Paging: Segmentation

Lecture 5: Mon 19.9.2011

- 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 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

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 limits
- One table may bump into another
- Solution: separate tables to different segments

Page Tables

Internal operation of MMU with 16 4 KB pages

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

Paging: Segmentation

If no segmentation

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

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
Segmentation

- Programmer (or compiler) determines the logical, unequal-sized segments.
- Segments can be dynamically changed.
- Segment length must be known.
- Still using logical addresses (segment, offset).
- Segments can be freely located by OS.
- OS maintains segment table for each process.
- Each entry has a Present and Modified bit.
- 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.
- Memory allocated and deallocated for segments.
- Removal of the external fragments by compaction.

Comparison of paging and segmentation

Comparison

<table>
<thead>
<tr>
<th>Consideration</th>
<th>Paging</th>
<th>Segmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can the programmer be aware that the technique is being used?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>How many of these address spaces are there?</td>
<td>Many</td>
<td>One</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 allocated and separately accessed?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Can tiles whose size fluctuations be accommodated?</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 allow programs and data to be broken up into logically independent address space without loss of sharing and protection.</td>
<td>To allow programs and data to be broken up into logically independent address space without loss of 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 (si, alla data pistiutumista)
  - No compaction (si tisäntistä kärteä)
- 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

Memory management easier by pages

No external fragmentation

No compaction

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

- 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

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

Physical address, if paging is disabled

Virtual address, if paging is enabled

Pages within the segment

PCB

Segmentation with Paging: MULTICS

Segment table

Segment table entry

- 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

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

Physical address, if paging is disabled

Virtual address, if paging is enabled

Pages within the segment

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

Pentium: Segmentation

During execution:

Segment registers

CS - code selector

DS - data selector

(segment selection)

Pentium: Segmentation with Paging

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
  - 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

Fig 8.23 [Stal05]

Linux

Memory management

- Architecture independent
  - A thin 64-bit addresses, full support for 3 levels
  - Page-size 4 KB, offset 13 bits
  - x86: 32-bit address, only 2-levels
  - Page-size 4 KB, offset 13 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: 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: 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 prepagin to standby list

Windows: page frames
- Page fault is handled by demand paging
- Processes use page frames from disk
- Page frames are managed by the operating system

Windows address space
- Mapped regions with their shadow pages on disk. The tmpfile file is mapped into two address spaces at the same time.
http://lxr.linux.no/

- A Linux cross reference
  - 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