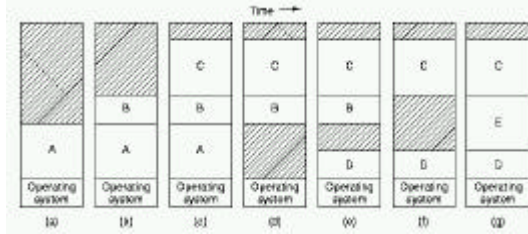


Multi-Programming(*)

- Variable Partitions

- Process is allocated as much memory as it requires and no more
 - Dynamic allocation as processes are created, terminated and swapped in/out



• Figure 4.3, osid2: Memory allocation changes (shaded memory is unused)

- How to allocate unused blocks to processes? (placement strategy)
 - More than one block of unused memory can be used, which one?
 - **First fit**: whichever unused block we examine first is big enough → simple
 - **Best fit**: examine all unused blocks and find the smallest one that can be used → expensive
 - Also *next-fit*, *quick-fit* and *worst-fit* and the *Buddy algorithm*
 - Merging unused blocks
 - Adjacent unused blocks can be merged into one unused block
 - e.g. if 128K and 256K unused blocks are adjacent → single 384K unused block
 - requires manipulation of data structures used to detect adjacent free blocks

3/02/00

COS214, Roberto Togneri, E&E Eng, Univ. of Western Australia

3.7

Example System

Total main memory is 2160K

P1 needs 600K and runs for 10 units of time == (600,10)

P2 = (1000,5) ; P3 = (300,20) ; P4 = (700,8) ; P5 = (500,15)

- P1,P2,P3 are initially created: [600 P1 | 1000 P2 | 300 P3 | 260 Fr]
- With RR scheduling ($q=1$) P2 finishes at $t=14$ and P2 memory released
[600 P1 | 1000 Fr | 300 P3 | 260 Fr]
- With best-fit P4 is swapped into memory into 2nd unused block
[600 P1 | 700 P4 | 300 Fr | 300 P3 | 260 Fr]
- At $t=29$, P1 finishes: [600 Fr | 700 P4 | 300 Fr | 300 P3 | 260 Fr]
- P5 is swapped into memory into 1st unused block
[500 P5 | 100 Fr | 700 P4 | 300 Fr | 300 P3 | 260 Fr]
- P4 finishes:
[500 P5 | 100 Fr | 700 Fr | 300 Fr | 300 P3 | 260 Fr]
- Adjacent unused blocks are detected and merged:
[500 P5 | 1100 Fr | 300 P3 | 260 Fr]

VA to PA mapping: Example (*)

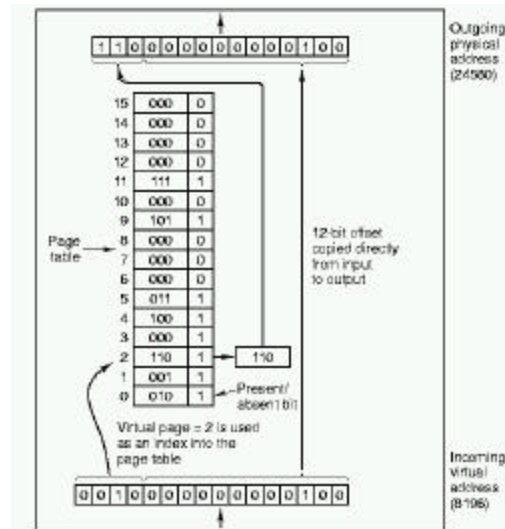


Figure 4.9, osid2: Example of VA to PA mapping

3/02/00

COS214, Roberto Togneri, E&E Eng, Univ. of Western Australia

3.22

VA to PA mapping example

4K pages $\rightarrow d = 12$ (since $2^{12} = 4096$)

64K of VA $\rightarrow p + d = 16$ (since $2^{16} = 65536$) $\rightarrow p = 4$

32K of PA $\rightarrow f + d = 15$ (since $2^{15} = 32768$) $\rightarrow f = 3$

\therefore 16 entry page table needed ($2^p = 2^4 = 16$)

Each entry = [3-bit frame address | P] = 4-bit entry

Process issues instruction: MOV REG,8192

VA = 8192 = 001000000000100 \rightarrow [p = 0010 | d = 00000000100]

\therefore p = 2 and entry #2 from page table yields P = 1 and f = 110
page frame #6 is being referenced

PA = [f = 110 | d = 00000000100] = 24576

MMU mapping produces: MOV REG,24576

Multi-Level Page Tables (*)

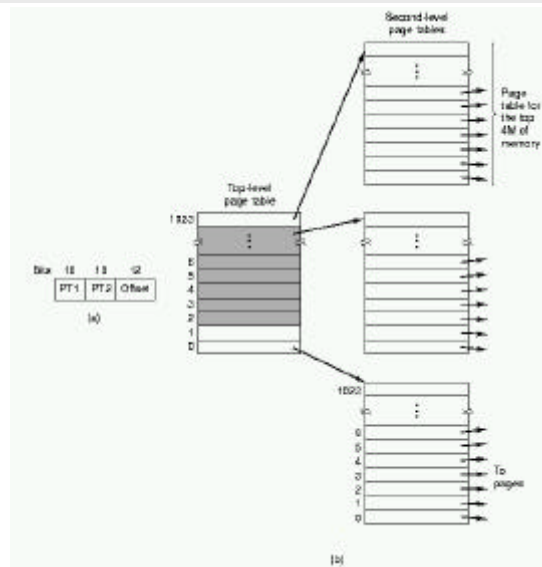


Figure 4.10, osdi2: 2-level page table example

3/02/00

COS214, Roberto Togneri, E&E Eng, Univ. of Western Australia

3.27

Multi-level page table example

4K pages $\rightarrow d = 12$ and 32-bit CPU $\rightarrow p+q+d = 32$

$\therefore p+q = 20 \rightarrow$ let $p = 10$ and $q = 10$

Top-level page table has $2^p = 2^{10} = 1024$ entries and there are 1024 possible 2nd-level page tables. Each 2nd-level page table entry spans $2^{q+d} = 2^{22} = 4\text{MB}$ of VA

A 2nd-level page table has $2^q = 2^{10} = 1024$ entries and each entry returns a 4K page frame

Say VA = 00403004H $\rightarrow p = 0000000001$, $q = 0000000011$, $d = 0 \dots 0100$

$\therefore p = 1 \rightarrow$ get entry #1 from top-level page table

\rightarrow returns pointer to #2 2nd-level page table (VA range 4M - 8M)
(entry #n returns pointer to #(n+1) 2nd level page table)

$q = 3 \rightarrow$ get entry #3 from #2 2nd-level page table \rightarrow returns f

Process needs 12 MB of memory as follows:

4MB for text + 4MB for data + 4MB for stack

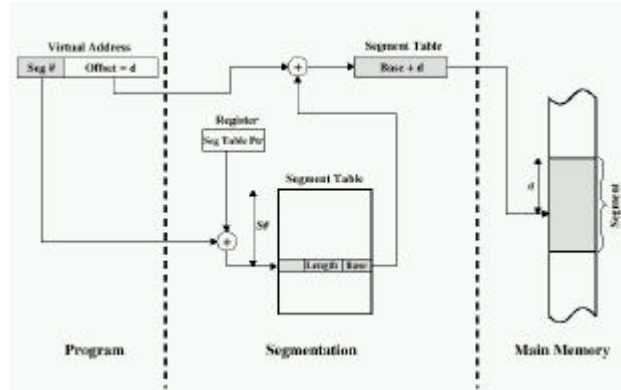
text+data is located at the bottom 8MB of the 4GB VA range

stack located at top 4MB of 4GB VA range

Only need to keep top-level page table and 2nd-level page tables numbers 0,1 and 1023 \rightarrow 4 page tables \rightarrow 4 x 1024 entries \ll 1 million!

Pure Segmentation (*)

- $SA = [s \mid d]$ and $PA = [Base + d]$
 - s -bit used to reference entry in **segment table**
 - 2^s entries in segment table; Segment table entry = [length | base]
 - *Length (limit)*: specifies the maximum size of the segment
 - *Base*: start address of segment in main memory
 - If $(d > \text{length})$ then “segmentation fault” else $PA = (Base + d)$



• Figure 8.10, OS3e: Address translation in a segmentation system

3/02/00

COS214, Roberto Togneri, E&E Eng, Univ. of Western Australia

3.31

Pure Segmentation Example

Segment #	Length (limit)	Base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

$SA = [s = 2 \mid d = 53]$

Entry 2 from segment table \rightarrow Base = 4300, Limit = 400

Since $(d = 53) < (\text{Limit} = 400) \rightarrow$ reference within segment, OK!

$PA = \text{Base} + d = 4300 + 53 = 4353$

Page Fault Handling (*)

- If page table or TLB has $P = 0$ then a page fault occurs
 - Requested page must be fetched from the swap disk (or created if new)
 - A free page frame in physical memory is needed or an existing page needs to be evicted
 - Page Replacement Policy: Which page should be evicted?

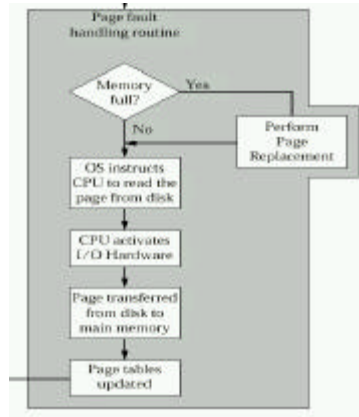


Figure 8.6, OS3e: Page Fault Handling

3/02/00

COS214, Roberto Togneri, E&E Eng, Univ. of Western Australia

3.36

Updating Page Table

Process q experiences a page fault when attempting to access virtual page m . Say the page replacement algorithm selects virtual page n (page frame f) from process p to be replaced (if modified this page must be *paged out* to disk). The virtual page m of process q is then *paged in* to page frame f .

Change to the page table for process p :

The (page n , frame f , $P = 1$) entry in the process table is changed to (page n , frame $?$, $P = 0$)

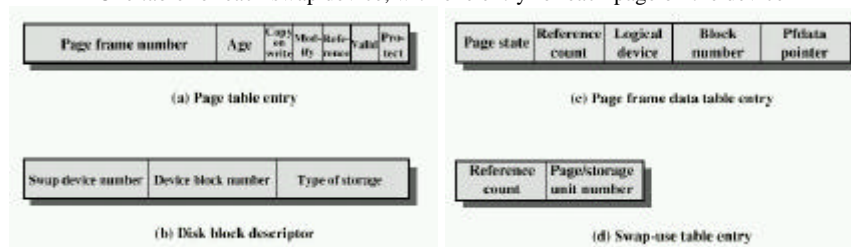
Change to the page table for process q :

The (page m , frame $?$, $P = 0$) entry in the process table is changed to (page m , frame f , $P = 1$)

The instruction is then re-started.

Case Study: UNIX System V (*)

- Paged Virtual Memory Data Structure Formats
 - Page Table
 - One page table per process
 - Disk Block Descriptor
 - One entry for each process page describing where on disk swap the page copy is kept
 - Each process needs to keep track of the copies on swap for (page in) and (page out)
 - Page Frame Data Table
 - Describes each frame of real memory and indexed by frame number
 - Used to keep track of free pages and identify I/O (locked) pages
 - kernel forces page release in order to keep a pool of free pages
 - Swap-use Table
 - One table for each swap device, with one entry for each page on the device



• Figure 8.20, OS3e: *UNIX SVR4 Memory Management Formats*

3/02/00

COS214, Roberto Togneri, E&E Eng, Univ. of Western Australia

3.48

Age: how long in memory without being referenced

Copy-on-Write: refer to Windows NT case study

Modify: the M bit

Reference: the R bit

Valid: the P bit

Protect: is writing allowed (read-only or read-write)