Before We Begin ...

Read Chapter 9 (on Paging and Segmentation)

Midterm grading

Programming assignment 2 will be available this weekend
  • due in 2 weeks, Sunday Nov 16 at midnight

Structure of Process Address Space

<table>
<thead>
<tr>
<th>Text: program instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>• execute-only, fixed size</td>
</tr>
</tbody>
</table>

Data: variables (global, heap)

<table>
<thead>
<tr>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>• read/write, variable size</td>
</tr>
<tr>
<td>• dynamic allocation by request</td>
</tr>
</tbody>
</table>

Stack: activation records

<table>
<thead>
<tr>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>• read/write, variable size</td>
</tr>
<tr>
<td>• automatic growth/shrinkage</td>
</tr>
</tbody>
</table>

Segmented Address Space

Address space is a set of segments

Segment: a linearly address memory
  • typically contains logically-related information
  • examples: program code, data, stack

Each segment has an identifier s, and a size n
  • s between 0 and S-1, S = number of segments

Logical addresses are of form (s, i)
  • offset i within segment s, i must be less than n
**Ex. of Segmented Address Space**

Segment 0
- Text
  - Base: 0, 0
  - Bound: 0, n
  - Permissions:-read, write

Segment 1
- Data
  - Base: 1, 0
  - Bound: 1, n
  - Permissions: read, write

Segment 2
- Stack
  - Base: 2, M
  - Bound: 2, M-n

Segment 3
- Shared Data
  - Base: 3, 0
  - Bound: 3, n
  - Permissions: read-only

\( n_s \) = size of segment \( s \)

**Address Translation for Segments**

Segment table contains, for each segment \( s \)
- base, bound, permissions, (+ valid bit)

Logical to physical address translation
- check if operation is permitted
- check if \( i < s.\text{bound} \)
- physical address = \( s.\text{base} + i \)

**Example of Address Translation**

32 bit logical address

10 bits segment \( s \)

22 bits offset \( i \)

- Segment Table
- Base Register
- Bound Register

\( \text{seg}[s].\text{base} + i \)

**Advantages of Segmentation**

Each segment can be
- located independently
- separately protected
- grow independently

Segments can be shared between processes
Problems with Segmentation

Variable allocation

Difficult to find holes in physical memory

Must use one of non-trivial placement algorithm
  • first fit, next fit, best fit, worst fit

External fragmentation

Paged Address Space

Address space is linear sequence of pages

Page
  • physical unit of information
  • fixed size

Physical memory is linear sequence of frames
  • a page fits exactly into a frame

Addressing

Each page is identified by a page number 0 to N-1
  • N = number of pages in address space
  • N * pagesize = size of address space

Logical addresses are of form (p, i)
  • offset i within page p
  • i is less than page size

Address Translation for Pages

Page table contains, for each page p
  • frame number that corresponds to p
  • other: perms, valid bit, reference bit, modified bit

Logical address (p, i) to physical address translation
  • check if operation is permitted
  • physical address = p.frame + i
**Example of Address Translation**

<table>
<thead>
<tr>
<th>32 bit logical address</th>
<th>22 bits</th>
<th>10 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>page p</td>
<td>offset i</td>
<td></td>
</tr>
</tbody>
</table>

Page Table Register

| 32 bit physical address | page[p].frame | i |

**Multi-Level Page Tables**

<table>
<thead>
<tr>
<th>32 bit logical address</th>
<th>12 bits</th>
<th>10 bits</th>
<th>10 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>page dir d</td>
<td>page p</td>
<td>offset i</td>
<td></td>
</tr>
</tbody>
</table>

| 32 bit physical address | dir[d]->page[p].frame | i |

**Segmentation vs. Paging**

Segment is good logical unit of information
- sharing, protection

Page is good physical unit of information
- simple memory management

Best of both
- segmentation on top of paging

**Combining Segmentation and Paging**

Logical memory is composed of segments
- each segment is composed of pages

Segment table
- per process, in memory pointed to by register
- entries map seg # to page table base
- shared segment: entry points to shared page table

Page tables (like before)
Address Translation

Logical address: segment #, page #, and offset

Index seg # into seg table: get base of page table

Check bounds (number of pages using page #)
  • may get a segmentation violation

Use page # to index into page table, get frame #

Concatenate frame # and offset to get physical address

Cost of Translation

Each page table lookup costs a memory reference
  • for each reference, additional references required
  • slows machine down by factor of 2 or more

Take advantage of locality of reference
  • most references are to a small number of pages
  • keep translations of these in high-speed memory

Problem: we don’t know which pages until referenced

Example of Address Translation

32 bit logical address

32 bit physical address

Translation Lookaside Buffer (TLB)

Fast associative memory keeps most recent translations

Determine whether non-offset part of LA is in TLB
  • yes: get corresponding frame num for phys addr
  • no: wait for normal memory translation (parallel)
Translation Cost with TLB

Cost is determined by
- speed of memory: ~ 100 nsec
- speed of TLB: ~ 20 nsec
- hit ratio: fraction of refs satisfied by TLB, ~95%

Speed with no address translation: 100 nsec

Speed with address translation
- TLB miss: 200 nsec (100% slowdown)
- TLB hit: 120 nsec (20% slowdown)
- average: $120 \times .95 + 200 \times .05 = 124$ nsec

TLB Design Issues

The larger the TLB
- the higher the hit rate
- the slower the response
- the greater the expense

TLB has a major effect on performance!
- must be flushed on context switches
- alternative: tagging entries with PIDs

MIPS: has only a TLB, no page tables!
- devote more chip space to TLB