CS 355
Operating Systems
Paging Algorithms and Segmentation

The Working Set
• Processes exhibit locality of memory reference
  – During any phase of execution, the process
    references only a relatively small fraction of its
    pages.
• The set of pages a process is currently using is
called its working set
• Demand paging
  – A process starts up with no pages in memory and a
    bunch of page faults until the working set is in
    memory
• If memory is too small to hold the entire
  working set, it will lead to trashing
• What happens during a context switch?

Working Set Model
• Designed to greatly reduce the page fault rate
• Pages are loaded before letting a process run
  – this is also known as pre-paging
• At any instance of time $t$, there exists a set $w(k, t)$ consisting of all pages used by the $k$
  most recent memory references.
• The working set changes over time
• References tend to cluster on a small number
  of pages: data, instruction and stack

The Working Set
Page Replacement Algorithm
• Keeping track of all pages used in the last $k$
  memory references is expensive
• Instead, record pages used in the last time
  interval, say 100 msec.
• Virtual time (kept per process)
• The working set is the set of pages referenced
during the past $t$ seconds of virtual time

The Working Set
Page Replacement Algorithm
• Algorithm: replace the page not in working set
• Page table: clock tick clears R bit periodically
• On page fault, scan all R bits.
  – If 1, the current virtual time is written into tolu
    (time of last use) field.
  – If 0, compute age of page with virtual-tolu.
• If age is smaller than time interval $t$?
The Working Set Page Replacement Algorithm

Review of Page Replacement Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>Not implementable, but useful as a benchmark</td>
</tr>
<tr>
<td>NRU (Not Recently Used)</td>
<td>Very crude</td>
</tr>
<tr>
<td>FIFO (First-In, First-Out)</td>
<td>Might throw out important pages</td>
</tr>
<tr>
<td>Second chance</td>
<td>Big improvement over FIFO</td>
</tr>
<tr>
<td>Clock</td>
<td>Realistic</td>
</tr>
<tr>
<td>LRU (Least Recently Used)</td>
<td>Excellent, but difficult to implement exactly</td>
</tr>
<tr>
<td>NFU (Not Frequently Used)</td>
<td>Fairly crude approximation to LRU</td>
</tr>
<tr>
<td>Aging</td>
<td>Efficient algorithm that approximates LRU well</td>
</tr>
<tr>
<td>Working set</td>
<td>Somewhat expensive to implement</td>
</tr>
<tr>
<td>WSClock</td>
<td>Good efficient algorithm</td>
</tr>
</tbody>
</table>

Modeling Page Replacement Algorithms

- The more page frames there are, the fewer pages faults a program will get?

Belady's Anomaly

Paging

- Suffers from internal fragmentation
- NO user view
- Page size fixed
- In paging, user specifies only a single address
- Single address is partitioned by HW into page no. and offset
Segmentation

- So far virtual addresses go from 0 to max
- Consider programs with many growing memory chunks
- The solution is to have more than one virtual addressing spaces, so that each may grow and shrink at will, called segments
- Each segment consist of linear addresses going from 0 to some max
- An address in segmented memory has two parts – segment number and address within the segment

One Dimensional Address Space

- One-dimensional address space with growing tables
- One table may bump into another

Advantage of Segmentation

- Allows data structures to grow independently
- Simplifies the linking process for compilation
  - Each procedure/function in separate segment
  - All separately compiled procedures start at segment address 0, respectively
  - A procedure call to a procedure in segment n will be linked as a two part address (n, 0)
  - If the same procedure is subsequently modified and recompiled, no other procedures need be changed, as their starting addresses have not be modified
- Shared libraries are shared by putting it in a segment shared by multiple processes

Implementation of Pure Segmentation

- Development of checkerboarding
- Removal of the checkerboarding by compaction
Segmentation with Paging: MULTICS

- Descriptor segment points to page tables
- Segment descriptor – numbers are field lengths

MULTICS Virtual Address

A 34-bit MULTICS virtual address

MULTICS Address Translation

Conversion of a 2-part MULTICS address into a main memory address

MULTICS TLB

Segmentation with Paging: Pentium

Pentium Segment Descriptors

- Pentium code segment descriptor
- Data segments differ slightly
Pentium: Linear Address

Conversion of a (selector, offset) pair to a linear address

Pentium: 2-level Paging

Mapping of a linear address onto a physical address

Segmentation and Paging

- Paging is transparent to the programmer
- Paging eliminates external fragmentation
- Pieces are to be moved in and out – paging helps!

- Segmentation is visible to the programmer
- Segmentation allows for growing data structures, modularity, and support for sharing and protection
- Each segment is broken into fixed-size pages

Segmentation and Paging

- Combine paging & segmentation -- best of both worlds
- What about overheads in terms of HW support?
- Is it worth it?

Protection

- As long as a program restricts itself to using segments at its own level, everything works fine.
- Attempts to access data at a higher level are permitted, at a lower level are illegal and causes traps.
- Attempts to call procedures at a different level (higher or lower) are allowed but controlled.
  - To make an inter-level call, the CALL instruction must contain a selector instead of an address.
  - This selector designates a descriptor called a callgate, which gives the address of the procedure to be called.