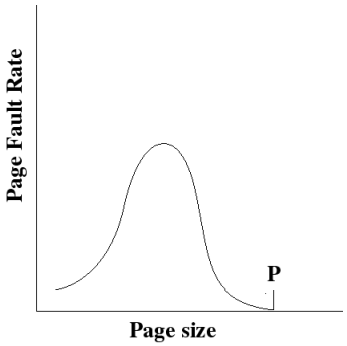
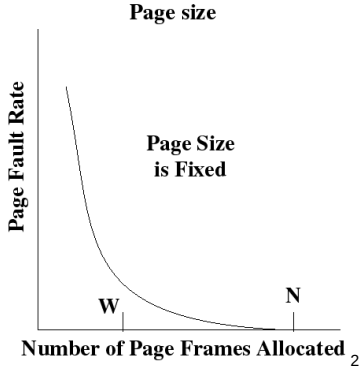
Paging: design and implementation issues

1

Effect of page size

- More small pages to the same memory space
- References from large pages more probable to go to a page not yet in memory
- References from small pages often to other pages -> often used pages selected to the memory
- Too small working set size -> larger page fault rate
- Large working set size -> nearly all pages in the memory





2

Page Size – Design Issues

- Reduce internal fragmentation ⟶ small
- Reduce page table size ⟶ large
- Proportional (1x, 2x, …) to disk block size
- Optimal value is different for different programs
- Increase TLB hit ratio ⟶ large
- Different page size for different applications?
- How does MMU know the page size?

Tbl 8.2 [Stal05]

3

Table 8.2 Example Page Sizes

Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1024 36-bit word
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 kbyes to 16 Mbytes
UltraSPARC	8 Kbytes to 4 Mbytes
Pentium	4 Kbytes or 4 Mbytes
PowerPc	4 Kbytes
Itanium	4 Kbytes to 256 Mbytes

4

Page Size

- Overhead due to page table and internal fragmentation

page table space

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

internal fragmentation

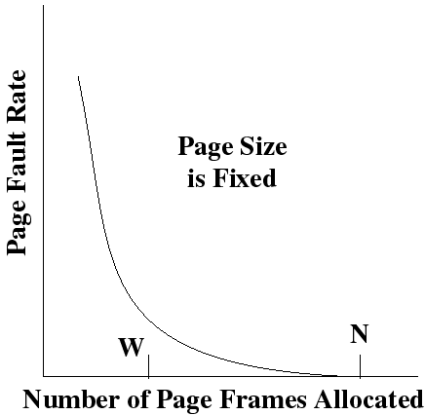
- Where
  - s = average process size in bytes
  - p = page size in bytes
  - e = page entry

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

s = 1 MB e = 8B → p_{opt} = 4 KB
s = 100 MB e = 8B → p_{opt} = 40 KB

Working set

- Set of pages used during last k references (window size)
- Pages not in the working set are candidates to be released or replaced
- Suitable size for the working set? Estimate based on page fault rate



$$1 \le |W(t, \Delta)| \le \min(\Delta, n)$$

6

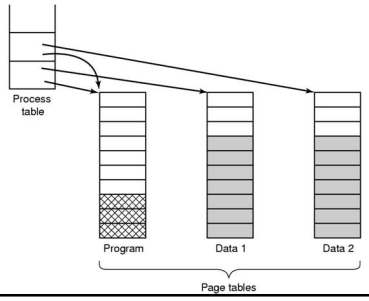
Sharing libraries

- Old style: Link the library with program statically
- New style: Link the library at run time dynamically
- Windows: DLL – dynamic link library
- Save space
  - Just one copy in the memory
- Removing bugs
  - Just make one correction in the shared library
- Save recompilation time
  - Library change do not require new linking

7

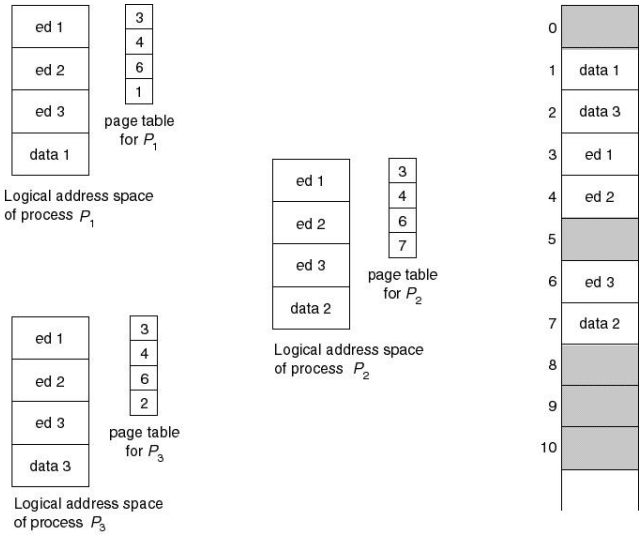
Shared pages and libraries

- Several processes use, just one copy in memory
  - Shared pages read-only
  - Same page frame pointed by multiple page tables
- Code must be position-independent
- Code must be re-entrant, does not change during execution
- A special case of memory-mapped files



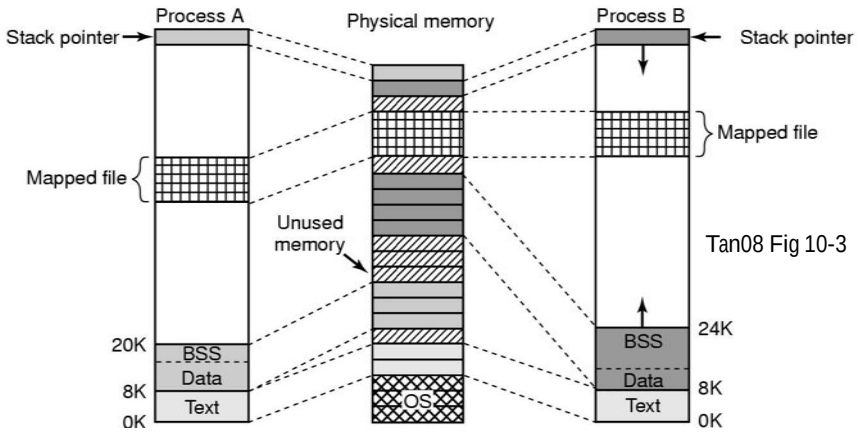
8

Sharing: example editor



9

Sharing a memory-mapped file



- Mapped file is an alternative of I/O, file is accessed as a big character array in memory
- Can be used as shared memory between processes

10

Operating System Involvement with Paging

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

11

Page Fault Handling

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
6. OS schedules disk operation to bring new page in from disk
7. Page tables updated
8. Faulting instruction backed up to when it began
9. Faulting process scheduled
10. Registers restored
11. Program continues

12

Locking Pages in Memory

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

13

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

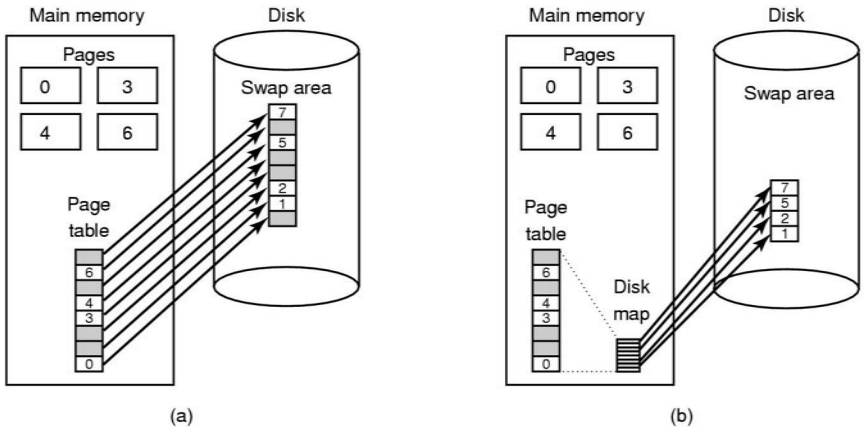
Swap location:

Windows: pagefile.sys, win386.swp

Linux: swap partition

14

Backing Store / Swap area



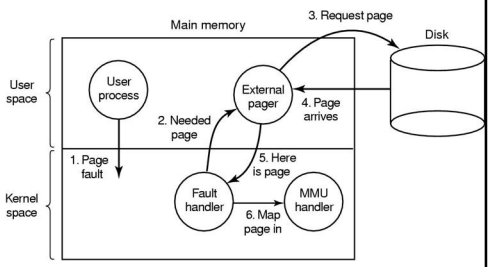
(a) Paging to static swap area

(b) Backing up pages dynamically

15

Separation of Policy and Mechanism

- Advantages:
  - Way to manage complexity
  - More modular code
- Disadvantages:
  - Split between user and kernel spaces
  - Extra overhead
  - More mode switches
  - More message passing

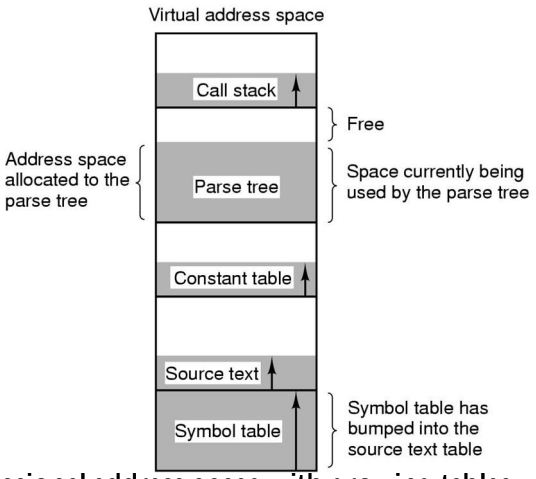


- External pager in user mode
  - Page mapping, swap area
- Page fault handler in kernel
  - Ask for page from external pager

16

Segmentation

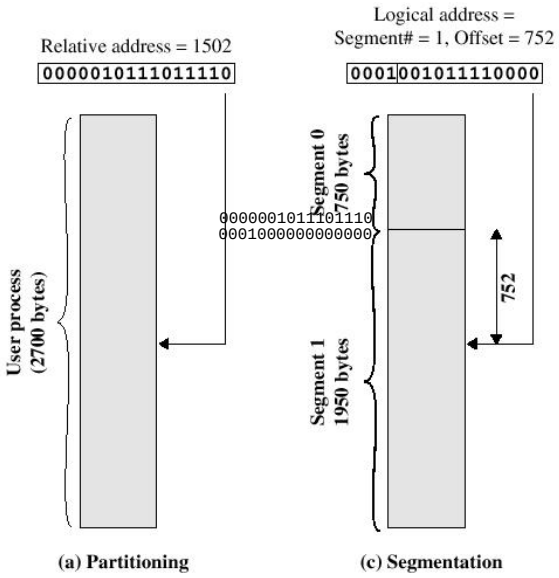
Segmentation



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

18

Segmentation

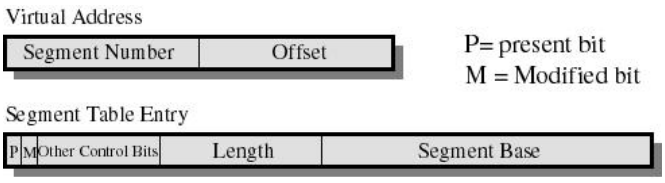


- Programmer (or compiler) determines the logical, unequal-sized segments
- Segments size can change dynamically
- Segment length must be known
- Still using logical addresses (segment, offset)
- Segments can be freely located by OS
- OS maintains a segment table for each process

Sta Fig 7.11

19

Segment table entry



Sta Fig 8.2b

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

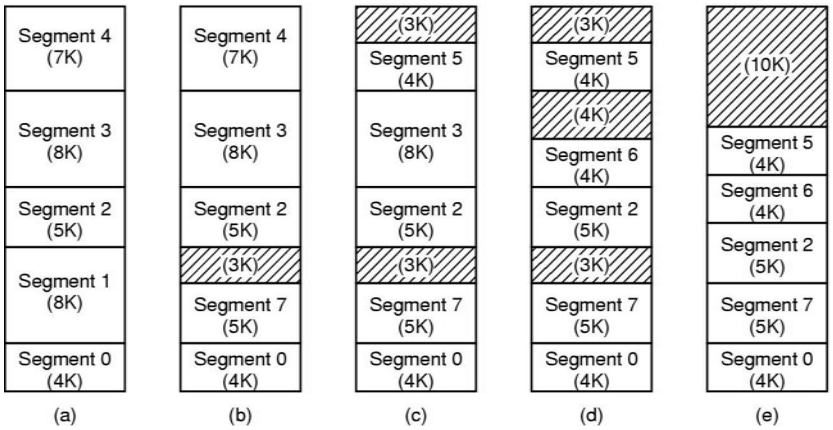
20

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

21

Example of Pure Segmentation



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

22

Comparison of paging and segmentation

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

23

Combining segmentation with paging

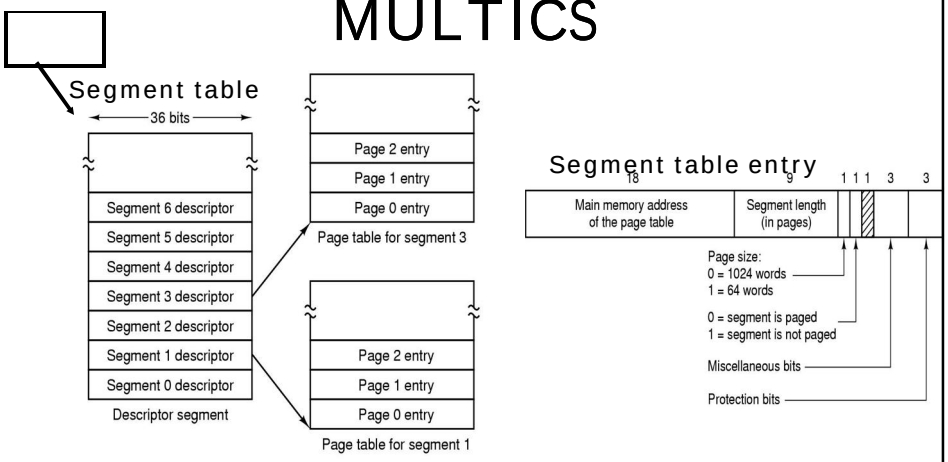
24

Segmentation with paging

- Segments split to pages
  - Memory management easier by pages
  - No external fragmentation *(ulkoista pirstoutumista)*
  - No compaction *(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 page entries to segment's page table

25

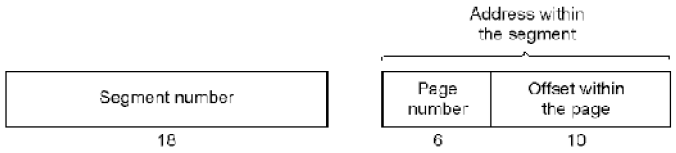
Segmentation with Paging: MULTICS



- Each segment table element points to page table
- Segment table still contains segment length

26

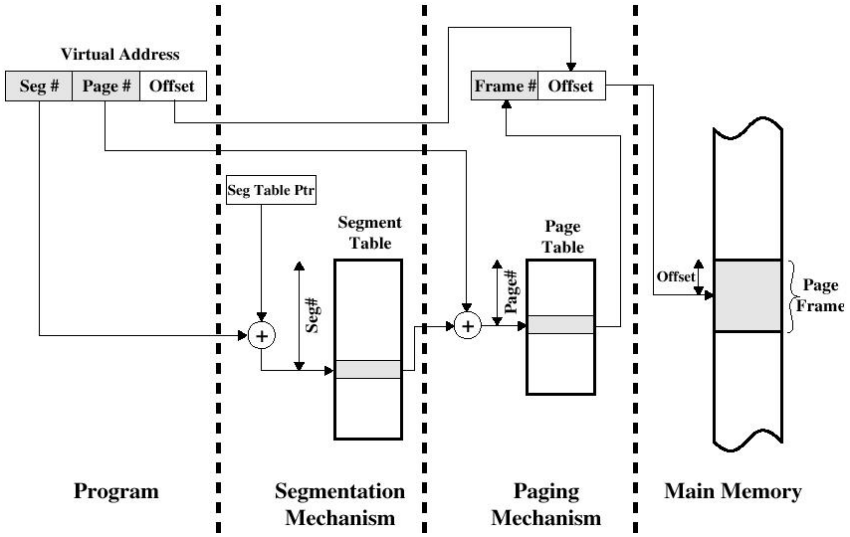
Segmentation with Paging: MULTICS



- 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

27

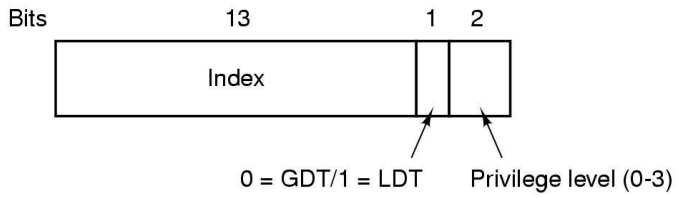
Address translation



Sta Fig 8.13

28

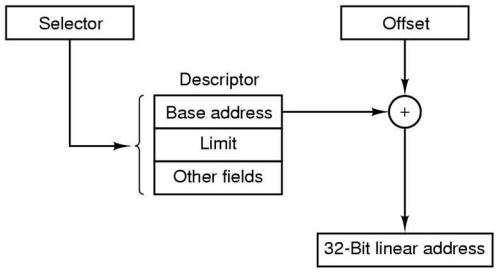
Segmentation with Paging: Pentium



- Pentium has two tables: LDT and GDT
  - At page access selector informs the CPU which one it is
  - Can contain 8K segment descriptors
- Local Descriptor Table (LDT) – one for each process
- Global Descriptor Table (GDT) - one, shared by all

29

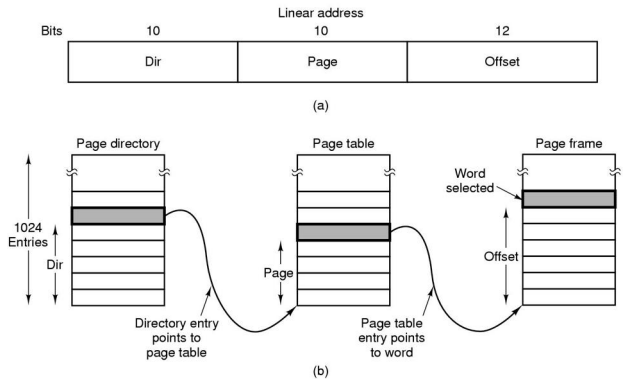
Segmentation with Paging: Pentium



- 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 enables
    - Pages within the segment

30

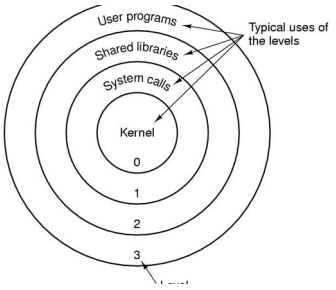
Segmentation with Paging: Pentium



- 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

31

Segmentation with Paging: Pentium



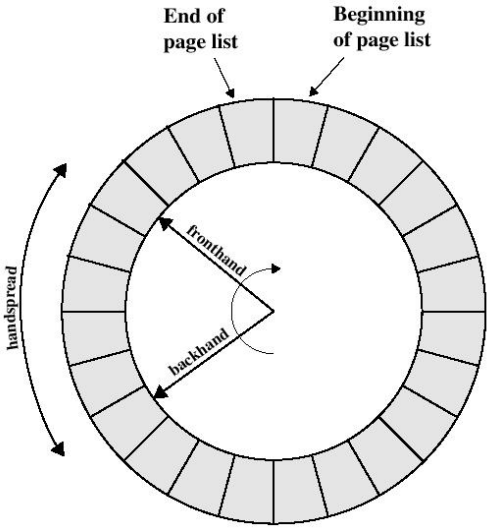
- Protection on the Pentium based on levels
- Each process and segment has protection level
- Process on one level
  - Can access segments on the same and higher levels, but not on lower level
  - May call procedures on different level using selector (call gate) instead of address

32

UNIX / Solaris (+4BSD) Memory management

33

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

34

Linux
Memory management

35

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

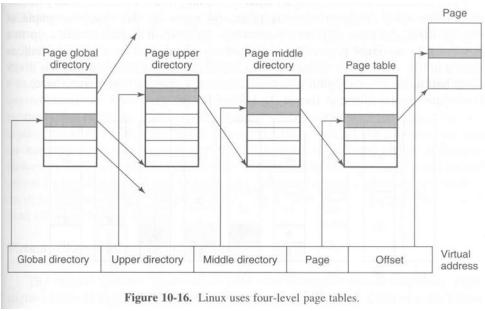


Figure 10-16. Linux uses four-level page tables.

Fig 10-16

36

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

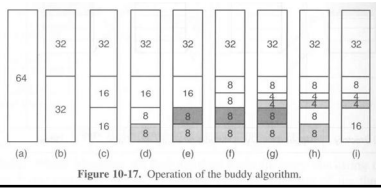


Figure 10-17. Operation of the buddy algorithm.

37

Windows
Memory management

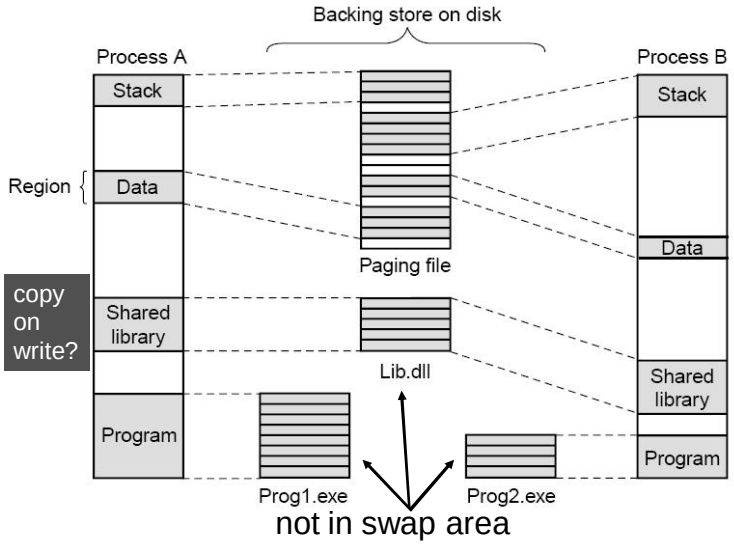
38

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

39

Windows address space

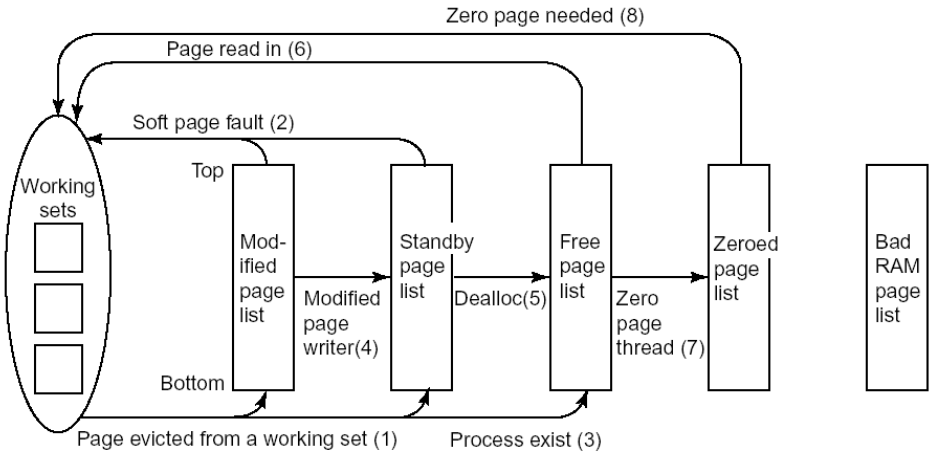


Tan08
Fig 11-32

[Tane01]
Fig. 11-24. Mapped regions with their shadow pages on disk. The *lib.dll* file is mapped into two address spaces at the same time.

40

Windows: page frames



(Fig 11.36 [Tane08])

41