldest page in the system.
  - Problem: lack of performance isolation.

- **Local (per-process) replacement**
  - Have a separate pool of pages for each process.
  - Page fault in one process can only replace pages from its own process.
  - Problem: might have idle resources.
Thrashing

- Excessive rate of page replacement
  - Keep throwing out page that will be referenced soon.
  - Keep referencing pages that are not in memory.
- Why does it occur?
  - Too many processes in the system.
- How can we solve this problem?
  - *Locality* model of process execution.
    - A locality is a set of pages that are actively used together.
Working Set

- Estimate locality $\rightarrow$ Identify useful pages $\rightarrow$ Do not evict these pages, because they are likely to be referenced again.

- Working Set = An approximation of the program's locality.
  - The set of pages in the most recent $\Delta$ page references.
  - $\Delta$: working-set window

- As a process executes, it moves from locality to locality.

- Example ($\Delta = 10$):
  - $t_1 \rightarrow WS = \{1,2,5,6,7\}$
  - $t_2 \rightarrow WS = \{3,4\}$

- If allocated frames do not accommodate current locality, the process will thrash.
Computing the working set

• Working set = sets of pages in the working set window.

• Difficulty: the working set window is a moving window. At each memory reference:
  – a new reference appears at one end -> the corresponding page should be marked as a member of the working set.
  – The oldest reference drops off the other end -> the corresponding page should be unmarked.
How can we compute WS
• without recording the reference history, but
• with specialized bits associated to page table entries?
  • These bits signify whether a page belongs to the WS.
Computing the working set

• Each page table entry is associated with 1 reference bit and Δ WS-bits.

• At every page reference:
  – Set the corresponding reference bit to 1.
  – Update the working set:
    • Shift WS-bits one bit to the right.
    • Put reference bit to the most significant WS-bit.
  – Reset reference bits to 0.

• If some WS-bits of a page are set to 1, then the page belongs to the WS.

• If all WS-bits of a page are 0, then the page does not belong to WS.
  – This page can be evicted.
Computing the working set

... 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 ...  \( \Delta = 10 \)

\[ \Delta \]

\[ t_1 \ t_2 \ t_3 \ t_4 \ t_5 \]

<table>
<thead>
<tr>
<th>Page number</th>
<th>WS-bits after access ( t_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( t_1 ) ( t_2 ) ( t_3 ) ( t_4 ) ( t_5 )</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>WS</td>
<td></td>
</tr>
</tbody>
</table>
Computing the working set

... 2 6 1 5 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 4 4 1 3 2 3 4 4 3 4 4 4 ...

\( \Delta = 10 \)

\begin{tabular}{|c|c|c|c|c|c|}
\hline
Page number & t1 & t2 & t3 & t4 & t5 \\
\hline
1 & 10000000100 & 0100000010 & 0010000001 & 0001000000 & 0000100000 \\
2 & 0000000001 & 0000000000 & 1000000000 & 0100000000 & 0010000000 \\
3 & 0000000000 & 0000000000 & 0000000000 & 1000000000 & 0100000000 \\
4 & 0000000000 & 0000000000 & 0000000000 & 0000000000 & 1000000000 \\
5 & 0100001000 & 0010000100 & 0001000010 & 0000100001 & 0000010000 \\
6 & 0000000010 & 1000000001 & 0100000000 & 0010000000 & 0001000000 \\
7 & 0011110000 & 0011110000 & 0001111000 & 0000111100 & 0000011111 \\
\hline
WS & \{1,2,5,6,7\} & \{1,5,6,7\} & \{1,2,5,6,7\} & \{1,2,3,5,6,7\} & \{1,2,3,4,5,6,7\} \\
\hline
\end{tabular}
Working Set Approximation

- It is expensive to update the WS at every page reference.
- We can approximate the WS by updating it after $\Delta/n$ references.
- Need $n$ WS-bits per page table entry.
- After $\Delta/n$ references there will be an interrupt.
- At every page reference:
  - Set the corresponding reference bit to 1.
- At every interrupt:
  - Update the working set:
    - Shift WS-bits one bit to the right.
    - Put reference bit to the most significant WS-bit of each page.
  - Reset reference bits to 0.
## Working Set Approximation

\[ \Delta = 10, \ n = 5 \]

<table>
<thead>
<tr>
<th>Page number</th>
<th>WS-bits after access ti</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>t1</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>WS</td>
<td></td>
</tr>
</tbody>
</table>
# Working Set Approximation

\[ \Delta = 10, \ n = 5 \]

<table>
<thead>
<tr>
<th>Page number</th>
<th>WS-bits after access ti</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>t1</td>
<td>t3</td>
<td>t5</td>
</tr>
<tr>
<td>1</td>
<td>10010</td>
<td>01001</td>
<td>00100</td>
</tr>
<tr>
<td>2</td>
<td>00001</td>
<td>10000</td>
<td>01000</td>
</tr>
<tr>
<td>3</td>
<td>00000</td>
<td>00000</td>
<td>10000</td>
</tr>
<tr>
<td>4</td>
<td>00000</td>
<td>00000</td>
<td>10000</td>
</tr>
<tr>
<td>5</td>
<td>10010</td>
<td>01001</td>
<td>00100</td>
</tr>
<tr>
<td>6</td>
<td>00001</td>
<td>10001</td>
<td>01000</td>
</tr>
<tr>
<td>7</td>
<td>01100</td>
<td>00110</td>
<td>00011</td>
</tr>
</tbody>
</table>

**WS** \{1,2,5,6,7\} \{1,2,5,6,7\} \{1,2,3,4,5,6,7\}
Page Fault Frequency

- PFF = page faults / instructions executed.
- If PFF rises above threshold, process needs more memory.
  - Not enough memory on the system? → Swap out.
- If PFF sinks below threshold, memory can be taken away.
Working Sets and Page Fault Rates
Today

• Algorithm that approximates the OPT replacement algorithm.
Coming up…

• Next lecture: Review
• HW3: due today
• Exam2: Wednesday, last 30mins of class