Preview

- Page Replacement Algorithms
  - Optimal Algorithm
  - First In First Out (FIFO)
  - Not Recently Used Algorithm (NRU)
  - The Least Recently Used (LRU)
- Modeling Page Replacement Algorithm
  - Belady’s Anomaly
  - Stack Algorithm
    - Model for Stack Algorithm
    - Property of Stack Algorithm
- Design Issues for Page System
- Segmentation
  - Segmentation Implementation
  - Segmentation with Paging
Page Replacement Algorithms

When a page fault occurs, OS needs to do the following steps:

1. Choose a victim page currently allocated in a page frame
2. If the page has been modified in the memory, rewrite it to the disk
3. A page is allocated into the page frame which was used by the victim page
4. Change page table

The first step depends on the replacement algorithm
Page Replacement Algorithms

- A process’s memory access can be characterized by a list of page numbers.
- This list is called the reference string.
- A paging system can be characterized by three items:
  1. The reference string of the executing process
  2. The page replacement algorithm
  3. The number of page frames available in memory
Page Replacement Algorithms

Optimal Algorithm

- Replace the page that will not be used for the longest period of time in the future.
- Optimal algorithm always guarantees the lowest possible page-fault rate for a fixed number of frames.
- Unfortunately, there is no such an optimal replacement algorithm, as it requires future knowledge of the reference string.
Page Replacement Algorithms

Optimal Algorithm

9 page Faults
Page Replacement Algorithms

Not Recently Used (NRU)

- When a page fault occurs, the operating system inspects all the pages and classifies them into four groups based on the page table information (Modified bit, Reference bit).
  - Class 0: Not referenced, not modified
  - Class 1: not referenced, modified
  - Class 2: referenced, not modified
  - Class 3: referenced, modified
- The NRU algorithm removes a page at random from the lowest numbered nonempty class.
Page Replacement Algorithms

First In First Out (FIFO)

- Replace the oldest page frame
- The FIFO algorithm is easy to understand and easy to program.
- However, its performance is not always good.
Page Replacement Algorithms

First In First Out

15 Page Faults
Page Replacement Algorithms

The Least Recently Used

- Replace the page that has not been used for the longest period of time.
- This is the optimal page replacement algorithm looking backward in time.
- LRU algorithms work quite well but it may require substantial hardware assistance to keep track of the information.
### Least Recently Used

The image shows a table representing page faults for the Least Recently Used (LRU) algorithm. The table has 7 columns representing different time points, and 5 rows indicating the pages in the system. The page numbers are represented by 0, 1, 2, 3, 4, 5, and 7. The number of page faults is shown at the bottom, which is 12.

<table>
<thead>
<tr>
<th>Time</th>
<th>Page 0</th>
<th>Page 1</th>
<th>Page 2</th>
<th>Page 3</th>
<th>Page 4</th>
<th>Page 5</th>
<th>Page 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
</tbody>
</table>

**12 Page Faults**
Modeling Page Replacement Algorithm

Belady’s Anomaly

- Intuitively, it might seem that the more page frames the memory has, the fewer page faults a program will cause. This is not always the case – Belady’s Anomaly
- Belady shows the case with FIFO cases; the more page frames the memory has, the more page faults a program causes.
Modeling Page Replacement Algorithm (Belady’s Anomaly)

9 page faults

10 page faults
Modeling Page Replacement Algorithm (Stack Algorithm)

A paging system can be characterized by three items:

1. The reference string of the executing process
2. The page replacement algorithm
3. The number of page frames available in memory
Stack Algorithm

Model for Stack Algorithm

- Maintains an internal array M that keeps track of the state of memory.
- M has as many as \( n \) virtual memory pages.
- Top \( m \) entries contain all the pages currently in the memory (page frames).
- Bottom \( n - m \) entries contains all the pages that have been referenced once but have been page out and are not currently in memory.
Stack Algorithm
(Model for Stack Algorithm)

Properties of the model

1. When a page is referenced, it is always moved to the top entry in M.
2. If the page referenced was already in M, all pages above it move down one position.
3. A transition from within the box to outside of it corresponds to a page being evicted from the memory.
4. The pages that were below the referenced page are not moved.
Stack Algorithm
(Property for Stack Algorithm)

Property of Stack Algorithm
- If any page replacement algorithm has the following property, we call it the stack algorithm

\[ M(m, r) \subseteq M(m + 1, r) \]

- \( m \) – varies over page frames
- \( r \) – is an index into the reference string

- If we increase memory size by one page frame and re-execute the process, at every point during the execution, all the pages that were present in the first run are also present in the second run, along with one additional page.
- Stack algorithm does not suffer from Belady’s Anomaly.
Stack Algorithm

LRU algorithm with the model

1. The reference strings: 0 2 1 3 5 4 6 3 7 4 7 3 3 5 3 1 1 1 7 2 3 4 1
2. The page replacement algorithm: LRU
3. The number of page frames available in memory:
   1. Virtual spaces: 8 pages
   2. Physical spaces: 4 pages or
   3. Physical spaces: 5 pages
Stack Algorithm

Reference string: 0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 1 3 4 1

Page faults: P P P P P P P P

Distance string: ∞ ∞ ∞ ∞ ∞ 4 ∞ 4 2 3 1 5 1 2 6 1 1 4 2 3 5 3

Reference string: 0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 1 3 4 1

Page faults: P P P P P P P

Distance string: ∞ ∞ ∞ ∞ ∞ 4 ∞ 4 2 3 1 5 1 2 6 1 1 4 2 3 5 3
Design Issues for Paging System
(Local vs. Global Allocation)

Local versus Global allocation Policies

- With multiple processes competing for frames, we can classify page replacement algorithms into two categories: global replacement and local replacement.
- Global replacement allows a process to select a replacement frame from the set of all frames.
- Local replacement requires that each process select from only its own set of allocated frame.
Design Issues for Paging System
(Page Size)

- Smaller page size means smaller internal fragmentation.
- Larger size will cause more unused program to be in memory than a small page size.
- Small pages means that programs will need many pages (hence a large page tables).
- Many small pages results in wasting time for the seek and rotation.
Design Issues for Paging System
(Page Size)

Mathematical Analysis

S: average size of process (byte)
P: the size of page (byte)
E: Each page table entry needs (byte) per page.

\[ \frac{S}{P} : \text{Average number of pages per process} \]
\[ \frac{S}{P} \times E : \text{Average page table space (for each process)} \]
\[ \frac{P}{2} : \text{the wasted memory (in average) in the last page of the process} \]
Design Issues for Paging System

(Page Size)

- $\frac{S}{P}$: Average number of pages per process
- $\frac{S}{P} \times E$: Average page table space (for each process)
- $\frac{P}{2}$: the wasted memory (in average) in the last page of the process

Total overhead by page table and internal fragmentation loss is:

$Overhead(P) = \frac{SE}{P} + \frac{P}{2}$

$Overhead'(P) = -\frac{SE}{P^2} + \frac{1}{2} = 0$

$P = \sqrt{2SE}$: optimal page size
Segmentation

- Segmentation is a logical entity, which the programmer knows and uses as a logical entity.
- Segmentation is a memory management scheme that supports the user’s view of memory!!
Segmentation
(User’s View of Memory)
Segmentation

- A logical address space is a collection of segments. Each segment has a name and a length.
- Each segment might have a different size.
- A segment might contain a procedure, an array, a stack, or a collection of scalar variables; but usually it does not contain a mixture of different types.
- Normally, the user program is compiled, and the compiler automatically constructs segments reflecting the input program.
Segmentation

Ex) Pascal compiler might create separate segments for following:

- The global variables;
- The procedure call stack, to store parameters and return address
- The code portion of each procedure or function
- The local variables of each procedure and function
Ex) A reference to segment 3 to byte 852 is mapped to
3200 + 852 = 4052

Ex) A reference to segment 0 to byte 1222
would result in error!
Disadvantages with Segmentation
(External Fragmentation)
Segmentation with Paging

Reason for segmentation with paging

- If the segments are large, then keeping them in the physical memory might be wasting memory space.
- If a segment’s virtual space is larger than physical space, it is not even possible to keep them in the physical memory.
- A solution is the segmentation with paging – each segment is divided into pages.