What is Virtual Memory?

- Virtual memory systems use a view of logical memory that is different and potentially much larger than that of physical memory.
- Virtual memory takes advantage of swapping to portray memory capacity as much larger than it really is.
- Virtual memory systems use disk storage to hold the entire contents of logical memory including whatever does not fit in physical memory.
Why Virtual Memory?

- Most programs do not need to keep everything resident to run efficiently:
  - Most error-handling routines are used rarely.
  - Arrays and other data structures have more storage allocated than they usually need.
  - Certain program features are rarely used.

Benefits of Virtual Memory

- Programs are limited to the size of physical memory.
- More programs can run concurrently without decreasing throughput.
- There is less swapping when switching processes.
Looking Up a Page In Physical Memory

When A Page In NOT In Physical Memory
Demand Paging

- Demand paging means that we load pages into physical memory only as we need them.
- This requires hardware support to distinguish between pages in memory that those that are not (such as a valid-invalid bit).
- As long as we are accessing pages that are memory-resident, everything is handled as usual.

Signaling A Page Fault

1. Fetch A
2. Look up
   - Step 1: Page Table
   - Step 2: Operating System
     - Signal page fault
Handling a Page Fault

1. Step 3 - Find the page on backing store
2. Step 4 - Load into physical memory
3. Step 5 - Update page table
4. Step 6 - Resume instruction

Resuming After A Page Fault

1. Fetch A
2. Step 6 - Resume instruction
3. Step 5 - Update page table
How Many Resident Pages Does An Instruction Need?

• A typical worst case would be a three-operand instruction
  – e.g., MOVL3 A, B, C
  – This requires 4 pages resident in memory
• The IBM 360/370 MVC instruction can require up to 5 pages.

The Performance of Demand Paging

• Paging compromises execution speed for smaller memory requirements.
• The effective time for executing an instruction:
  \[ t_{\text{eff}} = (1-p) \times ma + p \times t_{pf} \]
• The single largest factor in execution time is the time it takes to handle a page fault.
• The probability of a page fault is the most significant factor in a program’s execution time.
Page Replacement

- Eventually we will need to swap a page into physical memory but have no available frames for it.
- In most cases, we will choose which page will be replaced in physical memory.
- While the optimum choice is the page for which the time to the next reference is longest, this is almost impossible to determine.

Page-Replacement Algorithms

- The most common algorithms to consider are:
  - FIFO (First In, First Out)
  - LRU (Least Recently Used)
  - LFU (Least Frequently Used)
  - MFU (Most Frequently Used)
Evaluating A Page-Replacement Algorithm

- Reference string – a string of memory references.
  - e.g., 7FA3, 0106, 1191, 21B0, 0100, 3142, …
- We reduce these to the pages on which they are located (at 4096 bytes per page):
  - 7, 0, 1, 2, 0, 3, …
- Using our reference string, we can determine which algorithm produces the lower page fault rate.

More Frames Mean Fewer Page Faults

![Graph showing the relationship between number of frames and number of page faults](Graph.png)

- Number of Page Faults
- Number of Frames
FIFO Algorithm

Ref. string

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Page frames

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Belady’s Anomaly

Number of Page Faults

Belady’s Anomaly

Number of Frames
### Optimal Page-Replacement

**Ref. string**

<table>
<thead>
<tr>
<th>Ref. string</th>
<th>7</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>0</th>
<th>3</th>
<th>0</th>
<th>4</th>
<th>2</th>
<th>3</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7</td>
<td>7</td>
<td>7</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></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</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>
</tbody>
</table>

**Page frames**

<table>
<thead>
<tr>
<th>Page frames</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>2</th>
<th>0</th>
<th>1</th>
<th>7</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

### LRU Algorithm

**Ref. string**

<table>
<thead>
<tr>
<th>Ref. string</th>
<th>7</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>0</th>
<th>3</th>
<th>0</th>
<th>4</th>
<th>2</th>
<th>3</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</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>
</tbody>
</table>

**Page frames**

<table>
<thead>
<tr>
<th>Page frames</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>2</th>
<th>0</th>
<th>1</th>
<th>7</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></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></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>
</tbody>
</table>
Counting Algorithms

- Counting algorithms look at the number of occurrences of a particular page and use this as the criterion for replacement.
- Such counting algorithms includes:
  - LFU (Least Frequently Used)
  - MFU (Most Frequently Used)

An Example of Counting Algorithms

<table>
<thead>
<tr>
<th>Page</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>396</td>
</tr>
<tr>
<td>1</td>
<td>385</td>
</tr>
<tr>
<td>2</td>
<td>311</td>
</tr>
<tr>
<td>3</td>
<td>225</td>
</tr>
<tr>
<td>4</td>
<td>128</td>
</tr>
<tr>
<td>5</td>
<td>442</td>
</tr>
<tr>
<td>6</td>
<td>343</td>
</tr>
<tr>
<td>7</td>
<td>514</td>
</tr>
</tbody>
</table>
### LFU Algorithm

**Ref. string**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>0</th>
<th>3</th>
<th>0</th>
<th>4</th>
<th>2</th>
<th>3</th>
<th>0</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

**Page frames**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>0</th>
<th>1</th>
<th>7</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

### MFU Algorithm

**Ref. string**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>0</th>
<th>3</th>
<th>0</th>
<th>4</th>
<th>2</th>
<th>3</th>
<th>0</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</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>0</td>
<td>0</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>0</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

**Page frames**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>0</th>
<th>1</th>
<th>7</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
Allocating Frames

- How many frames do we allocate per process?
  - If it is a single-user, single-tasking system, it’s simple – all the frames belong to the user’s process.
- There are two other questions for which we need answers?
  - What is the minimum number of frames that a process needs?
  - Is page replacement global or local?

Minimum Number of Frames

- Every process must have enough pages to complete an instruction.
- We saw earlier that this can mean as many as 7 pages.
Equal Allocation

• If there are $i$ frames available for $n$ processes, we can give $i/n$ frames to each process. This is *equal allocation*.
• This is unfair to processes with larger memory requirements.

Proportional Allocation

• If each process $p_i$ has virtual memory $s_i$ with $S = \Sigma s_i$ and there are $m$ frames, we can give each process $a_i$ frames where
  
  $a_i = \frac{s_i}{S} \times m$

• This algorithm, which gives each process frames based on their needs, is called proportional allocation.
Global vs. Local Allocation

- When multiple processes are running, the operating system will either replace pages from a frame that already belongs to the process (local allocation) or from the set of frames that can be used by all user processes (global allocation).

Local Allocation

- Local replacement requires that the page being replaced be in a frame belonging to the same process.
- The number of frames belonging to the process will not change.
- This allows processes to control their own page fault rate.
Global Allocation

- The process can replace a page from a set that includes all the frames allocated to user processes.
- High-priority processes can increase their allocation at the expense of lower-priority processes.
- Global allocation makes for more efficient use of frames and there better throughput.

Thrashing

CPU utilization

degree of multiprogramming

Thrashing
Working-Set Model

reference string

\[ \Delta \]

\( t_1 \)

\( \text{WS}(t_1) = \{0, 1, 2, 3, 4, 7\} \)

\( t_2 \)

\( \text{WS}(t_2) = \{0, 1, 2\} \)

Page-Fault Frequency

page fault rate

\( \text{Not enough frames} \)

\( \text{Too many frames} \)

number of frames
Prepaging

- There is a large number of page faults when a process is first loaded.
- We can avoid this problem when we restore a suspended process by swapping in all the pages in its working set.
- The cost is swapping pages that we may not use again.

Optimum Page Size

- Pages invariably have $2^n$ bytes, with anywhere from 512 bytes to 16 KB per page.
- Larger pages lead to smaller page tables and fewer page faults.
- Smaller pages lead to less internal fragmentation and faster page fault handling.
Real-Time Processing

• Real-time processing is the very antithesis of virtual memory.
• Processes which need to run in real-time will have their required pages locked into memory.