Paging: Segmentation

Lecture 5: Mon 19.9.2011

Page table

Typical Page Table Entry, Fig. 3.11

- Each process has its own page table
- Each entry has a present bit, since not all pages need to be in the memory all the time -> page faults
- Remember the principle of locality
- Logical address space can be much larger than the physical
Page Tables

Internal operation of MMU with 16 4 KB pages

Swap area, VM backup store (Heittovaihtoalue)

Static
- Reserved in advance
- Space for whole process
  - Copy the code/text at start or
  - Reserve space, but swap gradually based on need
- PCB has swap location info

Dynamic
- Reserved space when needed
- Page table has swap block number for the page
  - Or use separate disk map to store page locations on swap
- No space reservation for pages that are never stored on swap

Swap location:
Windows: pagefile.sys, win386.swp
Linux: swap partition
Paging: Segmentation

If no segmentation

- One-dimensional address space with growing tables
- One table may bump into another
- Solution: separate tables to different segments
Segmentation

- Programmer (or compiler) determines the logical, unequal-sized segments
- Segment's size can change dynamically
- Segment length must be known
- Still using logical addresses (segment, offset)
- Segments can be freely located by OS
- OS maintains a segment table for each process

![Segmentation Diagram](image)

Segment table entry

- Each entry has a Present and Modified bits
- Segment location described using physical start address, segment base, and length
- Address translation

![Segment Table Entry](image)
Segmentation

- Segment size can be dynamically changed
  - Update the length in segment table entry
  - May require relocation of the segment in memory
- MMU must check address correction based on the current length value
- Segment allocation may cause external fragmentation
  - Memory compaction may be needed
- Segment is an excellent mechanism for protection and sharing
  - Logical entities make sense as elements for sharing
  - Programmer/User aware of segments, knows the content

Example of Pure Segmentation
- similar to dynamic partitioning

(a)-(d) Memory allocated and deallocated for segments
(e) Removal of the external fragments by compaction
Comparison of paging and segmentation

<table>
<thead>
<tr>
<th>Consideration</th>
<th>Paging</th>
<th>Segmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Need the programmer be aware that this technique is being used?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>How many linear address spaces are there?</td>
<td>1</td>
<td>Many</td>
</tr>
<tr>
<td>Can the total address space exceed the size of physical memory?</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Can procedures and data be distinguished and separately protected?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Can tables whose size fluctuates be accommodated easily?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Is sharing of procedures between users facilitated?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Why was this technique invented?</td>
<td>To get a large linear address space without having to buy more physical memory</td>
<td>To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection</td>
</tr>
</tbody>
</table>

Combining segmentation with paging
Segmentation with paging

- Segments split to pages
  - Memory management easier by pages
  - No external fragmentation (ei ulkoista pirstoutumista)
  - No compaction (ei tiivistämistarvetta)
- Process has
  - One segment table, contains sharing & protection info
  - One page table for each segment
- Adding new segment: new entry to segment table
- Segment grows: new entries to segment’s page table

Segmentation with Paging: MULTICS

- Each segment table element points to page table
- Segment table still contains segment length
Segmentation with Paging: MULTICS

- The virtual address has three parts
  - Segment number - index in segment table -> page table
  - Page number - index in page table -> page frame
  - Offset within the page

Address translation

Sta Fig 8.13
Pentium: Segmentation

- Pentium has two tables: LDT and GDT
  - At page access selector informs the CPU which one it is
  - Can contain 8K segment entry descriptors (13 bits)
- Local Descriptor Table (LDT) – one for each process
  - Process’s code, data, stack etc. segments
- Global Descriptor Table (GDT) - one, shared by all
  - System segments, also OS segments

During execution:
Segment registers
CS – code selector
DS – data selector
(segment selection)

Conversion of a (selector, offset) pair to a linear address
The linear address from the conversion is
- Physical address, if paging is disabled
  - Pure segmentation scheme in this case!
- Virtual address, if paging is enabled
  - Pages within the segment
Pentium: Paging

- Two-level page table
  - Each entry is 32 bit long, 20 bits for the frame number
  - Page directory and each page table has 1024 entries
  - Each page table fits to one 4 KB page

Pentium: protection (and segments)

- Protection on the Pentium based on levels
- Each process and segment has protection level
- Process on one level
  - Can access segments on the same and higher levels, but not on lower level
  - May call procedures on different level using selector (call gate) instead of address
UNIX / Solaris (+4BSD)
Memory management

Two-handed Clock

(Fig 8.23 [Stal05])

Fronthand:
set Reference = 0

Backhand:
if (Reference == 0)
  page out

Pages not used during the sweep are assumed not to be in the working sets of the processes.
Suits better for large memories.

Speed of the hands (frame/sec)? Scanrate
Gap between the hands?
Handspread
Linux
Memory management

Linux: paging

- Architecture independent
  - Alpha: 64b addresses, full support for 3 levels,
    - Page size 8KB, offset 13 bits
  - x86: 32b address, only 2-levels,
    - Page size 4KB, offset 12 bits
- 4-level page table
  - Page directory, 1 page
  - Page upper directory, multiple pages
  - page middle directory, multiple pages
  - page table, multiple pages
- Page directory always in memory
  - The page table address given to MMU at process switch
  - Other pages can be on disk

![Fig 10-16: Linux uses four-level page tables.](image)
**Linux: placement (allocation)**

- Reserves a consecutive region for consecutive pages
  - More efficient fetching and storing with disks
  - Optimize the disk utilisation, with the cost of memory allocation
- Buddy System: 1, 2, 4, 8, 16 or 32 page frames
  - Allocates pages in large groups, increases internal fragmentation
  - Implementation: table with list of different sizes unallocated page regions

**Buddy system**

- Allocation: if no proper-sized element available, split one larger area to two – repeat if needed
- Freeing: Combine two, earlier split, joint areas back to one larger area – repeat if needed
Windows
Memory management

Windows: Paging

- No segmentation
- Variable resident set size per process
  - Each process has minimum and maximum limits for resident set
  - Limits are not hard bounds
  - If large number of free frames, process can get more frames
    - A greedy one is not allowed to allocate last 512 frames
  - If the free memory reduces, the resident set of processes can be decreased
- Demand fetch and prepaging to standby list
Windows address space

[Tane01]

Fig. 11-24. Mapped regions with their shadow pages on disk. The \texttt{lib.dll} file is mapped into two address spaces at the same time.

Windows: page frames

(Fig 11.36 [Tane08])
http://lxr.linux.no/

- A linux cross referencer
  - links the same element from all files together

- Read the code yourself

- For example:
  - include/linux/mm_types.h
    contains struct page
  - linux/mm/slub.c
    slab allocator routines use page struct

Design issues omitted at Lecture 4 can be covered now