Lecture 15: Background Information for the VMWare ESX Memory Management paper
40 users, all running the same editor:
- 150KB of code +
- 50KB of data each.

How much physical memory needed with shared pages?
Performance of Demand Paging

- **Page Fault Rate** $0 \leq p \leq 1.0$
  - if $p = 0$ no page faults
  - if $p = 1$, every reference is a fault

- **Effective Access Time (EAT)**
  \[
  \text{EAT} = (1 - p) \times \text{memory access} \\
  + p \times (\text{page fault overhead} \\
  + \text{swap page out} \\
  + \text{swap page in} \\
  + \text{restart overhead})
  \]
Demand Paging Example

- Memory access time = 200 nanoseconds

- Average page-fault service time = 8 milliseconds

- EAT = \((1 - p) \times 200 + p \times 8,000,000\)
  = \(200 + p \times 7,999,800\)

- If one access out of 1,000 causes a page fault, then
  EAT = 8.2 microseconds.
  This is a slowdown by a factor of 40!!
Page Replacement

1. Swap out victim page
2. Change to invalid
3. Swap desired page in
4. Reset page table for new page
Page Replacement Algorithms

- Performance objective: lowest page-fault rate
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string

Common page replacement algorithms:
- FIFO
- Least Recently Used
- Optimal (theoretical only)
- Counting: LFU and MFU
- ... Countless others in the literature
Allocation of Frames

- Each process needs *minimum* number of pages
- Two major allocation schemes
  - fixed allocation
  - priority allocation
- Global vs. local page replacement
  - **Global replacement** – process selects a replacement frame from the set of all frames; one process can take a frame from another
  - **Local replacement** – each process selects from only its own set of allocated frames
Fixed Allocation

- Equal allocation – For example, if there are 100 frames and 5 processes, give each process 20 frames.

- Proportional allocation – Allocate according to the size of process
  - \( s_i = \text{size of process } p_i \)
  - \( S = \sum s_i \)
  - \( m = \text{total number of frames} \)
  - \( a_i = \text{allocation for } p_i = \frac{s_i}{S} \times m \)

\[
\begin{align*}
m & = 64 \\
s_i & = 10 \\
s_2 & = 127 \\
a_1 & = \frac{10}{137} \times 64 \approx 5 \\
a_2 & = \frac{127}{137} \times 64 \approx 59
\end{align*}
\]
Priority Allocation

- Use a proportional allocation scheme using priorities rather than size

- If process $P_i$ generates a page fault,
  - select for replacement one of its frames
  - select for replacement a frame from a process with lower priority number
First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)

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

9 page faults

- 4 frames

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

10 page faults

- Belady’s Anomaly: more frames ⇒ more page faults
### FIFO Page Replacement

**Reference String**: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

**Page Frames**: 7 7 2 2 4 4 0 0 0 2 2 2 1 1 7 7 7 1 0 0 3 3 3 2 2 1

- Each element in the reference string represents the page requested at that time.
- The page frames show the pages that are currently in memory.
FIFO Illustrating Belady’s Anomaly

The graph illustrates the number of page faults as a function of the number of frames. As the number of frames increases, the number of page faults decreases. This is an example of Belady’s Anomaly, where increasing the number of frames can lead to an increase in the number of page faults.
First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)

```
Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
3 frames (3 pages can be in memory at a time per process)
```

- 4 frames

```
Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
4 frames
```

- Belady’s Anomaly: more frames ⇒ more page faults
FIFO Page Replacement

reference string

| 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |

page frames

| 7 | 7 | 7 | 7 | 2 | 2 | 2 | 4 | 4 | 4 | 0 | 0 | 0 | 7 | 7 | 7 |
| 0 | 0 | 0 | 3 | 3 | 3 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 3 | 3 | 3 | 2 | 2 | 2 | 1 | 1 |

| 7 | 7 | 7 | 7 | 0 | 0 | 0 | 7 | 7 | 7 | 7 | 0 | 0 | 7 | 7 | 7 |
FIFO Illustrating Belady’s Anomaly

The graph illustrates the number of page faults over the number of frames. As the number of frames increases, the number of page faults decreases. This is an example of Belady’s Anomaly, where the optimal number of frames is not always the best choice.

- Number of page faults:
  - 16
  - 14
  - 12
  - 10
  - 8
  - 6
  - 4
  - 2
  - 0

- Number of frames:
  - 1
  - 2
  - 3
  - 4
  - 5
  - 6
  - 7

The graph shows a decrease in page faults as the number of frames increases, but there is a point where adding more frames results in fewer page faults.
Optimal Algorithm

- Replace page that will not be used for longest period of time
- 4 frames example
  
  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

  
  1
  2
  3
  4
  5

  6 page faults

- Used for measuring how well a new algorithm performs
Optimal Page Replacement

reference string
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

page frames
7 7 7 2 2 2 2 2 7
0 0 0 0 4 0 0 0
1 1 1 3 3 3 1 1
Least Recently Used (LRU) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Counter implementation

- Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
- When a page needs to be changed, look at the counters to determine which are to change
### LRU Page Replacement

**Reference String**

| 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |

**Page Frames**

<table>
<thead>
<tr>
<th>7</th>
<th>7</th>
<th>7</th>
<th>2</th>
<th>2</th>
<th>4</th>
<th>4</th>
<th>4</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>3</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>7</th>
<th>7</th>
<th>7</th>
<th>7</th>
</tr>
</thead>
</table>

---

14.19 Silberschatz, Galvin and Gagne ©2009
Counting Algorithms

- Keep a counter of the number of references that have been made to each page

- **LFU Algorithm**: replaces page with smallest count

- **MFU Algorithm**: based on the argument that the page with the smallest count was probably just brought in and has yet to be used
Thrashing

- If a process does not have “enough” pages: high page-fault rate:
  - low CPU utilization
  - OS thinks that it needs to increase the degree of multiprogramming
  - another process added to the system
- **Thrashing** ≡ a process is busy swapping pages in and out
Demand Paging and Thrashing

- Why does demand paging work?
  Locality model
  - Process migrates from one locality to another
  - Localities may overlap

- Why does thrashing occur?
  \[ \sum \text{size of locality} > \text{total memory size} \]
Locality In A Memory-Reference Pattern
Working-Set Model

- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references
  - Example: 10,000 instruction
- $WSS_i$ (working set of Process $P_i$) =
  - total number of pages referenced in the most recent $\Delta$ (varies in time)
    - if $\Delta$ too small will not encompass entire locality
    - if $\Delta$ too large will encompass several localities
    - if $\Delta = \infty \Rightarrow$ will encompass entire program
- $D = \Sigma WSS_i \equiv$ total demand frames
- if $D > m \Rightarrow$ Thrashing
- Policy: if $D > m$, then suspend one of the processes
Working-set model
Page-Fault Frequency Scheme

- Establish “acceptable” page-fault rate
  - If actual rate too low, process loses frame
  - If actual rate too high, process gains frame
Working Sets and Page Fault Rates

A graph showing the relationship between page fault rate and time, with a highlighted working set.
Memory-Mapped Files

- Memory-mapped file I/O allows file I/O to be treated as routine memory access by mapping a disk block to a page in memory.

- A file is initially read using demand paging. A page-sized portion of the file is read from the file system into a physical page. Subsequent reads/writes to/from the file are treated as ordinary memory accesses.

- Simplifies file access by treating file I/O through memory rather than `read()` `write()` system calls.

- Also allows several processes to map the same file allowing the pages in memory to be shared.