CS 333
Introduction to Operating Systems

Class 14 – Page Replacement

Jonathan Walpole
Computer Science
Portland State University
Page replacement

- Assume a normal page table (e.g., BLITZ)
- User-program is executing
- A PageInvalidFault occurs!
  - The page needed is not in memory
- Select some frame and remove the page in it
  - If it has been modified, it must be written back to disk
    - the “dirty” bit in its page table entry tells us if this is necessary
- Figure out which page was needed from the faulting addr
- Read the needed page into this frame
- Restart the interrupted process by retrying the same instruction
Page replacement algorithms

- Which frame to replace?

  - **Algorithms:**
    - The Optimal Algorithm
    - First In First Out (FIFO)
    - Not Recently Used (NRU)
    - Second Chance / Clock
    - Least Recently Used (LRU)
    - Not Frequently Used (NFU)
    - Working Set (WS)
    - WSClock
The optimal page replacement algorithm

- **Idea:**
  - Select the page that will not be needed for the longest time
Optimal page replacement

- Replace the page that will not be needed for the longest.
- 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>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</th>
<th>0</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frames</td>
<td>1</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
</tr>
</tbody>
</table>

Page faults: x
Optimal page replacement

- Select the page that will not be needed for the longest time
- 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>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

| Page | 0 | a | a | a | a | a | a | a | a | a | a |
| Frames | 1 | 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 | e | e | e | e | e | e |

Page faults

X    X
The optimal page replacement algorithm

- **Idea:**
  - Select the page that will not be needed for the longest time

- **Problem?**
The optimal page replacement algorithm

- **Idea:**
  - Select the page that will not be needed for the longest time

- **Problem:**
  - Can’t know the future of a program
  - Can’t know when a given page will be needed next
  - The optimal algorithm is unrealizable
The optimal page replacement algorithm

- **However:**
  - We can use it as a control case for simulation studies
    - Run the program once
    - *Generate a log of all memory references*
      - Do we need all of them?
    - Use the log to simulate various page replacement algorithms
    - Can compare others to “optimal” algorithm
FIFO page replacement algorithm

- **Always replace the oldest page ...**
  - “Replace the page that has been in memory for the longest time.”
**FIFO page replacement algorithm**

- Replace the page that was first brought into memory
- Example: Memory system with 4 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>a</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</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>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td></td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td></td>
<td>d</td>
<td>d</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Page faults

\[x\]
FIFO page replacement algorithm

- Replace the page that was first brought into memory
- Example: Memory system with 4 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>a</td>
<td>a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</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>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Page faults: X X
FIFO page replacement algorithm

- Replace the page that was first brought into memory

- Example: Memory system with 4 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>a</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</th>
<th>0</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frames</td>
<td>1</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>d</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
</tbody>
</table>

Page faults

X

X

X
**FIFO page replacement algorithm**

- Replace the page that was first brought into memory
- Example: Memory system with 4 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>a</td>
<td></td>
</tr>
<tr>
<td>Page 0</td>
<td>a</td>
<td></td>
<td></td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>Frames 1</td>
<td>b</td>
<td></td>
<td></td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td></td>
<td></td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>3</td>
<td>d</td>
<td></td>
<td></td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>Page faults</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**FIFO page replacement algorithm**

- **Always replace the oldest page.**
  
  “Replace the page that has been in memory for the longest time.”

- **Implementation**
  
  - Maintain a linked list of all pages in memory
  - Keep it in order of when they came into memory
  - The page at the tail of the list is oldest
  - Add new page to head of list
FIFO page replacement algorithm

- Disadvantage?
FIFO page replacement algorithm

- **Disadvantage:**
  - The oldest page may be needed again soon
  - Some page may be important throughout execution
  - It will get old, but replacing it will cause an immediate page fault
How can we do better?

- Need an approximation of how likely each frame is to be accessed in the future
  - If we base this on past behavior we need a way to track past behavior
  - Tracking memory accesses requires hardware support to be efficient
Page table: referenced and dirty bits

- Each page table entry (and TLB entry!) has a
  - Referenced bit - set by TLB when page read / written
  - Dirty / modified bit - set when page is written
  - If TLB entry for this page is valid, it has the most up to date version of these bits for the page
    - OS must copy them into the page table entry during fault handling

- Idea: use the information contained in these bits to drive the page replacement algorithm
Page table: referenced and dirty bits

- Some hardware does not have support for the dirty bit
- Instead, memory protection can be used to emulate it
- **Idea:**
  - Software sets the protection bits for all pages to “read only”
  - When program tries to update the page...
    - A trap occurs
    - Software sets the Dirty Bit in the page table and clears the ReadOnly bit
    - Resumes execution of the program
Not recently used page replacement alg.

- Uses the Referenced Bit and the Dirty Bit

- Initially, all pages have
  - Referenced Bit = 0
  - Dirty Bit = 0

- Periodically... (e.g. whenever a timer interrupt occurs)
  - Clear the Referenced Bit
  - Referenced bit now indicates “recent” access
Not recently used page replacement alg.

- When a page fault occurs...

- Categorize each page...
  - **Class 1**: Referenced = 0 Dirty = 0
  - **Class 2**: Referenced = 0 Dirty = 1
  - **Class 3**: Referenced = 1 Dirty = 0
  - **Class 4**: Referenced = 1 Dirty = 1

- Choose a victim page from class 1 ... why?
- If none, choose a page from class 2 ... why?
- If none, choose a page from class 3 ... why?
- If none, choose a page from class 4 ... why?
Second chance page replacement alg.

- An implementation of NRU based on FIFO
- Pages kept in a linked list
  - Oldest is at the front of the list
- Look at the oldest page
  - If its “referenced bit” is 0...
    - Select it for replacement
  - Else
    - It was used recently; don’t want to replace it
    - Clear its “referenced bit”
    - Move it to the end of the list
  - Repeat
- What if every page was used in last clock tick?
  - Select a page at random
Clock algorithm (an implementation of NRU)

- Maintain a circular list of pages in memory
- Set a bit for the page when a page is referenced
- Clock sweeps over memory looking for a victim page that does not have the referenced bit set
  - If the bit is set, clear it and move on to the next page
  - Replaces pages that haven’t been referenced for one complete clock revolution – essentially an implementation of NRU
Least recently used algorithm (LRU)

- A refinement of NRU that orders how recently a page was used
  - Keep track of when a page is used
  - Replace the page that has been used least recently
LRU page replacement

- Replace the page that hasn’t been referenced in the longest time

<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>a</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

| Page | 0 | a | a | a | a | a | a | a | a | a | a |
| Frames | 1 | b | b | b | b | b | b | b | b | b | b |
| 2 | c | c | c | c | c | e | e | e | e | e | e |
| 3 | d | d | d | d | d | d | d | d | c | c | c |

Page faults

- X
- X
- X
Least recently used algorithm (LRU)

- But how can we implement this?
Least recently used algorithm (LRU)

- But how can we implement this?

- **Implementation #1:**
  - Keep a linked list of all pages
  - On every memory reference,
    - Move that page to the front of the list
  - The page at the tail of the list is replaced
Least recently used algorithm (LRU)

But how can we implement this?
- ... without requiring “every access” to be recorded?

Implementation #2:
- MMU (hardware) maintains a counter
- Incremented on every clock cycle
- Every time a page table entry is used
  - MMU writes the value to the page table entry
  - This timestamp value is the time-of-last-use
- When a page fault occurs
  - Software looks through the page table
  - Identifies the entry with the oldest timestamp
Least recently used algorithm (LRU)

- What if we don’t have hardware support for a counter?

- **Implementation #3:**
  - Maintain a counter in software
  - One every timer interrupt...
    - Increment counter
    - Run through the page table
    - For every entry that has “ReferencedBit” = 1
      - Update its timestamp
      - Clear the ReferencedBit
  - Approximates LRU
  - If several have oldest time, choose one arbitrarily
Not frequently used algorithm (NFU)

- Bases decision of frequency of use rather than recency
- Associate a counter with each page
- On every clock interrupt, the OS looks at each page.
  - If the Reference Bit is set...
    - Increment that page’s counter & clear the bit.
- The counter approximates how often the page is used.
- For replacement, choose the page with lowest counter.
Not frequently used algorithm (NFU)

- **Problem:**
  - Some page may be heavily used
    - $\implies$ Its counter is large
  - The program’s behavior changes
    - Now, this page is not used ever again (or only rarely)
  - This algorithm never forgets!
    - *This page will never be chosen for replacement!*
  - We may want to combine frequency and recency
Modified NFU with aging

- Associate a counter with each page
- On every clock tick, the OS looks at each page.
  - Shift the counter right 1 bit (divide its value by 2)
  - If the Reference Bit is set...
    - Set the most-significant bit
    - Clear the Referenced Bit

<table>
<thead>
<tr>
<th>Page</th>
<th>Binary</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₁</td>
<td>100000</td>
<td>32</td>
</tr>
<tr>
<td>T₂</td>
<td>010000</td>
<td>16</td>
</tr>
<tr>
<td>T₃</td>
<td>001000</td>
<td>8</td>
</tr>
<tr>
<td>T₄</td>
<td>000100</td>
<td>4</td>
</tr>
<tr>
<td>T₅</td>
<td>100010</td>
<td>34</td>
</tr>
</tbody>
</table>
Working set page replacement

- **Demand paging**
  - Pages are only loaded when accessed
  - When process begins, all pages marked INVALID
Working set page replacement

- **Demand paging**
  - Pages are only loaded when accessed
  - When process begins, all pages marked INVALID

- **Locality of reference**
  - Processes tend to use only a small fraction of their pages
Working set page replacement

- **Demand paging**
  - Pages are only loaded when accessed
  - When process begins, all pages marked INVALID

- **Locality of Reference**
  - Processes tend to use only a small fraction of their pages

- **Working Set**
  - The set of pages a process needs
  - If working set is in memory, no page faults
  - What if you can’t get working set into memory?
Working set page replacement

- **Thrashing**
  - If you can’t get working set into memory page faults occur every few instructions
  - Little work gets done
  - Most of the CPU’s time is going on overhead
Working set page replacement

- **Based on prepaging (prefetching)**
  - Load pages before they are needed

- **Main idea:**
  - Try to identify the process’s “working set”

- **How big is the working set?**
  - Look at the last K memory references
  - As K gets bigger, more pages needed.
  - In the limit, all pages are needed.
Working set page replacement

- The size of the working set:
Working set page replacement

- **Idea:**
  - Look back over the last T msec of time
  - Which pages were referenced?
    - This is the working set.

- **Current Virtual Time**
  - Only consider how much CPU time this process has seen.

- **Implementation**
  - On each clock tick, look at each page
  - Was it referenced?
    - Yes: Make a note of Current Virtual Time
  - If a page has not been used in the last T msec,
    - It is not in the working set!
    - Evict it; write it out if it is dirty.
Working set page replacement

Page table

- Information about one page
  - 2084
  - 2003
- Time of last use
  - 1980
- Page referenced during this tick
  - 1213
  - 2014
  - 2020
- Page not referenced during this tick
  - 2032
  - 1620

R (Referenced) bit

Scan all pages examining R bit:
- if (R == 1)
  set time of last use to current virtual time
- if (R == 0 and age > τ)
  remove this page
- if (R == 0 and age ≤ τ)
  remember the smallest time

Current virtual time: 2204
WSClock page replacement algorithm

- An implementation of the working set algorithm
- All pages are kept in a circular list (ring)
- As pages are added, they go into the ring
- The “clock hand” advances around the ring
- Each entry contains “time of last use”
- Upon a page fault...
  - If Reference Bit = 1...
    - Page is in use now. Do not evict.
    - Clear the Referenced Bit.
    - Update the “time of last use” field.
WSClock page replacement algorithm

- If Reference Bit = 0
  - If the age of the page is less than T...
    - This page is in the working set.
    - Advance the hand and keep looking
  - If the age of the page is greater than T...
    - If page is clean
      - Reclaim the frame and we are done!
    - If page is dirty
      - Schedule a write for the page
      - Advance the hand and keep looking
But which algorithm is best?
Comparing algorithms through modeling

- Run a program
  - Look at all memory references
  - Don’t need all this data
  - Look at which pages are accessed
    000001222333300114444001123444
  - Eliminate duplicates
    012301401234

- This defines the **Reference String**
  - Use this to evaluate different page replacement algorithms
  - Count page faults given the same reference string
Proactive use of replacement algorithm

- Replacing victim frame on each page fault typically requires two disk accesses per page fault.

- Alternative → the O.S. can keep several pages free in anticipation of upcoming page faults.
  
  In Unix: low and high water marks

```
  low water mark
  high water mark

  low < # free pages < high
```
The UNIX memory model

- UNIX page replacement
  - clock algorithm for page replacement
    - If page has not been accessed move it to the free list for use as allocatable page
      - If modified/dirty → write to disk (still keep stuff in memory though)
      - If unmodified → just move to free list
  - High and low water marks for free pages
  - Pages on the free-list can be re-allocated if they are accessed again before being overwritten
Local vs. global page replacement

- Assume several processes: A, B, C, ...
- Some process gets a page fault (say, process A)
- Choose a page to replace.

- **Local page replacement**
  - Only choose one of A’s pages

- **Global page replacement**
  - Choose any page
Local vs. global page replacement

- **Example:** Process has a page fault...

<table>
<thead>
<tr>
<th></th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>10</td>
</tr>
<tr>
<td>A1</td>
<td>7</td>
</tr>
<tr>
<td>A2</td>
<td>5</td>
</tr>
<tr>
<td>A3</td>
<td>4</td>
</tr>
<tr>
<td>A4</td>
<td>6</td>
</tr>
<tr>
<td>A5</td>
<td>3</td>
</tr>
<tr>
<td>B0</td>
<td>9</td>
</tr>
<tr>
<td>B1</td>
<td>4</td>
</tr>
<tr>
<td>B2</td>
<td>6</td>
</tr>
<tr>
<td>B3</td>
<td>2</td>
</tr>
<tr>
<td>B4</td>
<td>5</td>
</tr>
<tr>
<td>B5</td>
<td>6</td>
</tr>
<tr>
<td>B6</td>
<td>12</td>
</tr>
<tr>
<td>C1</td>
<td>3</td>
</tr>
<tr>
<td>C2</td>
<td>5</td>
</tr>
<tr>
<td>C3</td>
<td>6</td>
</tr>
</tbody>
</table>

**Original**

- A0
- A1
- A2
- A3
- A4
- A5
- B0
- B1
- B2
- B3
- B4
- B5
- B6
- C1
- C2
- C3

**Local**

- A0
- A1
- A2
- A3
- A4
- A5
- B0
- B1
- B2
- B3
- B4
- B5
- B6
- C1
- C2
- C3

**Global**

- A0
- A1
- A2
- A3
- A4
- A5
- B0
- B1
- B2
- B3
- B4
- B5
- B6
- C1
- C2
- C3
Local vs. global page replacement

- Assume we have
  - 5,000 frames in memory
  - 10 processes
- Idea: Give each process 500 frames

- Fairness?
  - Small processes: do not need all those pages
  - Large processes: may benefit from even more frames

- Idea:
  - Look at the size of each process (... but how?)
  - Give them a pro-rated number of frames
  - With a minimum of (say) 10 frames per process
Page fault frequency

“If you give a process more pages, its page fault frequency will decline.”
Page fault frequency

- “If you give a process more pages, its page fault frequency will decline.”

Too High: Need to give this process some more frames!

Too Low: Take some frames away and give to other processes!
Page fault frequency

- Measure the page fault frequency of each process.
- Count the number of faults every second.
- May want to consider the past few seconds as well.
Page fault frequency

- Measure the page fault frequency of each process.
- Count the number of faults every second.
- May want to consider the past few seconds as well.

- **Aging:**
  - Keep a running value.
  - Every second
    - Count number of page faults
    - Divide running value by 2
    - Add in the count for this second
Load control

- **Assume:**
  - The best page replacement algorithm
  - Optimal global allocation of page frames
Load control

- **Assume:**
  - The best page replacement algorithm
  - Optimal global allocation of page frames

- **Thrashing is still possible!**
Load control

- **Assume:**
  - The best page replacement algorithm
  - Optimal global allocation of page frames

- **Thrashing is still possible!**
  - Too many page faults!
  - No useful work is getting done!
  - Demand for frames is too great!
Load Control

- **Assume:**
  - The best page replacement algorithm
  - Optimal global allocation of page frames

- **Thrashing is still possible!**
  - Too many page faults!
  - No useful work is getting done!
  - Demand for frames is too great!

- **Solution:**
  - Get rid of some processes (temporarily).
  - Swap them out.
  - “Two-level scheduling”