Virtual Memory

Role of OS in virtual memory management
Role of OS memory management

- Design of memory-management portion of OS depends on 3 fundamental areas of choice
  - Whether to use virtual memory or not
  - Whether to use paging, segmentation, or both
  - Algorithms employed for various aspects of memory management
OS Policies for virtual memory

- Algorithms for:
  - Fetch policy
  - Placement Policy
  - Replacement policy
  - Resident Set Management
  - Cleaning policy
  - Load Control
Fetch policy

- When to bring page into memory
- Demand paging:
  - fetch when a reference is made to a location in the page
- Prepaging
  - Load referenced page as well as “n” neighboring pages
  - Bring in more pages than needed
  - Exploits the characteristics of most secondary devices
    - If the pages of a process are stored in contiguous locations, then it may be effective.
Placement policy

- Where to place a portion
- With segments:
  - Use best-fit, first-fit etc to minimize external fragmentation
- With paging or paging combined with segmentation:
  - Placement is not an issue.
  - Use any available frame
Which page to replace

- Resident set management
  - How many pages per process in main memory
  - When bringing in a new page:
    - should the page being replaced be that of the process that caused the page fault or
    - could it belong to any process

- Replacement policy
  - Among pages eligible to be replaced, which one should be replaced to keep the page faults at a minimum?
  - Must be the one least likely to be accessed in immediate future
Page replacement: demand paging

- Static page replacement algorithms: fixed number of pages allocated to process
  - Optimal
  - Least Recently Used
  - First-in, First-out
  - Clock algorithm

- Dynamic Paging Algorithms
  - Working Set Algorithms
Static page replacement algorithm

- **Optimal:**
  - **Idea:** Replace page for which the time to the next reference is the longest
  - Impossible to implement as must know what will happen in the future.

- **First-in, first-out (FIFO):**
  - **Idea:** Replace page that has been in memory the longest (i.e., the oldest page)
  - Treats frames allocated to a process as a circular buffer
  - Pages are removed in round-robin order
  - Simplest replacement policy to implement
### Figure 8.15  Behavior of Four Page-Replacement Algorithms

<table>
<thead>
<tr>
<th>Page address stream</th>
<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>OPT</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIFO</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>F</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*F = page fault occurring after the frame allocation is initially filled*
Static page replacement algorithm

- Least Recently Used (LRU):
  - **Idea:** Replace the page that hasn't been referenced for the longest time in the past
  - By the principle of locality, this should be the page least likely to be referenced in the near future
  - Does almost as well as optimal algorithm on some reference sequences
  - **Difficult to implement in hardware**
    - Each page could be tagged with the time of last reference.
    - This would require a great deal of overhead
<table>
<thead>
<tr>
<th>Page address stream</th>
<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><strong>OPT</strong></td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</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>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></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>
<td>5</td>
</tr>
<tr>
<td><strong>FIFO</strong></td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td>3</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></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>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td><strong>LRU</strong></td>
<td>2</td>
<td>2</td>
<td>2</td>
<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>
</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>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</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>
<tr>
<td></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>
<td>2</td>
</tr>
</tbody>
</table>

$F =$ page fault occurring after the frame allocation is initially filled

**Figure 8.15  Behavior of Four Page-Replacement Algorithms**
Static page replacement algorithm

- **Clock policy:**
  - Every frame has a *used bit* (U) that is set to 1 when a page is first loaded in it.
  - U is set to 1 every time the page is referenced.
  - Put pages on a circular buffer in the form of a clock with a “hand” (pointer) referencing the “next” page.
    - Next page: the page after the one that was just replaced in the circular buffer.
  - When page fault occurs,
    - if “next” page has U = 0 then replace
    - otherwise set U = 0 and advance to next page pointer.
    - Keep advancing until locate page with U = 0.
Need to load page 727
Which page to replace?

Page to replace

Page replaced and
Next pointer advanced

Figure 8.16 Example of Clock Policy Operation
Figure 8.15  Behavior of Four Page-Replacement Algorithms

F = page fault occurring after the frame allocation is initially filled
Variation of clock policy

- Variation of simple clock policy – make more powerful by increasing the # of bits.
- u (use bit), m (modify bit).
  - Not accessed recently, not modified (u=0,m=0)
  - Not accessed recently, modified (u=0,m=1)
  - Accessed recently, not modified (u=1,m=0)
  - Access recently, modified recently (u=1,m=1)
- Replace pages that have not been used recently else not modified.
Variation of clock policy

- **Step 1:**
  - Start scanning frame buffer from current pointer position and **make no chances to use bit**
  - replace first page with $u=0$, $m=0$

- **Step 2:**
  - If step 1 fails, scan again, setting each $u = 0$ for each frame bypassed
  - replace first page with $u=0$, $m=1$

- **Step 3:**
  - If step 2 fails, repeat 1
Page buffering

- A page that is to be replaced remains in memory for some more time and will require **no disk access** if it is needed in the immediate future.
- Also, since a list of modified pages is maintained, when writing back to disk, can **write all the modified pages** together.
- Used along with a simple page replacement scheme such as FIFO.
Page buffering

- A few frames are not allotted to any process
- When a page is to be replaced, its page table entry is added to one of two lists:
  - Free page list, if page not modified
  - Modified page list, if page modified
- When a new page is brought in,
  - it is placed in the frame vacated by the page whose page table entry is in the front of the free/modified page list,
  - and the page table entry of the page to be replaced is placed at the back of the free/modified page list.
Resident Set Management

- Resident set size
  - How many pages per process in main memory
- Replacement scope
  - When bringing in a new page,
    - should the page being replaced be that of the process that caused the page fault or
    - could it belong to any process
Resident set size

- **Fixed allocation**
  - fixed number of frames in main memory for a process
  - Number decided at creation time
  - Page to be replaced must be from the same process.

- **Variable allocation**
  - number of frames allocated can change during the duration of the process.
  - If there are many page faults, then increase the number of frames allocated and vice versa.

- Operating system has to observe program behavior and predict future behavior.
Replacement scope

- Local scope
  - consider only frames allocated to the process that caused the page fault as candidates for replacement

- Global scope
  - consider all unlocked frames in main memory as candidates for replacement.
  - simpler to implement and more commonly used
Resident set size/replacement scope

• Fixed allocation, local scope
• Variable allocation, global scope
  • Easiest to implement
  • Adopted by many operating systems
  • Operating system keeps list of free frames
  • Free frame is added to resident set of process when a page fault occurs
  • If no free frame, replaces one from another process
Resident set size/replacement scope

- Variable allocation, local scope
  - When new process added, allocate number of page frames based on
    - application type,
    - program request, or
    - other criteria
  - When a process page faults frequently, increase its resident set size.
  - Periodically, reevaluate the allocations to various processes and change resident set sizes.
Cleaning Policy

- When should a modified page be written back to disk.
- Demand cleaning
  - write when a page has to be replaced
  - Process that suffers a page fault must wait for two disk accesses before it can be unblocked =>
    - processor under-utilized.
- Precleaning
  - write pages in batches
  - a page may be written to disk many times before it is actually replaced

*Best approach uses page buffering*
Load control

- Determines the number of processes that will be resident in main memory
  - Too few resident processes could result in processor being idle.
  - Too many resident processes could result in frequent page faults and thrashing.
Load control and multiprogramming

- Goal is to maximize processor utilization
- Adjust the level of multiprogramming such that the mean time between faults equals the mean time required to process a page fault
  - This is when processor utilization is maximum
Load control and multiprogramming

- 50% criterion
  - keep utilization of the paging device at 50%
- Adapt the clock page replacement algorithm
  - Monitor the rate at which the pointer scans the circular buffer of frames
    - Few page faults => few requests to advance the pointer
    - For each request, if the number of pages being scanned is small => many pages in the resident set are not being accessed frequently and do not need to be in memory
    - Can decrease the resident set size and increase the multiprogramming level.
Process suspension

- Lowest priority process
- Faulting process
  - this process does not have its working set in main memory so it will be blocked anyway
- Last process activated
  - this process is least likely to have its working set resident
- Process with smallest resident set
  - this process requires the least future effort to reload
- Largest process
  - obtains the most free frames
- Process with the largest remaining execution window