Page Replacement Algorithms

**Concept**

- Typically $\text{VAS} \gg \text{Physical Memory}$
- With demand paging, physical memory fills quickly
- When a process faults & memory is full, some page must be swapped out
  - Handling a page fault now requires 2 disk accesses not 1!

**Which page should be replaced?**

- **Local replacement** — Replace a page of the faulting process
- **Global replacement** — Possibly replace the page of another process

Optimal Page Replacement

Clairvoyant replacement

- Replace the page that won’t be needed for the longest time in the future

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

**Faults**

<table>
<thead>
<tr>
<th>Time page needed next</th>
<th>Faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a a a a a a a a</td>
</tr>
<tr>
<td>1</td>
<td>b b b b b b b b</td>
</tr>
<tr>
<td>2</td>
<td>c c c c c c c c</td>
</tr>
<tr>
<td>3</td>
<td>d d d d d d d d</td>
</tr>
</tbody>
</table>

**Optimal Page Replacement**

Clairvoyant replacement

- Replace the page that won’t be needed for the longest time in the future

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

**Faults**

<table>
<thead>
<tr>
<th>Time page needed next</th>
<th>Faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a = 7</td>
</tr>
<tr>
<td></td>
<td>a = 15</td>
</tr>
</tbody>
</table>

**Virtual Memory Management**

**Fundamental issues: A Recap**

- Key concept: Demand paging
  - Load pages into memory only when a page fault occurs
- Issues:
  - Placement strategies
    - Place pages anywhere – no placement policy required
  - Replacement strategies
    - What to do when there exist more jobs than can fit in memory
  - Load control strategies
    - Determining how many jobs can be in memory at one time

**Evaluation methodology**

- Record a trace of the pages accessed by a process
  - Example: (Virtual page, offset) address trace...
    - $(3,0), (1,9), (4,1), (2,1), (5,3), (2,0), (1,9), (2,4), (3,1), (4,8)$
  - Generates page trace $3, 1, 4, 2, 5, 2, 1, 2, 3, 4$ (represented as $c, a, d, b, a, b, a, b, c, d$)
- Hardware can tell OS when a new page is loaded into the TLB
  - Set a used bit in the page table entry
  - Increment or shift a register
- Simulate the behavior of a page replacement algorithm on the trace and record the number of page faults generated
  - Fewer faults $\rightarrow$ better performance
### Implementation:

**Least Recently Used Page Replacement**

Use the recent past as a predictor of the near future.

- Simple to implement
  - A single pointer suffices
- Performance with 4 page frames:

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>Page Frames</td>
<td>0</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Faults:**

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>Page Frames</td>
<td>0</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LRU page stack:**

```
     || D | E | F |
Frame 0 || a | b | c |
Frame 1 || a | b | c |
Frame 2 || a | b | c |
Frame 3 || a | b | c |
```

**Page to replace:**

```
     || D | E | F |
Frame 0 || a | b | c |
Frame 1 || a | b | c |
Frame 2 || a | b | c |
Frame 3 || a | b | c |
```
What is the goal of a page replacement algorithm?

- A. Make life easier for OS implementer
- B. Reduce the number of page faults
- C. Reduce the penalty for page faults when they occur
- D. Minimize CPU time of algorithm

**The Second Chance Algorithm**

Optimizing Approximate LRU Replacement

- Maintain a circular list of pages resident in memory
- Use a clock (or used/referenced) bit to track how often a page is accessed
- The bit is set whenever a page is referenced
- Clock hand sweeps over pages looking for one with used/referenced = 0
- Replace pages that haven’t been referenced for one complete revolution of the clock

**Clock Page Replacement**

Example

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frames</td>
<td>c a d b e b a c d</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Faults</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Example

Approximate LRU Page Replacement

The Clock algorithm

```
begin
while (victim page not found) do
    if (used bit for current page = 0) then
        replace current page
    else
        use both the dirty bit and the used bit to drive replacement
    end if
end while
```

**Optimizing Approximate LRU Replacement**

The Second Chance Algorithm

Example

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frames</td>
<td>c a d b e b a c d</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Faults</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Example

```
begin
while (victim page not found) do
    if (used bit for current page = 0) then
        replace current page
    else
        use both the dirty bit and the used bit to drive replacement
    end if
end while
```

**Clock Page Replacement**

Example

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frames</td>
<td>c a d b e b a c d</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Faults</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Example

```
begin
while (victim page not found) do
    if (used bit for current page = 0) then
        replace current page
    else
        use both the dirty bit and the used bit to drive replacement
    end if
end while
```
The Second Chance Algorithm

Example

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>b</td>
<td>d</td>
</tr>
</tbody>
</table>

Page Frames

0 | a | a | a | a | a | a | a | a | a | a | a |
1 | b | b | b | b | b | b | b | b | b | b | b |
2 | c | c | c | c | c | c | c | c | c | c | c |
3 | d | d | d | d | d | d | d | d | d | d | d |

Faults

* * * * * * * * * * * *

Page table entries for resident pages:

Page 0: 0
Page 1: 1
Page 2: 2
Page 3: 3

---

The Problem With Local Page Replacement

How much memory do we allocate to a process?

Time 0 1 2 3 4 5 6 7 8 9 10 11 12
Requests a b c d a b c d a b c d

Page Frames

0 | a |
1 | b |
2 | c |
3 | - |

Faults

* * * * * * * * * * * *

---

The Problem With Local Page Replacement

Example:

```
Page table entries for resident pages:

Page 0: 0
Page 1: 1
Page 2: 2
Page 3: 3
```

Replace a page that is not referenced in the next \( \tau \) accesses.

Example: \( \tau = 4 \)

---

Optimal Page Replacement

For processes with a variable number of frames

- VMIN — Replace a page that is not referenced in the next \( r \) accesses
- Example: \( r = 4 \)

---

Optimal Page Replacement

For processes with a variable number of frames

- VMIN — Replace a page that is not referenced in the next \( r \) accesses
- Example: \( r = 4 \)

---

The principle of locality

- 90% of the execution of a program is sequential
- Most iterative constructs consist of a relatively small number of instructions
- When processing large data structures, the dominant cost is sequential processing on individual structure elements
- Temporal vs. physical locality

---

Page Replacement Algorithms

Performance

- Local page replacement
  - LRU — Ages pages based on when they were last used
  - FIFO — Ages pages based on when they’re brought into memory
- Towards global page replacement ... with variable number of page frames allocated to processes
Explicitly Using Locality

The working set model of page replacement:
- Assume recently referenced pages are likely to be referenced again soon...
- ... and only keep those pages recently referenced in memory (called the working set)
- Thus pages may be removed even when no page fault occurs
- The number of frames allocated to a process will vary over time
- A process is allowed to execute only if its working set fits into memory
- The working set model performs implicit load control

Working Set Page Replacement
Implementation

- Keep track of the last \( r \) references
  - The pages referenced during the last \( r \) memory accesses are the working set
  - \( r \) is called the window size
- Example: Working set computation, \( r = 4 \) references:

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pages in Memory</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page a</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>±</td>
<td>±</td>
<td>±</td>
</tr>
<tr>
<td>Page b</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>·</td>
<td>·</td>
<td>·</td>
</tr>
<tr>
<td>Page c</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
</tr>
<tr>
<td>Page d</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
</tr>
<tr>
<td>Page e</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>±</td>
</tr>
<tr>
<td>Requests</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>c</td>
<td>d</td>
<td>b</td>
<td>c</td>
<td>e</td>
<td>e</td>
<td>a</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Faults</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>±</td>
<td>·</td>
<td>·</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
</tr>
</tbody>
</table>

Page-Fault-Frequency Page Replacement

An alternate working set computation:

- Explicitly attempt to minimize page faults
  - When page fault frequency is high — increase working set
  - When page fault frequency is low — decrease working set

Algorithm:
- Keep track of the rate at which faults occur
- When a fault occurs, compute the time since the last page fault
- Record the time, \( t_{last} \), of the last page fault
- If the time between page faults is "large" then reduce the working set
  - If \( t_{current} - t_{last} > \tau \), then remove from memory all pages not referenced in \( [t_{last}, t_{current}] \)
- If the time between page faults is "small" then increase working set
  - If \( t_{current} - t_{last} < \tau \), then add faulting page to the working set

Page-Fault-Frequency Page Replacement
Example, window size = 2

If \( t_{current} - t_{last} > 2 \), remove pages not referenced in \( [t_{last}, t_{current}] \) from the working set
If \( t_{current} - t_{last} \leq 2 \), just add faulting page to the working set

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pages in Memory</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page a</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>±</td>
<td>±</td>
<td>±</td>
</tr>
<tr>
<td>Page b</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>·</td>
<td>·</td>
<td>·</td>
</tr>
<tr>
<td>Page c</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
</tr>
<tr>
<td>Page d</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
</tr>
<tr>
<td>Page e</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>±</td>
</tr>
<tr>
<td>Requests</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>c</td>
<td>d</td>
<td>b</td>
<td>c</td>
<td>e</td>
<td>e</td>
<td>a</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Faults</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>±</td>
<td>·</td>
<td>·</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
</tr>
</tbody>
</table>

Page-Fault-Frequency Page Replacement
Example, window size = 2

If \( t_{current} - t_{last} > 2 \), remove pages not referenced in \( [t_{last}, t_{current}] \) from the working set
If \( t_{current} - t_{last} \leq 2 \), just add faulting page to the working set

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pages in Memory</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page a</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>±</td>
<td>±</td>
<td>±</td>
</tr>
<tr>
<td>Page b</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>·</td>
<td>·</td>
<td>·</td>
</tr>
<tr>
<td>Page c</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
</tr>
<tr>
<td>Page d</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
</tr>
<tr>
<td>Page e</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>±</td>
</tr>
<tr>
<td>Requests</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>c</td>
<td>d</td>
<td>b</td>
<td>c</td>
<td>e</td>
<td>e</td>
<td>a</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Faults</td>
<td>·</td>
<td>·</td>
<td>·</td>
<td>±</td>
<td>·</td>
<td>·</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
<td>±</td>
</tr>
</tbody>
</table>

\( t_{current} - t_{last} \)
Load Control

Fundamental tradeoff

- High multiprogramming level
  \[ MPL_{\text{max}} = \frac{\text{number of page frames}}{\text{minimum number of frames required for a process to execute}} \]
- Low paging overhead
  \[ MPL_{\text{min}} = 1 \text{ process} \]

Issues
- What criterion should be used to determine when to increase or decrease the MPL?
- Which task should be swapped out if the MPL must be reduced?

Load Control

How not to do it: Base load control on CPU utilization

- Assume memory is nearly full
- A chain of page faults occurs
  - A queue of processes forms at the paging device
- CPU utilization fails
  - Operating system increases MPL
  - New processes fault, taking memory away from existing processes
  - CPU utilization goes to 0, the OS increases the MPL further...

System is thrashing — spending all of its time paging

Load Control

Thrashing

- Thrashing can be ameliorated by local page replacement
- Better criteria for load control: Adjust MPL so that:
  - mean time between page faults (MTBF) \(<\) page fault service time (PFST)
  - \[ \sum WS_i = \text{size of memory} \]

Load Control

Thrashing

- When the multiprogramming level should be decreased, which process should be swapped out?
  - Lowest priority process?
  - Smallest process?
  - Largest process?
  - Oldest process?
  - Faulting process?