VIRTUAL MEMORY

Ken Wong
Washington University, St. Louis

kenw@arl.wustl.edu
www.arl.wustl.edu/~kenw

VIRTUAL MEMORY OVERVIEW

- **Real Memory**: Main memory
- **Virtual Memory**: The memory perceived by the user or programmer
  * Implemented through paging and segmentation with *page swapping*
- **Properties of Paging and Segmentation**
  * **Dynamic Address Translation**: Memory references are logical addresses that are dynamically translated into physical addresses at run time.
  * **Non-contiguous Main Memory**: A process may be broken up into pieces that need not be contiguously located in main memory.
- **Potential Benefits**
  * **Effective Multiprogramming**: More processes in main memory
  * **Less Memory Constrained**: Loosens memory constraints
  * **Protection**: Memory references only to own physical memory

DEMAND PAGING

- **Bring a page into main memory only when it is needed**
  * Less I/O needed
  * Less memory needed
  * Faster response
  * More users
- **Page is needed only when it is referenced**
  * Abort invalid references
  * Swap in pages when referenced but not in main memory

MATRIX MULTIPLY

```c
long A[N][N], B[N][N], C[N][N];
for (int i=0; i<N; i++)
  for (int j=0; j<N; j++)
    for (int k=0; k<N; k++)
      C[i][j] = A[i][k] * B[k][j];
```

- **High Temporal Locality**: Instructions, A (not B)

Hit Rate | Memory
---|---
A | Text
B
C
**TYPICAL VIRTUAL MEMORY PARAMETERS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page Size</td>
<td>512 to 8,192 bytes</td>
</tr>
<tr>
<td>Hit Time</td>
<td>5 to 100 nanoseconds</td>
</tr>
<tr>
<td>Miss Penalty</td>
<td>5 to 30 milliseconds</td>
</tr>
<tr>
<td>Main Memory Size</td>
<td>64 to 2,048 MB</td>
</tr>
<tr>
<td>Desired Hit Rate</td>
<td>≥ 99.9 %</td>
</tr>
</tbody>
</table>

- Reference frame not in main memory ⇒ **Page Fault**

**PAGED VIRTUAL MEMORY ISSUES**

- *How is a page frame found if it is in main memory?*
  - Page Table: Physical address = <Frame #, Offset>
- *How large should a page frame be?*
  - Big page ⇒ 1) Small page table; 2) more efficient read/write; 3) greater internal fragmentation; 4) higher process load time
- *Which page frame should be replaced on a virtual memory miss?*
  - The one that is the least likely to be referenced in the future
- *Where can a page be placed in main memory?*
  - Almost anywhere in main memory ⇒ Need associative hardware for address translation
- *When should a page frame be written back to the swap device (disk)?*
  - Only if it has been modified (it is dirty) and as late as possible

---

**BASIC PAGE TABLE STRUCTURE**

- Translate a **virtual (logical) address** (page number, offset) into a **physical address** (frame number, offset) using a page table

---

**Huge Virtual Address Spaces**

- *If the page size is 1 KB in a virtual memory that can be as large as $2^{31}$ bytes (2 gigabytes), how many entries will the page table have?*
  - $2^{31} / 2^{10} = 2^{21}$, about 2 million entries
- *How many pages will be occupied by the page table if each row is 32 bits (4 bytes)?*
  - $2^{31} \cdot 2 / 2^{16} = 2^{33} = 8$ K pages! ⇒ Page tables can be huge and are subject to paging
### TYPICAL TLB PARAMETERS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Size</td>
<td>4 to 8 bytes (1 page entry)</td>
</tr>
<tr>
<td>Hit Time</td>
<td>2.5 to 5 nanoseconds (1 clock cycle)</td>
</tr>
<tr>
<td>Miss Penalty</td>
<td>50 to 150 nanoseconds</td>
</tr>
<tr>
<td>TLB Size</td>
<td>32 to 8,192 bytes</td>
</tr>
<tr>
<td>Hit Rate</td>
<td>98% to 99.9%</td>
</tr>
</tbody>
</table>

### PAGE FAULT RATE

- Consider the first array example. Approximately how many memory operations are represented by the inner loop?

* Assume: 1) There is no data cache; 2) All instructions are in an instruction cache; and 3) the accumulation of C[i][j] is done in a register.

  ```c
  R1 <-- 0;
  for (int k = 0; k < 1024; k++) {
    R2 <-- A[i][k];       // Load R2 with A[i][k]
    R3 <-- B[k][j];       // Load R3 with B[k][j]
    R1 = R1 + R2 * R3;
  }
  R1 --> C[i][j];        // Store
  ```

- Inner Loop: Approximately 2,048 memory loads and 1 memory store

- **Page Fault Rate**: The number of page faults per memory reference.
MULTIPROGRAMMING EFFECTS

REPLACEMENT ALGORITHMS

- **Goal**: Select a page to be replaced when a new page must be swapped into memory

- **Basic Algorithms**
  - *Optimal (Impractical)*
    - Select the P for which the time to the next reference is the longest.
  - *First-In, First-Out (FIFO)*
    - Select the oldest P
  - *Least Recently Used (LRU)*
    - Select the P that hasn’t been referenced for the longest time in the past
  - *Clock*
    - Approximates LRU using a clock structure

- **Example Page Reference Stream**: 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

### REPLACEMENT EXAMPLE

**OPT**

<table>
<thead>
<tr>
<th>2</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>5</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>3</th>
<th>2</th>
<th>5</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

**FIFO**

<table>
<thead>
<tr>
<th>2</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

**LRU**

<table>
<thead>
<tr>
<th>2</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
**The Working Set Model**

- $W(t, \Delta)$ is the working set at virtual time $t$ with a window size of $\Delta$ and is:
  - Defined over the page reference string for each process
  - The set of pages that have been referenced in the time interval $[t-\Delta, t]$.
- **Example:**
  
  
  \[
  \begin{array}{cccccccc}
  9 & 0 & 3 & 8 & 9 & 2 & 3 & 9 \\
  \end{array}
  \]

  
  
  \[
  W(4, 4) = \{0, 3, 8, 9\}
  \]

  
  
  \[
  W(2, 2) = \{0, 9\} \quad W(15, 5) = \{0, 2, 9\}
  \]

- $W(t, \Delta)$ varies over time $t$ even with a fixed window size $\Delta$

**The Working Set Strategy**

- **The Strategy**
  - Monitor $W(t, \Delta)$ for each process
  - Periodically remove pages from the resident set of a process that are not in its $W(t, \Delta)$
  - Schedule a process only if its working set is in main memory
- **Problems**
  - The past doesn’t always predict the future
  - An exact measurement of $W(t, \Delta)$ is impractical because it requires a time-ordered queue of pages.
  - The optimal value of $\Delta$ is unknown

**Page-Fault Frequency (PFF) Algorithm**

- **Idea:** Adjust the resident set size according to the page fault rate.
- **Basic Algorithm**
  - Select a threshold $F$, the minimum time between page faults
  - Mark each page that is referenced with a use-bit (U) of 1
  - When a page fault occurs, compute the time $F'$ since the last page fault and adjust the resident set size:
    - $F' < F$: Add a page to the resident set
    - $F' \geq F$: Discard all pages with a use-bit (U) of 0, and shrink the resident set size
  - Reset all use-bits after a page fault
- **A Variation:** Use 2 thresholds to provide hysteresis
- **A Flaw:** Poor performance during expanding transition periods