More on Paging Virtualization

Knut Omang
Ifi/Oracle
14 Oct, 2010

(with slides from V. Goebel, C. Griwodz (Ifi/UiO), P. Halvorsen (Ifi/UiO), K. Li (Princeton), A. Tanenbaum (VU Amsterdam), and M. van Steen (VU Amsterdam))

Today

- More on page replacement algorithms
- Design issues for paging systems
- Segmentation
- Addressing on x86
- Virtualization

Implementing LRU

	0	1	2	3
0	0	0	0	1
1	1	0	0	1
2	1	1	0	0
3	0	0	0	0

Page ref: 2

Least recently used: Read binary value across
- Smallest value = LRU

Implementing LRU

Page

	0	1	2	3
0	0	1	1	1
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

(a)

0	0	1	1
1	0	1	1
0	0	0	0
0	0	0	0

(b)

0	0	0	1
1	0	0	1
1	1	0	1
0	0	0	0

(c)

0	0	0	0
1	0	0	0
1	1	0	0
1	1	1	0

(d)

0	0	0	0
1	0	0	0
1	1	0	1
1	1	0	0

(e)

0	0	0	0
1	0	0	0
1	1	0	1
1	1	0	0

(f)

0	1	1	1
0	0	1	1
0	0	0	1
0	0	0	0

(g)

0	1	1	0
0	0	1	0
0	0	0	0
1	1	1	0

(h)

0	1	0	0
0	0	0	0
1	1	0	1
1	1	0	0

(i)

0	1	0	0
0	0	0	0
1	1	0	0
1	1	1	0

(j)

LRU using a matrix – pages referenced in order 0,1,2,3,2,1,0,3,2,3

Implementing LRU

- Problem: Requires special hardware
- Lots of bits to update

- Software approximation?
  - NFU (not frequently used)
    - Scan R (referenced) bit of page table every clock interrupt
  - "aging" by bit shift

Least Recently Used (LRU)

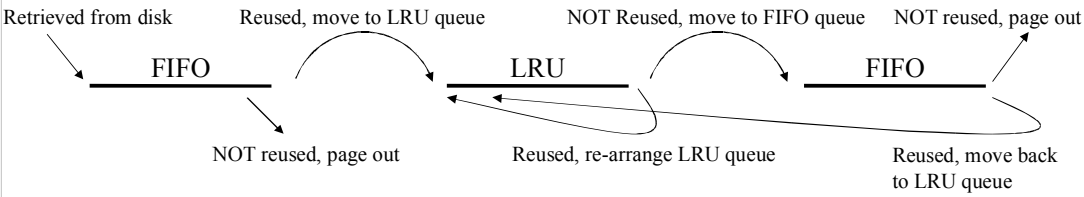
Simulating LRU by using aging:

- "reference counter" for each page
- after a clock tick:
  - shift bits in the reference counter to the right (rightmost bit is deleted)
  - add a page's referece bit in front of the reference counter (left)
- page with lowest counter is replaced

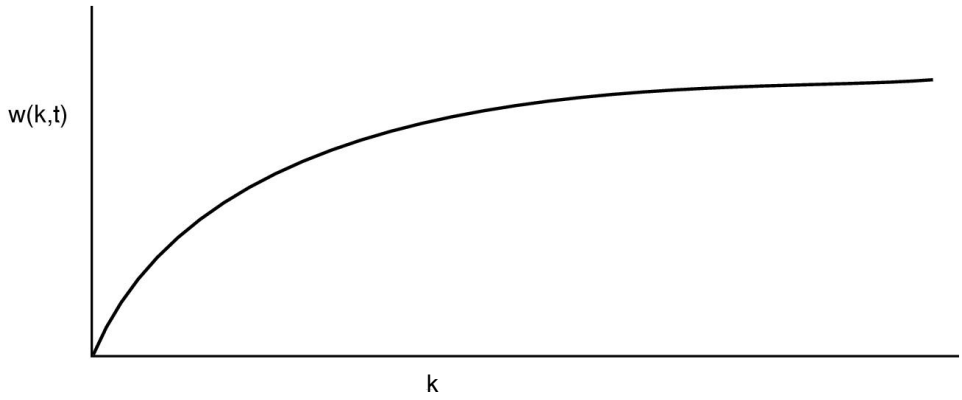
	Clock tick 0 1 0 1 0 1 1	Clock tick 1 1 1 0 1 0 0	Clock tick 2 1 1 0 1 0 1	Clock tick 3 1 0 0 0 1 0	Clock tick 4 0 1 1 0 0 0
1	00000000	10000000	11000000	11100000	01111000
2	00000000	00000000	10000000	01100000	10110000
3	00000000	10000000	01000000	00010000	10001000
4	00000000	00000000	10000000	01100000	00110000
5	00000000	10000000	01000000	10010000	01001000
6	00000000	10000000	01000000	01010000	00101000

LRU-K & 2Q

- **LRU-K:** bases page replacement in the *last K references* on a page [O'Neil et al. 93]
- **2Q:** uses 3 queues to hold much referenced and popular pages in memory [Johnson et al. 94]
  - 2 FIFO queues for seldom referenced pages
  - 1 LRU queue for much referenced pages



The Working Set Page Replacement Algorithm



- The working set is the set of pages used by the *k* most recent memory references
- w(k,t) is the size of the working set at time, *t*

Working Set Model

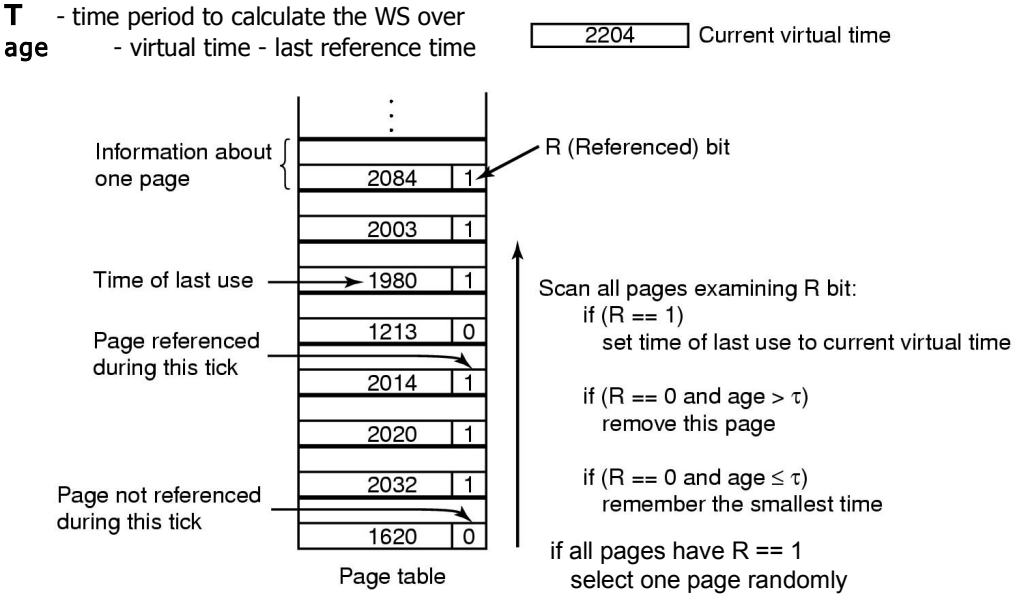
- Working set:
  set of pages which a process is currently using

- Working set model:
  paging system tries to keep track of each process' working set and makes sure that these pages is in memory before letting the process run
  → reduces page fault rate (prepaging)

- Defining the working set:
  - set of pages used in the last *k* memory references (must count backwards)
  - approximation is to use all references used in the last XX instructions

Working Set Page Replacement Algorithm

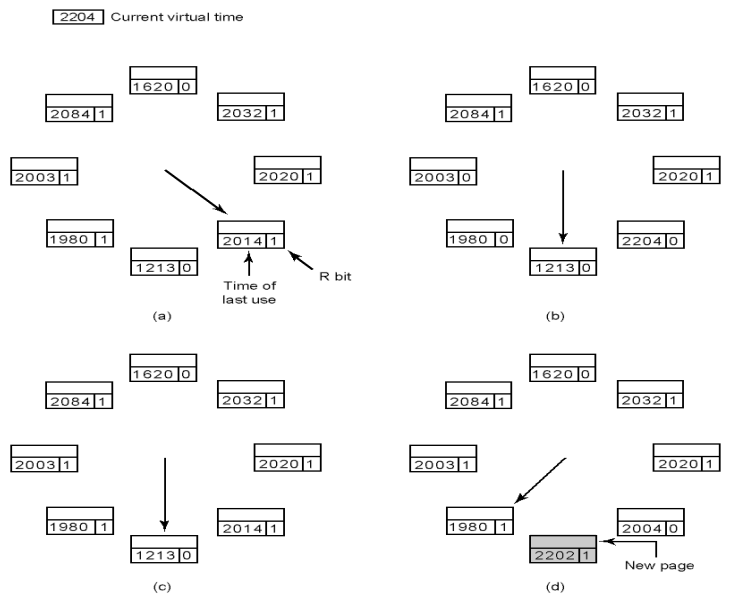
T - time period to calculate the WS over
age - virtual time - last reference time



Expensive - must search the whole page table

The WSClock Page Replacement Algorithm

- Working set algorithm expensive
  - Must scan whole table for each fault
- WSClock
  - Has simple implementation
  - good performance
  - good approximation
- Principle: Virtual timestamp + walk through circular list of used pages
  - If R = 0, evict -> done
  - If R = 1, R := 0 (consider scheduling flush)



Review of Page Replacement Algorithms

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

Locality and paging

- Reference locality:
  - Time:
    pages that are referenced in the last few instructions will probably be referenced again in the next few instructions
  - Space:
    pages that are located close to the page being referenced will probably also be referenced

Demand Paging Versus Prepaging

- Demand paging:
  pages are loaded on demand, i.e., after a process needs it
  - Should be used if we have no knowledge about future references
  - Each page is loaded separatly from disk, i.e., results in many disk accesses
- Prepaging:
  prefetching data in advance, i.e., before use
  - Should be used if we have knowledge about future references
  - # page faults is reduced, i.e., page in memory when needed by a process
  - # disk accesses can be reduced by loading several pages in one I/O-operation

Allocation Policies

- How should memory be allocated among the competing runnable processes?
- Equal allocation:
  all processes get the same amount of pages
- Proportional allocation:
  amount of pages is depending on process size

Allocation Policies

- Local page replacement:
  consider only pages of own process when replacing a page
  - corresponds to equal allocation
  - can cause thrashing
  - multiple, identical pages in memory

- Global page replacemet:
  consider all pages in memory when replacing a page
  - corresponds to proportional allocation
  - better performance in general
  - monitoring of working set size and aging bits
  - data sharing

Design Issues for Paging Systems
Local versus Global Allocation Policies (1)

	Age
A0	10
A1	7
A2	5
A3	4
A4	6
A5	3
B0	9
B1	4
B2	6
B3	2
B4	5
B5	6
B6	12
C1	3
C2	5
C3	6

(a)

A0
A1
A2
A3
A4
A6
B0
B1
B2
B3
B4
B5
B6
C1
C2
C3

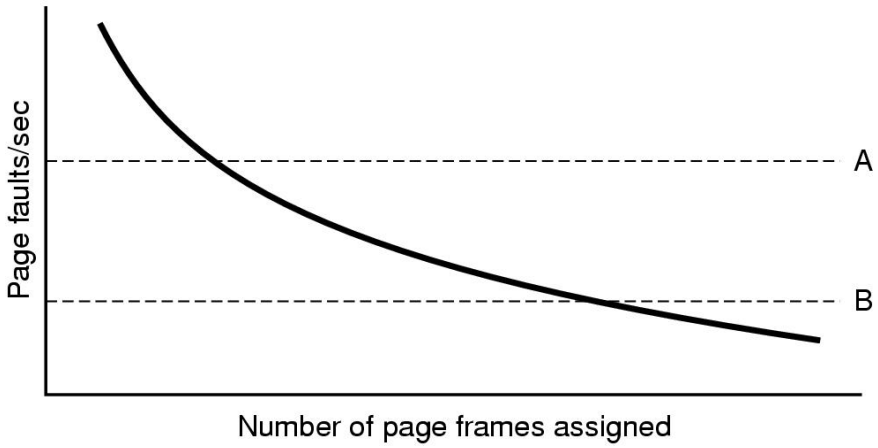
(b)

A0
A1
A2
A3
A4
A5
B0
B1
B2
A6
B4
B5
B6
C1
C2
C3

(c)

- Original configuration
- Local page replacement
- Global page replacement

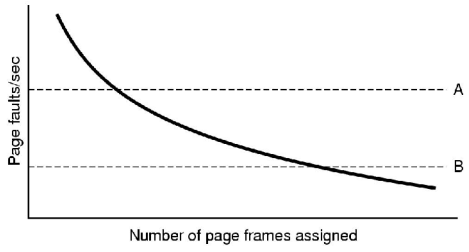
Local versus Global Allocation Policies (2)



- Page fault rate as a function of the number of page frames assigned to a process

PFF – Page fault frequency algorithm

- Measure page fault rate
- Allocate or reclaim pages to keep page faults between dotted lines A and B



Load Control

- Despite good designs, system may still thrash
  - When PFF (page fault frequency) algorithm indicates
    - some processes need more memory
    - but no processes need less
- Solution :
  Reduce number of processes competing for memory
  - swap one or more to disk, divide up pages they held
  - reconsider degree of multiprogramming

Page Size (1)

Small page size

- Advantages
  - less internal fragmentation
  - better fit for various data structures, code sections
  - less unused program in memory
- Disadvantages
  - programs need many pages, larger page tables

Page Size (2)

- Overhead due to page table and internal fragmentation

$$overhead = \frac{s \cdot e}{p} + \frac{p}{2}$$

page table space

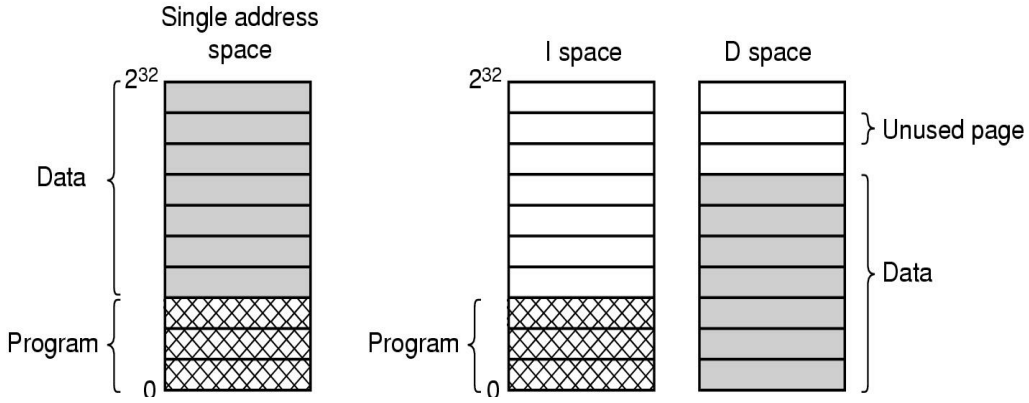
internal fragmentation

- Where
  - s = average process size in bytes
  - p = page size in bytes
  - e = page table entry

Optimized when

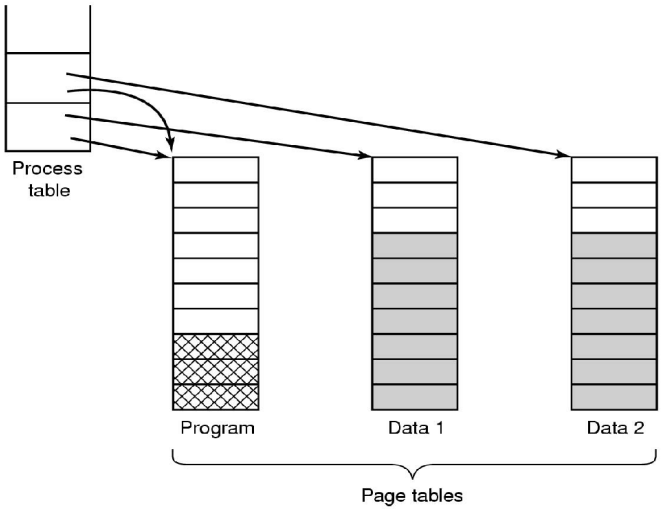
$$p = \sqrt{2se}$$

Separate Instruction and Data Spaces



- One address space
- Separate I and D spaces

Shared Pages



Two processes sharing same program sharing its page table

Cleaning Policy

- Need for a background process, paging daemon
  - periodically inspects state of memory
- When too few frames are free
  - selects pages to evict using a replacement algorithm
- It can use same circular list (clock)
  - as regular page replacement algorithm but with diff ptr

Implementation Issues

Operating System Involvement with Paging

1. Process creation
   - determine program size
   - create page table
1. Process execution
   - MMU reset for new process
   - TLB flushed
1. Page fault time
   - determine virtual address causing fault
   - swap target page out, needed page in
1. Process termination time
   - release page table, pages

Page Fault Handling (1)

1. Hardware traps to kernel
2. General registers saved
3. OS determines which virtual page needed
4. OS checks validity of address, seeks page frame
5. If selected frame is dirty, write it to disk

Page Fault Handling (2)

1. OS brings schedules new page in from disk
2. Page tables updated
   - Faulting instruction backed up (undone)
   - Faulting process scheduled
1. Registers restored
2. Program continues

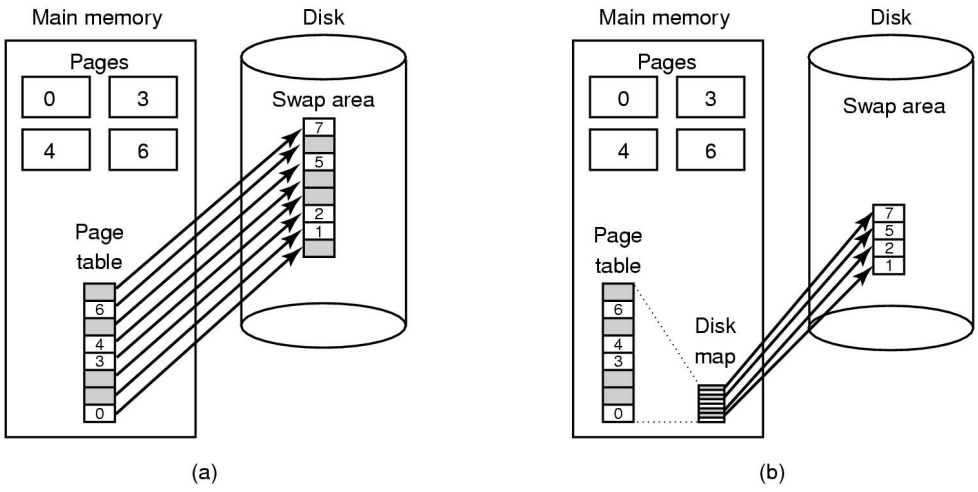
Paging Daemons

- Paging daemons:
  Background process which sleeps most of the time, but is for example awakened periodically or when the CPU is idle
  - Taking care that enough free page frames are available by writing back modified pages before they are reused
  - Prepaging

Locking Pages in Memory

- Virtual memory and I/O occasionally interact
- Proc issues call for read from device into buffer
  - DMA (Direct Memory Access -from device)
  - while waiting for I/O, another processes starts up
  - has a page fault
  - buffer for the first proc may be chosen to be paged out
- Need to specify some pages locked
  - exempted from being target pages

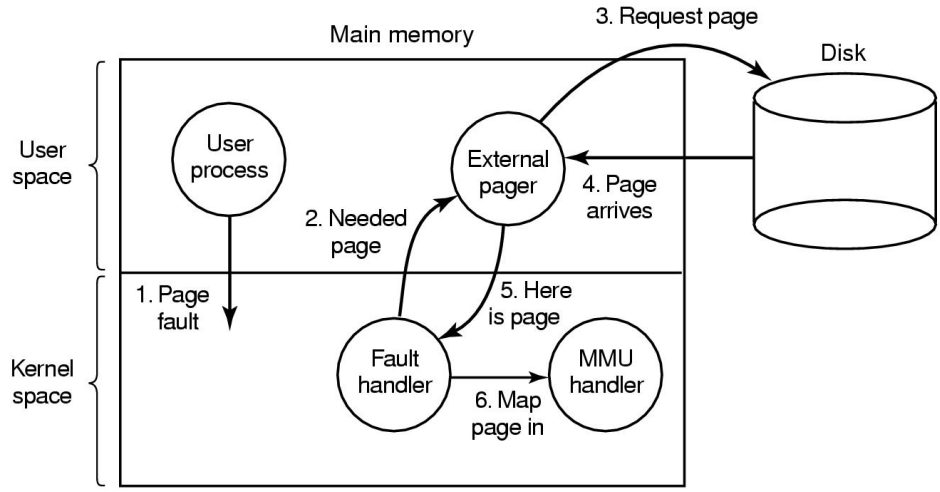
Backing Store



(a) Paging to static swap area

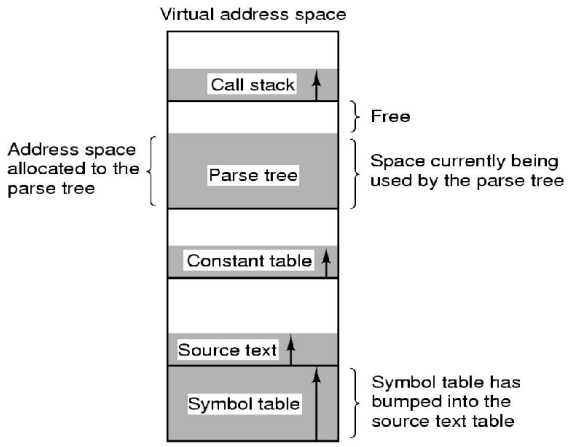
(b) Backing up pages dynamically

Separation of Policy and Mechanism



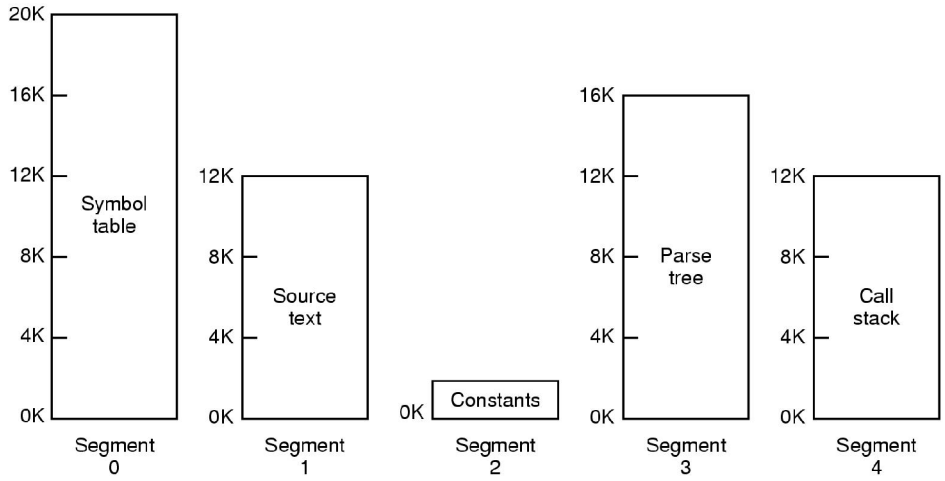
Page fault handling with an external pager

Segmentation (1)



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

Segmentation (2)



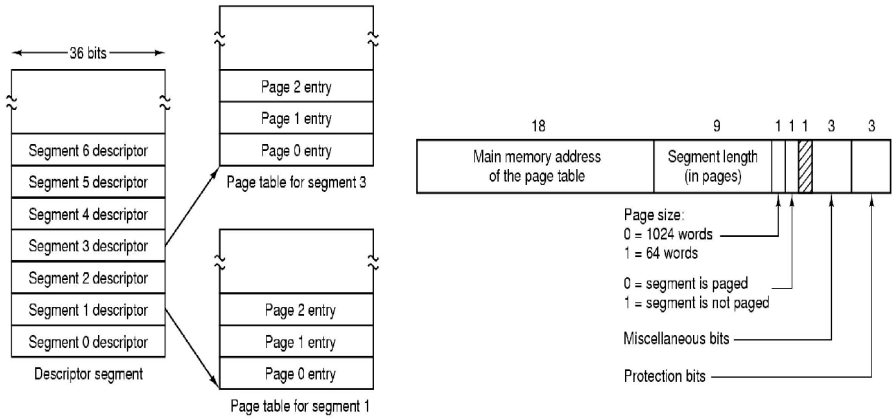
Allows each table to grow or shrink, independently

Segmentation (3)

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

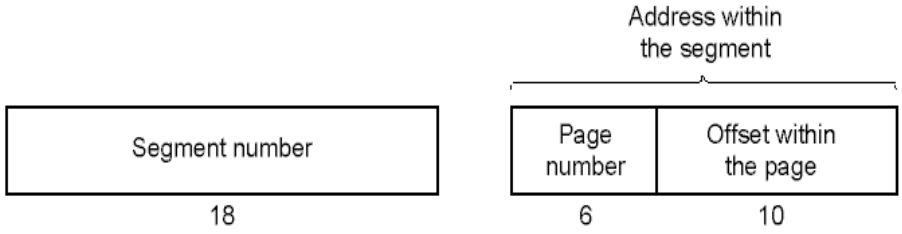
Comparison of paging and segmentation

Segmentation with Paging: MULTICS (1)



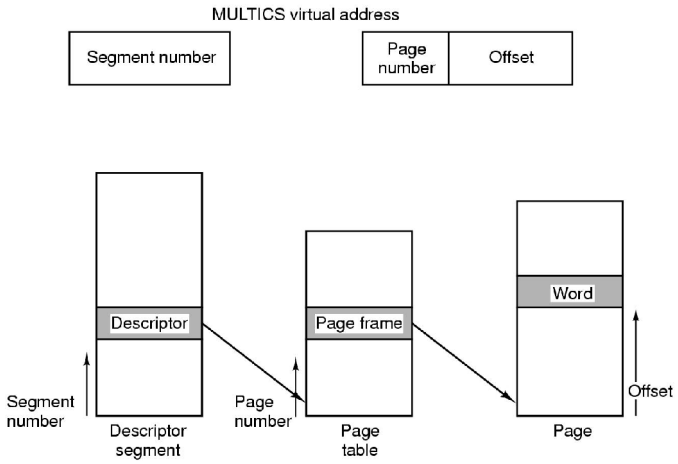
- Descriptor segment points to page tables
- Virtual address = seg# + page# + offset

Segmentation with Paging: MULTICS (2)



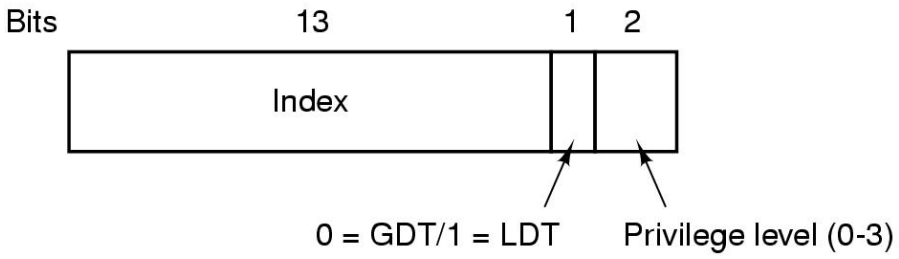
A 34-bit MULTICS virtual address

Segmentation with Paging: MULTICS (3)

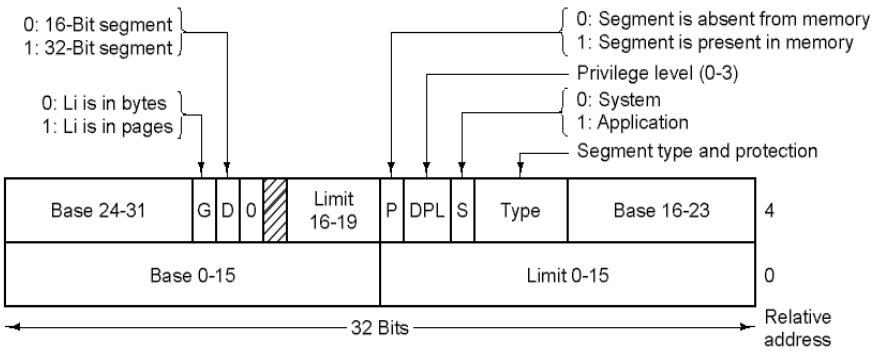


Conversion of a 2-part MULTICS address into a main memory address

Segmentation with Paging: Pentium (1)



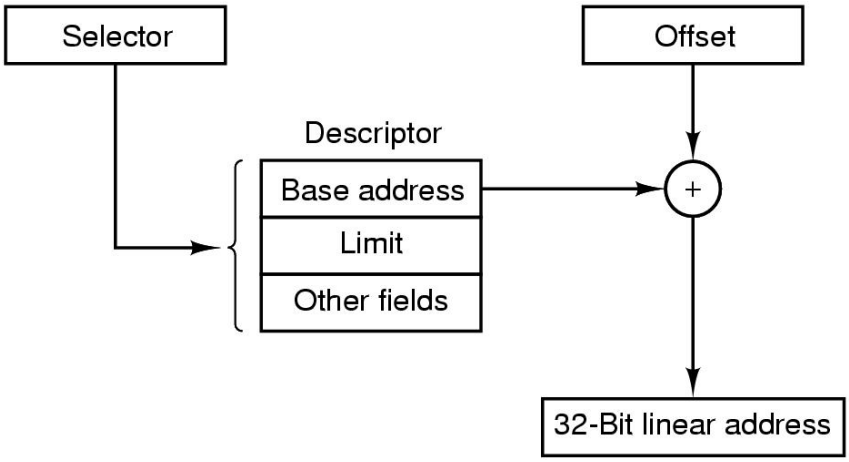
- A Pentium segment selector
  - CS register stores code segment selector
  - DS register stores data segment selector



Segmentation with Paging: Pentium (3)

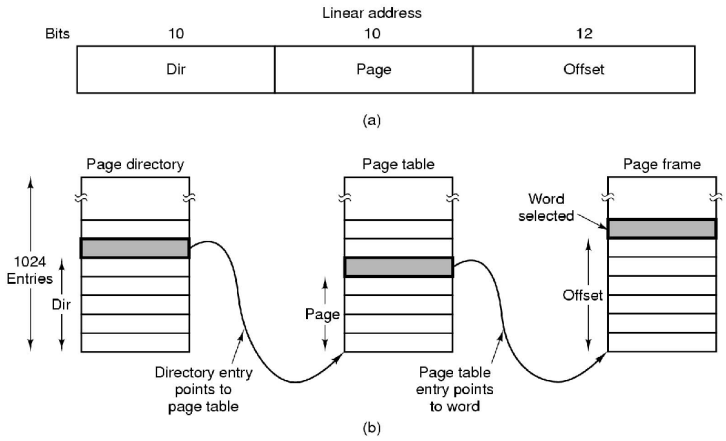
- Two tables
  - LDT (Local Descriptor Table)
  - GDT (Global Descriptor Table)
- Each segment up to 1G 32bit words
  - maps to 32 bit 'linear' address
- Linear address used to lookup in MMU

Segmentation with Paging: Pentium (4)



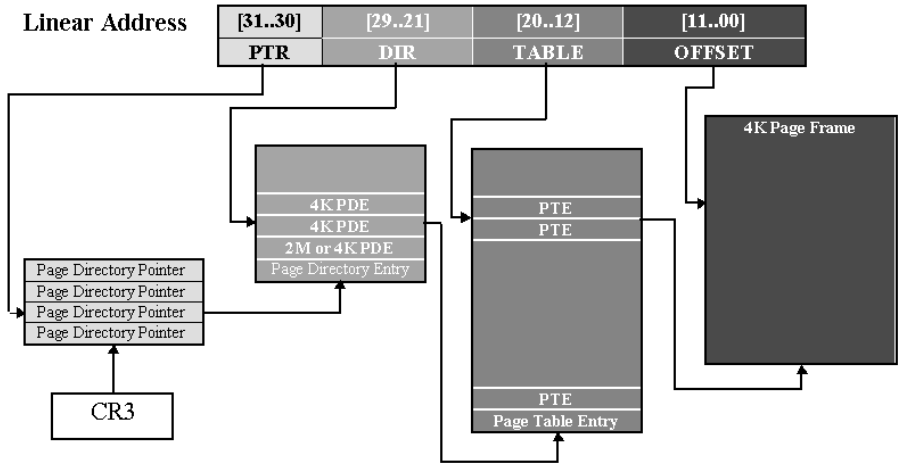
Conversion of a (selector, offset) pair to a linear address

Segmentation with Paging: Pentium (5)

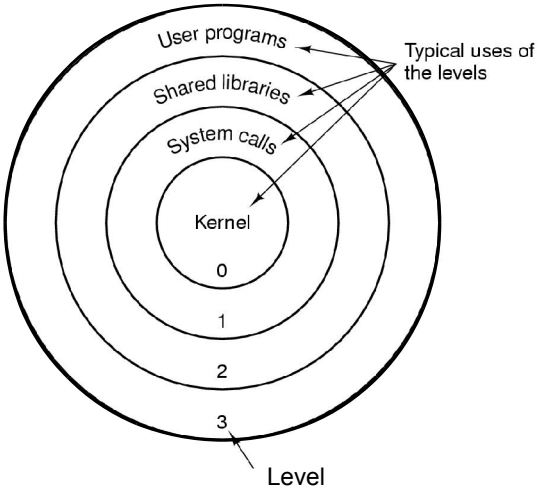


- Mapping of a linear address onto a physical address
- Dir: pointer to the right page table
- Each page table: 1024 entries of 4K = 4M addr.space

PAE (Physical Address Extension)



Protection on the Pentium



Virtualization

- Present a machine abstraction to *guest* operating systems:
  - Host operating system (often called hypervisor) sees whole computer
  - Guest operating system sees only a partition of the real computer
  - Adds another layer of protection
    - OS fault only affects part of the system
    - What about hardware fault? ...
  - Flexibility wrt use of resources
    - Imagine 100 services each 99% idle but requiring a separate computer (Why?...)

Virtualization -> isolation!

- Popek and Goldberg,1974:
  - *Sensitive instructions*: Instructions that for protection reasons must be executed in kernel mode
  - *Privileged instructions*: Instructions that causes a trap
  - A machine is *virtualizable* iff the set of sensitive instructions is a subset of the set of privileged instructions.

Virtualization before ca.1995

- IBM CP/CMS -> VM/370, 1979
  - Hardware support: Traps sensitive instructions
  - Architecture still in use for IBM "mainframes"
- Largely ignored by others
  - Taken up by Sun and HP around in 1990's
  - x86-world? Difficult because
    - Some sensitive instructions are ignored in user mode!
    - Some sensitive instructions are allowed from user mode!

Virtualization in the (limited) x86

- Solutions
  - Interpretation (emulating the instruction set)
    - Performance penalty of factor 5-10
    - Benefit: May emulate any type of CPU
  - "Full" virtualization
    - Privileged instructions in guest OS'es rewritten by virtualization software (binary translation)
    - Stanford DISCO --> VmWare workstation
    - Does not require source code of OS..
  - Paravirtualization
    - Replacing parts of the guest operating system with custom code for virtualization

Virtualization in the (limited) x86

- Problems:
  - Performance
    - I/O
    - Page faults
    - Interrupts (when?)
    - Virtual Machine perf
    - Host resource usage
  - Avoidig 'leaking' instructions
    - Pentium allows instruction that makes it possible to determine if it is executed in kernel mode
    - Might confuse OS..

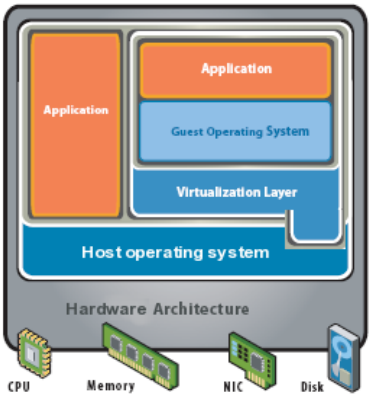
x86 virtualization in Xen (Paravirtualization)

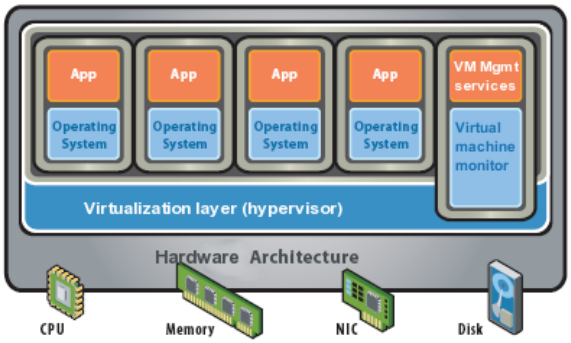
- Uses x86 privilege levels differently:
  - Rings: 0, 1, 2, 3 (highest to lowest privilege)
  - Normally OS executes in ring 0 and applications execute in ring 3
  - With Xen
    - 0 – Hypervisor
    - 1 – Guest OS
    - 2 – unused
    - 3 – Applications
- Guest OS modified for privileged instructions
- VMWare ESX: similar approach

Virtualization models

Special OS as hypervisor

or extensions to "full" OS





Virtualization terms

- Type 1 hypervisors:
  - Based solely on traps
    - requires sensitive ⊆ privileged)
- Type 2 hypervisors:
  - Based on some amount of binary translation on the fly
- Both runs unmodified OS'es

Virtualization with VT/SVM

- VT(Intel) and SVM(AMD):
  - Inspired by VM/370
  - Set of operations that trap controlled by bitmap managed by host OS/hypervisor
  - Present in most(all?) newer 64 bit versions of AMD/Intel processors
  - Allows type 1 hypervisors
  - Effectively privileged mode, guest privileged mode and user mode..
  - A lot of open source activity around this:
    - Qemu/KVM, VirtualBox, Xen,...

Memory virtualization

- Problem: Naive implementation would cause contention for physical pages!
  - Requires shadow page tables for guests, second layer of indirection:
    - Host physical addresses
    - Guest physical addresses
    - Guest virtual addresses
- Solution:Multi-level page tables
  - Available in newer CPUs

I/O Virtualization

- Virtual I/O devices:
  - Each OS expects it's own disk controllers, USB ports, keyboards, network devices...
  - DMA?
- Paravirtualization
  - Typical: simple devices emulated
    - IDE disk drive, simple PCI bus, simple USB device, old and simple network card
  - I/O Rings (Xen/KVM): generic support library for emulation
  - Emulation causes performance issues
- Can dedicate devices to Vms
- High end devices with hardware support:
  - Multiple logical devices in single physical
  - PCI Express extensions for virtualizaton