English Edition

Memory Management (4)

Segmentation with Paging (1)

- Partition programs into segments, which are placed in the physical memory using Paging (non-contiguous).
- Programmer uses segment numbers and relative addresses within each segment. Paging is handled by the hardware – transparent for the program.

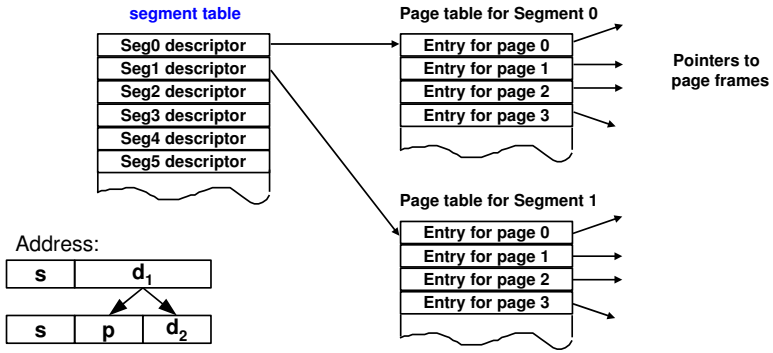
Segmentation with Paging (2)

Two ways of implementing this:

- One segment table and for each segment (per process) one page table.
  Entries in segment table point to the page table of the segment (e.g. Unix).
- One segment table and one single page table (per process) (e.g. Intel CPUs).
  - Entries in the segment table contain the base address of the segment in a linear (virtual) address space.
  - Add the relative address within the segment to this base address. The resulting linear (virtual) address is translated into a physical address by using the page table.

Segmentation with Paging (3)

- One page table per segment:



Segmentation with Paging (4)

- Addresses always consist of
  - a segment number and
  - a linear address within the segment.

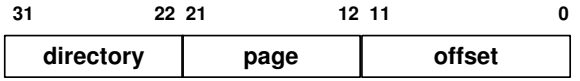
  Splitting of the linear address into page number and offset (for paging) is transparent for the program.

Segmentation with Paging: Intel 80386

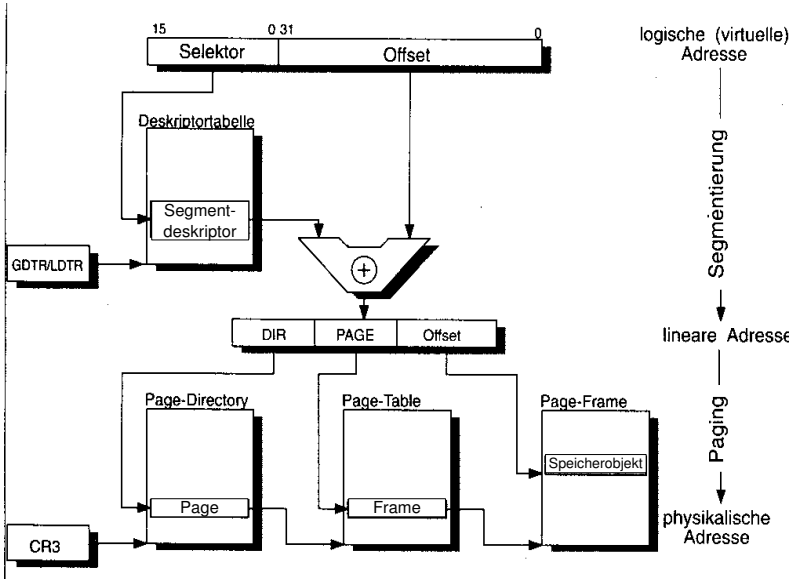
- Segments:
  - Up to 8 K process-private segments, described by entries in the local descriptor table (LDT).
  - Up to 8 K shared segments (used by all processes), described by entries in the global descriptor table (GDT).
  - Each segment can be up to 4 GByte in size.
- Six segment registers, at each time a process can use up to six segments.

Segmentation with Paging: Intel 80386

- Six internal registers which contain the corresponding segment descriptors.
- An address is a tupel (segment selector, offset).
- The segment descriptor is the base address of the segment in a 32 bit address space plus the offset, delivering the so-called linear address.
- The linear address looks like this (two-level paging):

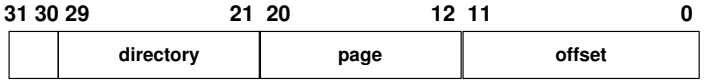


Address translation with Intel 80386



Physical Address Extension (PAE) of Intel Pentium Pro (1)

- PAE mode: Processor uses three-layer Paging



- The two highest bits contain the "Page Directory Pointer Index", which chooses one of four Page Directories.
- Entries in the Page Directories and in the Page Tables are 8 bytes long. (Since there are 512 entries in every one, these tables have a size of 4 KByte, i.e. one page.)

Physical Address Extension (PAE) of Intel Pentium Pro (2)

- Page frame numbers are 24 bits long. Thus physical addresses have 36 bit and the addressable physical memory size is 64 GByte.
- The Intel 450NX chipset lets x86 CPUs use PAE.

PAE and Linux 2.6 (1)

- Linux supports (on Intel 32 bit platforms) up to 64 GByte main memory
- Choice at kernal configuration time:
  - up to 1 GByte,
  - >1 up to 4 GByte,
  - >4 up to 64 GByte

PAE and Linux 2.6 (2)

arch/i386/Kconfig:

```
choice
        prompt "High Memory Support"
        default NOHIGHMEM

config NOHIGHMEM
        bool "off"

config HIGHMEM4G
        bool "4GB"
        help
          Select this if you have a 32-bit processor and between 1
          and 4 gigabytes of physical RAM.

config HIGHMEM64G
        bool "64GB"
        depends on X86_CMPXCHG64
        help
          Select this if you have a 32-bit processor and more than 4
          gigabytes of physical RAM.
```

PAE and Linux 2.6 (3)

- Memory partitioned:
  ZONE_DMA: First 16 MByte
  for old ISA devices
  ZONE_NORMAL: 16 MByte – 896 MByte
  Standard (kernel and user mode)
  ZONE_HIGHMEM: 896 MByte – Rest
- Kernel has 1 GByte of virtual memory, of which the first 896 MByte (ZONE_DMA und ZONE_NORMAL) are statically mapped.
- 128 MByte remain, which can be dynamically mapped to access memory > 1 GByte.

PAE and Windows 2000 (1)

- In order to use PAE, the operating system must be modified.
- For Windows 2000, Microsoft has developped a special PAE kernel (ntkrnlpa.exe). If the CPU is PAE-capable and the system has more than 4 GByte memory, this kernel is loaded.

PAE and Windows 2000 (2)

- The Windows PAE kernel supports the following physical memory sizes:
  - W2K Professional: 4 GByte
  - W2K Server: 4 GByte
  - W2K Advanced Server: 8 GByte
  - W2K Datacenter 64 GByte
- Microsoft offers „Address Windowing Extensions (AWE) API“ which let applications allocate physical memory for their exclusive usage and let them map parts of their address spaces to this memory.

Demand Paging (1)

- The address space of a process need not reside completely in main memory.
  - Locality principle says that a process will only access few, close (local) addresses in any given time span.
  - Parts of the program code possibly never execute during regular processing (e.g. special cases; error handling routines).

Demand Paging (2)

- Demand Paging means that
  - a page is only loaded into memory when a process accesses it,
  - a page can me removed from memory.
- Advantages of Demand Paging:
  - Address space of a process can be larger than the total physical memory.
  - Processes need less space in main memory, thus more processes can be (in memory and) ready at the same time.

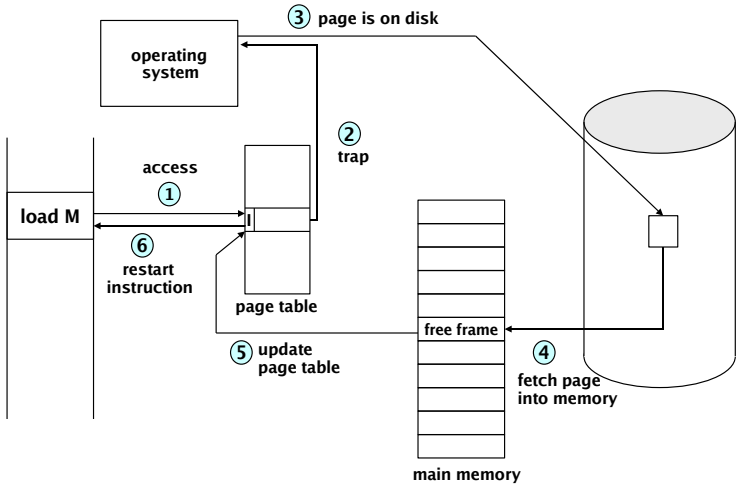
Requirements for Demand Paging (1)

- Each entry in the page table has a valid bit that tells whether the page is in memory or not.
- When a process accesses a page that is not in memory, this causes a special exception, the so-called page fault.
- An operating system routine, the page fault handler, loads the requested page from disk.

Requirements for Demand Paging (2)

- If there is no free page frame in the main memory, some other page has to be replaced. For selecting the page that is to be replaced, a replacement strategy must be implemented.
- The instruction that was interrupted by the page fault, must be re-executed (and it must be possible to do that).

Page Fault Handling



Page Replacement (1)

- When a Page Fault occurs and there is no free page frame available, the operating system must free one.
- An algorithm chooses this page frame according to a specific strategy.

Page Replacement (2)

- If the page to be replaced has been changed since it was last fetched from the disk, its content must be saved:
  - A modify bit (or dirty bit) in the page table entry tells whether the page has been modified.
  - A modified page will be written to disk (to the page or swap area).

Page Replacement (3)

- An unmodified page can later (if needed) be loaded from the old, known location on disk.
- In the page table entry for the replaced page
  - the valid bit will be deleted,
  - information about the location, from where the page can be retrieved, will be saved.

Page Replacement Strategies (1)

- Goal: As few Page Faults as possible.
- Two principal types of page replacement strategy:
  - local replacement
  - global replacement

Page Replacement Strategies (2)

local replacement:

Always replace a page that belongs to the process which requests a new page.

- The number of pages which can be allocated by a process has an upper bound. The maximum number is assigned per process (e.g. by the system administrator) and it can strongly influence runtime behavior of a process.
- A process which causes many Page Faults (e.g. because it does not conform to the locality principle) affects only itself, not the whole system.

Page Replacement Strategies (3)

global replacement:

Replace an arbitrary page in memory.

- processes take away each other's page frames.
- a process that causes many page faults automatically receives more page frames. (This can be an advantage or disadvantage.)

Optimal Strategy

Replace the page which will not be accessed for the longest time (in the future).

- Advantage: This strategy causes the smallest number of page faults.
- Disadvantage: This strategy is not implementable.

However, the optimal strategy can serve as a model in order to evaluate other strategies.

First In First Out (FIFO)

Replace the page which has been in memory for the longest time.

- Advantage: very easy to implement:
  - Manage a chained list of all pages in memory (global strategy) respectively of a process' pages (local strategy).
  - When a page fault occurs, replace the first page in the list and append the new page to the end of the list.
- Disadvantage: The replaced page can be in constant use and might be re-requested immediately.

Least Recently Used (LRU) (1)

Replace the page, that has not been used for the longest time.

- Advantage: Typically less Page Faults than with FIFO strategy.
- Nachteil: complex implementation.

two approaches for implementation:

- with counter
- with chained list

Least Recently Used (LRU) (2)

- Implementation with counter:
  - system-wide counter that is incremented at each memory access.
  - write recent value of counter into a field in the data structure which describes the accessed page.
  - replace page with lowest value.

Least Recently Used (LRU) (3)

- Implementation with chained list:
  - a chained list contains all pages.
  - at each memory access: bring the accessed page to the head of the list.
    (walk through list and change order, i.e. change pointers)
  - replace page at the end of the list.

Using a Reference Bit (1)

- each page table entry can contains a reference bit
  - that is set at accessing this page (by the hardware),
  - that will be unset according to specific criteria (by the software).
- a reference bit
  - holds the information, whether a page was accessed since last unsetting of the bit,
  - says nothing about when the page was accessed,
  - says nothing about the order of accesses of several pages.

Using a Reference Bit (2)

- reference bits let you implement further page replacement strategies, e.g.
  - modifications of LRU which are less complex.
  - Second Chance Algorithm, an improved version of the FIFO strategy.

Modification of LRU (1)

- create a binary counter for each page.
- in regular intervals:
  - shift each counter one position to the right ("aging"),
  - copy the reference bit into the highest bit of the counter,
  - unset the reference bit.

Modification of LRU (2)

- replace one of the pages having the minimal counter value.
  - If several pages have the same counter value, it is unknown which one was accessed last.
  - Accesses which happened longer ago, will first be weighted less and finally be "forgotten" (shifted out of the counter).