Segmentation

- May be unequal, dynamic size
- Simplifies handling of growing data structures
- Allows programs to be altered and recompiled independently
- Lends itself to sharing data among processes
- Lends itself to protection

Segment Tables

- Corresponding segment in main memory
- Each entry contains the length of the segment
- A bit is needed to determine if segment is already in main memory
- Another bit is needed to determine if the segment has been modified since it was loaded in main memory
Segment Table Entries

(b) Segmentation only

Figure 8.12  Address Translation in a Segmentation System
Combined Paging and Segmentation

- Paging is transparent to the programmer
- Segmentation is visible to the programmer
- Each segment is broken into fixed-size pages
Figure 8.13 Address Translation in a Segmentation/Paging System

Figure 8.14 Protection Relationships Between Segments
Fetch Policy

- Fetch Policy
  - Determines when a page should be brought into memory
  - **Demand paging** only brings pages into main memory when a reference is made to a location on the page
    - Many page faults when process first started
  - **Prepaging** brings in more pages than needed
    - More efficient to bring in pages that reside contiguously on the disk

Placement Policy

- Determines where in real memory a process piece is to reside
- Important in a segmentation system
- Paging or combined paging with segmentation hardware performs address translation
Replacement Policy

• Placement Policy
  – Which page is to be replaced?
  – Page removed should be the page least likely to be referenced in the near future
  – Most policies predict the future behavior on the basis of past behavior

Replacement Policy

• Frame Locking
  – If frame is locked, it may not be replaced
  – Kernel of the operating system
  – Control structures
  – I/O buffers
  – Associate a lock bit with each frame
Basic Replacement Algorithms

• Optimal policy
  – Selects for replacement that page for which the time to the next reference is the longest
  – Impossible to have perfect knowledge of future events
  – This policy is “wishful thinking”, but can serve as a base-line when post-evaluating different policies

• Least Recently Used (LRU)
  – Replaces the page that has not been referenced for the longest time
  – By the principle of locality, this should be the page least likely to be referenced in the near future
  – Each page could be tagged with the time of last reference. This would require a great deal of overhead.
Basic Replacement Algorithms

• First-in, first-out (FIFO)
  – Treats page frames allocated to a process as a circular buffer
  – Pages are removed in round-robin style
  – Simplest replacement policy to implement
  – Page that has been in memory the longest is replaced
  – These pages may be needed again very soon
  – Performs relatively poorly

Basic Replacement Algorithms

• Clock Policy
  – Additional bit called a *use* bit
  – When a page is first loaded in memory, the *use* bit is set to 1
  – When the page is referenced, the use bit is set to 1
  – When it is time to replace a page, the first frame encountered with the *use* bit set to 0 is replaced.
  – During the search for replacement, each *use* bit set to 1 is changed to 0
Figure 8.15  Behavior of Four Page-Replacement Algorithms
Comparison of Placement Algorithms

Figure 8.16 Example of Clock Policy Operation

Figure 8.17 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

case study: page size = 256 words